The Software Tester's "*All-You-Need-To-Know*" Action Guide

This book may be freely distributed, redistributed, passed along, given, taken, bequeathed, inherited, or otherwise transferred from one party to another (or to many) so long as NO ALTERATIONS are made to its content, links, graphics, format, etc.

Limits of Liability / Disclaimer of Warranty:
The author and publisher of this book and the accompanying materials have used their best efforts in preparing this program. The author and publisher make no representation or warranties with respect to the completeness of the contents of this program. They disclaim any warranties (expressed or implied) or merchantability. The author and publisher make no guarantee of employment. The author and publisher shall in no event be held liable for any loss or other damages, including but not limited to special, incidental, consequential, or other damages. This material should not be used in place of competent legal, tax, accounting, or other professional expertise.

This manual contains material protected under International and Federal Copyright Laws and Treaties. Any unauthorized reprint, resale, or use of this material is prohibited.

Resource Links

Throughout this book you will see **bold, blue, underlined text. All text like this indicates a clickable link (except, of course, this one).** These are links to more information and/or resources that have been specifically included to help you overcome any challenges to master the skills in this book.

This eBook is my personal eBook and I provide it to you for free. I am grateful to be of service and provide you my real-world experience in how to become a Software Quality Assurance professional and succeed in the high-tech industry!

I do this proudly and I support this book by using affiliate marketing. Please be aware that if you buy through some of the links given in this eBook, I receive a small commission as a thank you for my support to the providers of those products. These small commissions continue to fund my writing, allowing me to offer this book, as well as future installments, to you for free.

Any products mentioned are recommended because I truly believe in their quality and value to you.

Thank you for your support.

Also, please remember as you read through this book that if you find anything confusing, **you can submit your questions to me. If anything is unclear, or you need more information – let me know.**

I want to enable you to create the career of your choice in the high-tech industry. This book is just the first step. I will do all I can to empower you to have control over your future in the Software Quality Assurance field.

The information in this book takes more than a single reading to understand. It takes practice and repetition to master the habits and principles. It takes time to create true mastery – if you think you are worth it, you will get there!

All the information you need is here at your fingertips. Read, absorb, practice, review, gain insight, continue to practice, continue to improve, gain more insight…***you will become an Elite Software Quality Assurance tester before you know it.***

© Phillip Bailey - All Rights Reserved

Acknowledgements

The task of leadership is not to put greatness into people, but to elicit it, for the greatness is there already.
~ John Buchan

This book would not have been possible if not for the influence of countless people on what we shall call, for the sake of this book, my life. My journey, my career through Quality Assurance has been colored by many personalities, innumerable teachers, near impossible tasks, improbable expectations, valued mentors, and questioning colleagues.

Without them I would most probably not have developed the courage, nor experienced the necessary frustration to write this book. For that, and more, I am now and will always be thankful.

I want to thank those that most affected my life in QA and beyond. To those that embodied the highest integrity, unquestioned accountability, and dedication to excellence, I have the deepest admiration and the most profound respect. Thank you for carrying the standard, for doing whatever it took, and for believing.

Thank you to Adam, Casey, Brad, and Elias. To Elaine, Jimmy, DJ Jonas, Rod, Tulio, Sean, Beth, Deirdre, and Big Ebu. To Debbie, Debra, and Julian, to Don, Dennis, and Drewsky, to Greg, Todd, and Che. To Alicia, Randy, and Gary, to Dan, Chuck, and to Lilly.

Thank you to Kevin, James G., Jason H., Mike H., Joshua, Andy, Minh, Faithlyn, Ben, Robert H., and Mike F. Thank you to Andrew, Mike D., Stan, Albert, Nate, Grant, Brennan, Maria, and Jhoey. Thank you to Jen, Srini, Chad S., Peter, Richard, Renee, Adam, Tom, and Alex.

Thank you Robert E., David B., Doug, Andre, Ruth, and Richard. Thank you for your valuable insights, advice, and friendship.

To James, Jason, Jude, Tim, Vlad, Chad, Syruss, and Steven, from the bottom of my heart, I say thank you. You all left me better than you found me.

© Phillip Bailey - All Rights Reserved

And thank you to my first guide in QA who nurtured my questioning attitude, taught me to understand a program's weaknesses, and rewarded my propensity to be hyper-critical; Phred, wherever you are…Thank you.

Thank you all…You taught me well.

Thank you Josh for all of your valuable feedback, insight, and perspective. This book would be much less organized and leave far more to be desired without your input.

And most of all, to my beautiful and wonderful wife Kristiina, without whom I would still be wandering in the cold, dark wilderness. Thank you from the bottom of my heart. This book would not have been possible without your love and support.

The reasonable man adapts himself to the world; the unreasonable one persists in trying to adapt the world to himself. Therefore all progress depends on the unreasonable man.
~ George Bernard Shaw

Introduction

Greetings! Welcome to the wild and wonderful world of Software Quality Assurance testing. Whether you are looking to start a career in the testing field or if you are a tester that wants to take your career to the next level, you have come to the right place.

This book will give you the information you need to join an elite group of Quality Assurance Professionals. This book is full of information that I wish I had had when I began testing back in the late 1990s. This is a book that will guide you through the day-to-day work you will perform and show you how to excel as a tester.

All of the information in this book comes from my own first-hand experience. Every single lesson I detail in the book has been used by me and by others to bring success. Each action listed in this book has been proven to yield high-value results. This book is geared specifically for the front-line Quality Assurance tester who is doing real work every day.

Using the experience that I gained in personally training over 200 testers, I provide you the most impactful, valuable set of tools that you will need to become an effective tester in the shortest amount of time. I highlight the time-wasting activities that prevent other testers from succeeding.

I warn you of the pitfalls and show you what actions to take to differentiate yourself. I deliver to you the most effective lessons that have seen tester after tester achieve great success.

I have condensed the most high-value daily activities that you can perform as a Quality Assurance tester. Master the principles compiled in this book and you will significantly increase your odds of success.

In this book I give you actionable, step-by-step directions that will allow you to excel. You have in front of you right now the tools you need to be more than just a tester. You can be a true Quality Assurance Professional that helps in the emergence of the Innovation Economy.

© Phillip Bailey - All Rights Reserved

I have assembled the keys, steps, actions, and skills you need and reduced them into their most manageable elements. I have simplified the essential building blocks you need to quickly succeed in the testing field.

I hope that this book helps you avoid many of the mistakes others have made and propels you to the success you desire. I look forward to witnessing your influence on products yet to ship, services yet to launch, and efficiencies yet to be born.

Re-examine all that you have been told . . . dismiss that which insults your soul.
~ Walt Whitman

© Phillip Bailey - All Rights Reserved

Practice What You Have Learned

There are five (5) intentional, non-informational errors in this book…

If, after reading it, you decide to test yourself…

Identify as many of the errors as you can and…

Write each error as an individual bug and

Submit them to me…

I will respond with a professional critique of your work and provide feedback on areas on which you can improve…

© Phillip Bailey - All Rights Reserved

RESOURCE LINKS **2**

ACKNOWLEDGEMENTS **3**

INTRODUCTION **5**

PRACTICE WHAT YOU HAVE LEARNED **7**

WHAT IS SOFTWARE QUALITY ASSURANCE? **12**

WHY DID I WRITE THIS BOOK? **16**

WHAT IS IN THIS BOOK **17**

CHAPTER #1: WHY QUALITY ASSURANCE? **19**

LOCATION! 19
FINDING YOUR NICHE! 20
BE CUTTING EDGE! 21

CHAPTER #2: INTRODUCTION TO SOFTWARE DEVELOPMENT **23**

METHODS: WATERFALL VS. AGILE 23
TESTING: BLACK BOX VS. WHITE BOX 27
TESTING: MANUAL VS. AUTOMATED 30
TERMINOLOGY 32
NUMBERS ARE YOUR FRIEND 34

CHAPTER #3: TESTING: SCIENCE VS. ART **35**

SCIENCE 35
ART 35
THE SCIENCE OF QA TESTING 37
THE ART OF QA TESTING 38

CHAPTER #4: 3 ESSENTIALS SKILLS **46**

ESSENTIAL SKILL #1: TESTING 47
ESSENTIAL SKILL #2: COMPUTERS 61
ESSENTIAL SKILL #3: COMMUNICATION 65
YOUR BEST INVESTMENT 68
ESSENTIAL SKILL RESOURCES 70

CHAPTER #5: THE 5 KEYS WHEN GETTING STARTED **72**

KEY #1: THE TESTING PROCESS 73
KEY #2: HOW TO EXECUTE TESTS AND FIND BUGS 75
KEY #3: WHAT TO DO WHEN YOU FIND A BUG 77
KEY #4: HOW TO WRITE A BUG 82
KEY #5: HOW TO DELIVER TEST RESULTS 85

© Phillip Bailey - All Rights Reserved

CHAPTER #6: 10 SKILLS OF ELITE TESTERS 87

ELITE SKILL #1: COMMUNICATION 89
ELITE SKILL #2: BULLETPROOF BUG WRITING 96
ELITE SKILL #3: MANAGING EXPECTATIONS 101
ELITE SKILL #4: ATTENTION TO DETAIL 103
ELITE SKILL #5: ASKING THE RIGHT QUESTIONS 105
ELITE SKILL #6: BE SOLUTION-ORIENTED AND PROACTIVE 108
ELITE SKILL #7: MASTER YOUR BUGBASE 111
ELITE SKILL #8: DELIVER STELLAR REPORTS 119
ELITE SKILL #9: DELIVER ON YOUR WORD 122
ELITE SKILL #10: NEVER STOP LEARNING 124

CHAPTER #7: THE 10 CRITICAL ELEMENTS OF A BUG 128

ELEMENT #1: BRIEF DESCRIPTION 129
ELEMENT #2: EXPANDED DESCRIPTION 129
ELEMENT #3: REPRODUCIBILITY 130
ELEMENT #4: STEPS TO REPRODUCE 130
ELEMENT #5: RESULT 131
ELEMENT #6: EXPECTED RESULT 131
ELEMENT #7: SEVERITY 131
ELEMENT #8: PRIORITY 133
ELEMENT #9: STATUS 134
ELEMENT #10: ASSIGNEE 136

CHAPTER #8: THE 5 KEYS TO A BULLETPROOF BUG 138

KEY #1: DESCRIPTIONS 139
KEY #2: REPRODUCIBILITY 141
KEY #3: STEPS TO REPRODUCE 145
KEY #4: RESULT 149
KEY #5: EXPECTED RESULT 151

CHAPTER #9: THE 5 MOST COMMON BUG WRITING ERRORS 154

ERROR #1: NOT ENOUGH INFORMATION 155
ERROR #2: UNCLEAR REPRODUCIBILITY 156
ERROR #3: OMITTING THE EXPECTED RESULT 157
ERROR #4: SELECTING THE WRONG SEVERITY 158
ERROR #5: ASSIGNING BUG TO THE WRONG PERSON 159

CHAPTER #10: THE LIFE OF A BUG 160

CHAPTER #11: THE 5 STEPS TO BUG REGRESSION 163

5 STEPS TO BUG REGRESSION 164
STEP #1: KNOW THE VERSION 165
STEP #2: UNDERSTAND THE FIX 168

© Phillip Bailey - All Rights Reserved

STEP #3: VALIDATE APPROPRIATELY....170
STEP #4: COMMENT THOROUGHLY....173
STEP #5: ASSIGN ACCURATELY....176

CHAPTER #12: THE 5 MOST COMMON BUG REGRESSION MISTAKES....178

MISTAKE #1: QA RESOLVES A BUG....180
MISTAKE #2: NON-QA CLOSES A BUG....181
MISTAKE #3: NOT COMMENTING....183
MISTAKE #4: COMMENTING IN THE WRONG PLACE....185
MISTAKE #5: NOT ASKING FOR HELP....186

CHAPTER #13: HOW TO ACE A POST MORTEM WITH 3 ACTIONS....189

ACE A POST MORTEM WITH 3 ACTIONS....190
ACTION #1: PREPARE....191
ACTION #2: PARTICIPATE....193
ACTION #3: PRODUCE....195

CHAPTER #14: THE TOP 5 CHALLENGES OF BEING A QA TESTER....199

CHALLENGE #1: LACK OF RESPECT....200
CHALLENGE #2: IGNORANCE ABOUT QA....208
CHALLENGE #3: POSITION OF QA....211
CHALLENGE #4: LACK OF TOOLS....215
CHALLENGE #5: MINISCULE MARGIN FOR ERROR....217

CHAPTER #15: GETTING A RAISE....220

CHAPTER #16: THE JOB HUNT....223

WHERE DO YOU WANT TO MAKE A DIFFERENCE?....224
HOW DO YOU WANT TO MAKE A DIFFERENCE?....226
PREPARE YOUR RESUME....227
YOUR COVER LETTER....229
PROOFREAD YOUR RESUME!....233
POST YOUR RESUME ONLINE – EVERYWHERE!....234
RESEARCH COMPANIES YOU WANT TO WORK FOR....235
PREPARE FOR YOUR INTERVIEW....236
WHEN YOU INTERVIEW....240
JOB HUNT RESOURCES....245

CHAPTER 17: WANT TO BE A GAME TESTER?....246

CAN YOU BECOME A GAME TESTER?....247
WHAT DOES IT TAKE TO BECOME A GAME TESTER?....249
WHERE TO START?....252
WHAT IF YOU LAND A PLAY TESTING POSITION?....254
WHAT SHOULD A PLAY TESTING REPORT CONTAIN?....255

© Phillip Bailey - All Rights Reserved

DON'T WAIT TO GAIN EXPERIENCE! 260
GAME TESTING RESOURCES (EMPLOYMENT RESOURCES): 263
GAME TESTING RESOURCES (GAMING RESOURCES): 264

CHAPTER 18: YOU'VE BEEN LAID OFF – 265

NOW WHAT? 265

IT'S NO BIG DEAL 267
SEIZE THE OPPORTUNITY! 268
KEEP DEVELOPING YOUR SKILLS 269

EPILOGUE 271

LINKS TO RESOURCES 274

GLOSSARY 277

ABOUT THE AUTHOR 281

SAMPLE STATUS REPORTS 282

STATUS REPORT #1 282
STATUS REPORT #2 283
STATUS REPORT #3 284
STATUS REPORT #4 285

SAMPLE BUGS 286

SAMPLE BUG #1 286
SAMPLE BUG #2 287
SAMPLE BUG #3 289
SAMPLE BUG #4 290
BUG TEMPLATE 291

What is Software Quality Assurance?

Wikipedia.com defines Software Quality Assurance in this way:

> *Software quality assurance (SQA) consists of a means of monitoring the software engineering processes and methods used to ensure quality. It does this by means of audits of the quality management system under which the software system is created. These audits are backed by one or more standards, usually ISO 9000.*
>
> - http://en.wikipedia.org/wiki/Software_Quality_Assurance (5.31.08)

A less technical way of stating it is this:

> *Software Quality Assurance is a systematic process of checking to see whether a product or service that is being developed meets the specified requirements.*

That's all well and good, but **what does it mean to you?** What does a QA tester really do? What will you, as a tester, be relied upon to do day-to-day? This book will explain the process of Software Quality Assurance as it relates to you, the tester. It will detail what you need to know and how you should perform your job to obtain the best results.

A real Quality Assurance Professional does exactly what it sounds like; they "assure" the "quality" of the product they are testing. They accomplish this by performing multiple combinations of tests on the product they are responsible for. This can be software (games, word processing, etc.), hardware (cameras, DVD players, phones, etc.), web sites, databases, fuel injection systems, weapons control systems, medical study software, and the list goes on and on.

Technology is not about to cease being designed and built. And as long as this is the case, it will need to be tested. Otherwise, how will a company know that their products work as intended and that people will like their products and continue to buy them?

Without testing there is no way of knowing whether a player will be able to finish their game, or that the database with information on your 100,000 clients is stable and will store the information from one day to the next, or that when you hit <Send> that your email will actually go where you intended.

© Phillip Bailey - All Rights Reserved

All of this appears almost seamless to the end-user. That their product does what is says it will do is taken for granted, but it doesn't just happen by having good thoughts, strong intentions, or by being a kind and giving person. It happens by what many view as "magic"! It happens by executing a thorough test program, without which a company would have no better idea of whether its product worked than would a person who had never seen the product before.

Unfortunately, many companies staff their QA department with testers that either don't understand what they are doing, are not committed to their work, or both, instead of with Quality Assurance Professionals. This book will guide you in your quest to become one of these Quality Assurance Professionals who carry the Quality Standard and champion products for the consumer.

This book will prepare you to get a job as a tester. This book will give you the tools you need to have an immediate head-start over other entry-level testers. By following the lessons in this book, by mastering the principles, you will soon surpass even experienced QA testers and become an elite Quality Assurance Professional. You will be viewed as one of *those* guys who perform the magic that ensures a company's good name.

So what is this "magic"? This "magic" is QA; properly executed and communicated by a trained Quality Assurance Professional! Here are the basics of how it works:

- Someone comes up with an idea for a product
- Marketing, design, and/or technical specifications are written up (hopefully) so that there is some idea of what the end product should look like, how it should perform, etc. when completed
- Someone begins building the first iteration of the product
- In order to determine whether or not what has been built so far works or not, someone tests it. This is you! This is where QA comes in! This is where the magic happens.

Now it's time for you to unleash your most deadly kung fu! Following the strategy you laid out when you first built your test plans, you spring into action! You attack the product like no other; breaking it in so many ways that those who built it will wonder if they should get their money back for their hard-earned engineering degree. You test, find defects and log them, complete your tests, and deliver a detailed report encompassing the many shortfalls of the product, the number of defects awaiting attention, the amount of effort involved, and your

© Phillip Bailey - All Rights Reserved

projections for the next phase in the life of the product. You now see awe and envy in your colleague's eyes.

Well, that's how it should happen…

If you are interested in sitting on your couch getting paid to play video games, this book can help you more effectively find bugs and report them, but it is not my goal to teach you how to be a game-playing couch potato. Anyone can play a game (that's the point of them isn't it), but only the trained can consider themselves real Quality Assurance Professionals.

Only those who understand the disciplines and methodologies necessary to ensure all issues in a product are known are actually of value in a real QA environment. Anyone can notice that their game isn't doing what they think it should do, but:

- Are they right?
- Is the game really doing something that it shouldn't?
- How can they know what it is supposed to do?
- What do they do once they notice this?
- Can they actually provide useful information about what is wrong so that someone can fix it?
- How will they figure out what made this behavior happen?

Being able to accurately answer these questions is what begins to separate the gamer or run-of-the-mill tester from a true Quality Assurance Professional.

There is nothing wrong with gaming. I am a fan of many games myself. I played them before I began my QA career and continue to play them to this day. But that did NOT make me a tester.

It takes great skill and lots of practice to complete many of the games today, and the gamer is not without value in providing feedback as to what the market wants and whether or not a game has a bug in its final stages where you have to defeat the "big boss". But the game would never see the light of day if not for the dedication of Software Quality Assurance Professionals.

These professionals that devoted the hundreds and thousands of hours to ensuring that the game was playable on a thousand different PCs and Macs, made sure that all levels were reachable as designed when the player did what they were supposed to do, ensured that the game did not erase the computer's operating system when installed, and that when uninstalled that your system was

© Phillip Bailey - All Rights Reserved

truly “free” of the game. This is the job of the Quality Assurance Professional. Not often glamorous, but always necessary!

If you build it they will come?

Maybe, but not necessarily…

But if you build it, it *will* need to be tested!

© Phillip Bailey - All Rights Reserved

Why Did I Write This Book?

I wrote this book because I could not find any "how-to" book on being a Software Quality Assurance tester that showed how to navigate the software industry. There is no step-by-step action guide on the market that details what a tester needs to know on "day one" and shows them how to excel in the software industry today.

There are volumes that describe and debate the various test methodologies to use when creating software, there are libraries that have been written about the proper processes and procedures for software development, and how to successfully lead the software development process. You can find tomes dedicated to explaining the most effective way to compile test metrics. But there is no straightforward, useful guide for the front-line tester that tells them what they need to know and how to do it.

You need to know how to get a job and then how to separate yourself from the pack. You need to know how to become an elite tester. You need to know how to showcase your abilities. That is what this book is about. That is why this book will be so valuable to you.

To become a valuable **Quality Assurance Professional**, you must first know how to test. You must understand why you are performing each test. You must be able to speak to the results of your testing and you must be able to improve your testing each and every day. To do this requires certain skills; skills that are not detailed in any work that I can find on the open market.

The software industry needs Quality Professionals. As you master the lessons detailed in this book, you will have the opportunity to become one of these immensely valuable commodities. You will understand the skills you need to excel.

You will be able to add value to the software development process by going beyond simply following the tests you are given. You can become the member of the team that can find bugs *before* testing begins. The one that asks the right questions resulting in lower development costs. A star on the rise that is preparing to lead a team. You will become a **Quality Professional**!

© Phillip Bailey - All Rights Reserved

What Is In This Book

This book is specifically geared to enable anyone who wants to get into the Software Testing field and be a true Quality Assurance Professional the key information and tools they will need to do so. Whether you have no experience at all or have been testing for years and want to jump-start your career, this book will give you the critical information you need to have when you interview, for your first day on the job, and beyond. It will give you the hands-on insight and first-hand knowledge gained from over 10 years in the industry; what you need to know to get the job done.

This is not some theoretical treatise on the necessity for Quality Assurance testing, how the industry has changed, or a review of the best automation tools. This is a real "how-to" that anyone can use to get a job and become an invaluable member of a QA team. I know what you need to know because I have trained over 200 testers and leads in my years in QA and know what it takes to do the job in the most effective, professional manner possible.

I will show you where the rubber meets the road. I will teach you what real day-to-day QA is in today's workplace and how to excel at it. I will give you the resources you will need to continue to enhance your knowledge. I will show you what a QA Lead or Manager is looking for in a tester and how you can most effectively meet those needs.

This book will detail:

- The **3 Essential Skills** needed to begin
- The **5 Keys** When Getting Started
- The **10 Skills** of Elite Testers
- The **10 Critical** Elements of a Bug
- The **5 Keys** to a Bulletproof Bug
- The **5 Most Common** Bug Writing Errors
- The **Life of a Bug**
- The **5 Steps** to Bug Regression
- The **5 Most Common** Bug Regression Mistakes
- The **3 Actions** to Ace the Post Mortem
- The **Top 5 Challenges** of Being a QA Tester

© Phillip Bailey - All Rights Reserved

For the sake of simplicity, I will teach you about bugs in five different sections of this book. My intention is to introduce you to the information you need as you need it so that you will have a better chance of retaining it the very first time.

Here is what is contained in each section:

- **Bulletproof Bug Writing** – In the Skill #2: Bulletproof Bug Writing section, I explain the actions you must take, **apart** from writing the bug, in order to guarantee that the bugs you write are of the highest quality possible
- **The 10 Critical Elements of a Bug** – In this section you will be given an overview of the anatomy of a bug. We define the 10 elements that must be used in order to create a high-quality bug.
- **The 5 Keys to Create a Bulletproof Bug** – In this section we go in-depth to explain the detailed information you need when writing a bug. What should you write, how you should write it, and why is all covered in this section. Disregard these lessons at your own peril
- **The Life of a Bug** – Here we look at the lifecycle that a bug follows. Using the flowchart as reference, I show how a bug's life flows from initial entry to finally being closed
- **Bug Regression** – Once a bug has been fixed, it must be tested to verify the fix. This is known as Regressing a Bug. In this section I detail the keys you must follow when you are performing this Regression testing

Learn the lessons in this book and you will be well on your way to a rewarding career as a Quality Assurance Professional. You can do this without having an engineering or computer science degree. I know. I don't have either of those; instead I got a job as an entry-level tester and rose to become an extremely effective QA Project Manager. It can be done!

If you master the principles put forth in this book, you will have become one of the truly great testers and well on your way to leading a team of testers that will excel in any endeavor they undertake! Just master the fundamentals and you can do anything in this industry!

© Phillip Bailey - All Rights Reserved

Chapter #1: Why Quality Assurance?

To err is human, but to really foul things up you need a computer.
~ Paul Ehrlich

Location!

Do not go where the path may lead, go instead where there is no path and leave a trail.
~ Ralph Waldo Emerson

Think about it: The opportunities afforded you as a true Quality Assurance Professional are global in nature, cutting-edge in practice, and ever-changing due to their environment. If you are a true QA Professional, you have opportunities to ply your trade throughout the United States and the world.

Technological development is taking place all over the world! There is need for

Quality Assurance Professionals everywhere that this development is happening. In the U.S. on the west coast, testers are needed in the Silicon Valley and the San Francisco Bay Area, in the greater Los Angeles area, in Portland, Oregon, in Seattle, Washington, and even in Redmond, Washington – where there is a fledgling company you may have heard of by the name of Microsoft. Tech development needs testers in Oklahoma City, in St. Louis, in Michigan, in Minnesota, in Texas, in Florida, in Maryland, and in Washington D.C.

I have worked with companies based in Japan, Israel, Germany, Hungary, France, and England. I have worked for companies that have contracted their testing out to companies in Canada, Romania, and Argentina.

Tech development is taking place right now all over the globe; in Brazil, in India, in Japan, in Australia, and in Europe. All of this development requires testing. If you become a Quality Assurance Professional, opportunities throughout the world will be available to you. Would you like to travel?

© Phillip Bailey - All Rights Reserved

Finding Your Niche!

I can't believe that God put us on this earth to be ordinary.
~ Lou Holtz

Do people tell you that you nitpick? Boy, do I know a place you can shine!

Are you the kind of person that is "overly" critical? Do you see what is wrong and needs to be fixed? I know a job that you may find quite rewarding.

Do people tell you that you spend too much time analyzing everything? How would like to make money doing that analysis and have people appreciate it?

Do you like to be behind the scenes? Do you have a passion to know how things work? Are you the kind of person that has a passion to be in-the-know? What if you could work on projects long before they are ever available to the consuming public?

Do you or other think of you as a gadget freak? Do you love to have the latest cell phone, camera, PDA as soon as it's on the market? Would you like to test those gadgets and add your input about what features they should have and how they should work?

What if I told you that the traits that people say are your personality flaws could be turned into strengths? What if people actually praised you for being hypercritical? Would that type of job appeal to you?

© Phillip Bailey - All Rights Reserved

Be Cutting Edge!

It is never too late to be what you might have been.
~ George Eliot

When you work in the tech sector as a tester, you get to see the latest and greatest products before they are ever released to the public! You will have insight into how and why things work at a level most people never even consider. You will be "in-the-know".

New testing opportunities appear every day! Whether the economy is booming or in recession, new technology continues to be developed. It has to in order to meet the ever-changing demands people have for what they want, how they want to do it, and to make their daily lives easier and more fulfilling. Where there is technology, there is a need for testing!

Just a small sample of the technologies that need testing right now, today:

- Web Sites
- Web-based applications
- Digital cameras
- DVD players
- Mp3 players
- Cell phones
- Video games
- Software for running pharmaceutical drug trials
- Educational software
- Seismic sensors with thermal stabilization
- A host of emerging Green technologies that the public will be clamoring for
- And, of course…computers!

The high tech field is huge! New ideas that will make the world a better place are being researched, funded, and created every single day.

© Phillip Bailey - All Rights Reserved

Not all of these companies will survive, but they will need to know if their product is viable – this requires testing. ***Testing that you can do!***

Quality Assurance is not the sexiest or glamorous job, but it is certainly interesting and is a field that I have found immensely rewarding. By mastering the principles in this book, you will be able to enter into and excel in this field. Once you do, I hope that you find it as rewarding as I have.

© Phillip Bailey - All Rights Reserved

Chapter #2: Introduction to Software Development

Methods: Waterfall vs. Agile

Software Development is generally categorized into one of two methods: Agile and Waterfall

I will give an overview of both methods. Keep in mind that all of the principles in this book are applicable to both, but for simplicity, the examples that I use in this book represent more of the Waterfall testing method.

If you want to delve further into these methodologies, there is a great deal of information on the internet. Much of it is even useful and accurate. The links at the end of this section will give you a start if you would like to continue your study.

Agile Method

Agile testing methodology focuses around small, iterative development cycles. The entire development team works in very close physical proximity to one another, focusing on use cases. This allows smaller, more frequent builds to be released to the public.

The development team creates only what the client specifies and is able to modify the feature set on the fly to meet the client's needs. In this way the project is not burdened with code that may never be used and can respond to changes to the feature set quickly without having to rework thousands of lines of code.

In Agile, instead of coding for every possibility, the testing focus is on modular test cases that can be added or subtracted with greater ease. Agile allows for more flexibility in the feature set to be released at any given time without causing a severe ripple effect on the schedule.

© Phillip Bailey - All Rights Reserved

As a Quality Assurance Professional working within an Agile framework, you will most probably be sitting with the engineers, producers, and other invested team members instead of sitting off in a QA lab. This enables the entire project team to communicate quickly and easily. This can be very beneficial to the health and speed of a project.

It is not, however, necessary to all sit together in order for a project to maximize the efficiencies available to it. I will show you in this book how to facilitate this communication without sitting right next to the rest of the project team.

We are uncovering better ways of developing software by doing it and helping others do it. Through this work we have come to value: Individuals and interactions over processes and tools, working software over comprehensive documentation, customer collaboration over contract negotiation, responding to change over following a plan.

~ Agile Software Development Manifesto

Waterfall Method

A more traditional method of developing software is known as the Waterfall Method. In this method, the project is planned as thoroughly as possible before any development occurs. At least, that is the goal.

When the builds are in their earliest state, there are only a few features and assets. As the testing progresses, fixes are made to the code and more features are added. This continues until the product reaches a functioning state with all of the features planned for it. At this point the product is ready to be delivered.

In a Waterfall framework, projects are scheduled around well-established milestones. The earliest builds delivered for testing are often defined as Feature Drops or Pre-Alpha.

Once specific milestone criteria are met, like all of the assets are in place or the program has reached a pre-specified state of functionality, then the software has achieved the Alpha milestone. All builds delivered are designated as Alpha builds.

As the software continues to evolve and achieve further pre-defined milestones, builds are designated Beta. Beta builds are considered to be feature complete.

© Phillip Bailey - All Rights Reserved

This does not mean that all of the features work as planned, but that they are present. Also, a Beta build may be released to a limited number of public users in an effort to get their feedback and more accurately fine tune the product.

After Beta, the software gets the Release Candidate designation. In this phase, the goal of the builds is to be ready for shipping. QA may show that the delivered builds are not yet ready to ship due to the number and/or severity of bugs remaining in the program. Engineering will endeavor to deliver builds that are a "ship-ready" as possible.

Some companies use a designation after Release Candidate that is known as Final Candidate. This refers to a Release Candidate that is expected to definitely be ready for public consumption.

Regardless of what all of the milestones and designations are during the creation, testing, and fixing of the product, once it meets the pre-defined criteria (functionally, aesthetically, etc.) a build is deemed a Gold Master. This is the build that will be shipped.

In achieving Gold Master status, a build has met all of the functional, usability, asset, content, and quality standards that were set in place when the project was planned. There are no Active bugs logged against the product (all bugs are either Closed or Deferred) and QA is ready to give their blessing to the product.

Methods: Waterfall vs. Agile

So that is an overview of Agile and Waterfall testing methods. Which do I prefer? Which is better? What is the best use of each?

In my experience, no project is really one extreme or the other. All projects that I have worked on in the more than 10 years I have been in QA have been a synthesis of both Agile and Waterfall methodologies. The key for you is to understand the strengths of both, know how to adapt to whatever version of them you encounter, and be able to use the strengths of each to their fullest.

If you can adapt to and use the strengths of both methodologies, you will become a masterful Quality Assurance Tester. You will be well on your way to becoming an invaluable team member no matter the project.

The method changes, even slightly, with each project. Learn the principles.

© Phillip Bailey - All Rights Reserved

Useful Agile Links:

Agile Information on Wikipedia: http://en.wikipedia.org/wiki/Agile_software_development
The Agile Manifesto: http://agilemanifesto.org/
The Agile Alliance: http://www.agilealliance.com/

Useful Waterfall Links:

Waterfall Information on Wikipedia: http://en.wikipedia.org/wiki/Waterfall_model
PowerPoint Description of Waterfall:
http://faculty.winthrop.edu/dannellys/csci626/07_Life_Cycles.ppt
Dr. Winston Royce: http://www.cs.umd.edu/class/spring2003/cmsc838p/Process/waterfall.pdf

© Phillip Bailey - All Rights Reserved

Testing: Black Box vs. White Box

There are two major products that come out of Berkeley: LSD and UNIX. We don't believe this to be a coincidence.
~ Jeremy S. Anderson

Software Quality Assurance testing is generally categorized into two testing types: White Box testing and Black Box testing.

Wikipedia.com defines Black Box Testing as:
Black box testing takes an external perspective of the test object to derive test cases. These tests can be functional or non-functional, though usually functional. The test designer selects valid and invalid input and determines the correct output. ***There is no knowledge of the test object's internal structure.*** (emphasis mine)
- http://en.wikipedia.org/wiki/Black_box_testing (5.31.08)

As a Black Box tester, the idea is that you don't have to have an understanding of the program code. You don't have to be able to read code and you don't need to be able to write code. What you must be able to do is "test the black box"; the "box" being whatever program, widget, or doodad is placed in front of you.

What this means to you, as a tester, on a day-to-day basis is that even though you don't have all the details of each nook and cranny of the code, you must gain an understanding of how the program works. You must be able to exercise the code through the user interface (UI) as thoroughly as you could if you had access to the code.

This is a skill that takes time and effort to develop, but it can be developed. Through practice, observance, awareness, patience, improvement, and perseverance you can develop the necessary tools to excel at this skill. Even the greatest of testers have all had to learn and develop this skill, no matter how unnatural it felt the first time.

© Phillip Bailey - All Rights Reserved

Wikipedia.com defines White Box Testing as:
*White box testing (a.k.a. clear box testing, glass box testing or structural testing) uses an **internal perspective of the system** to design test cases based on internal structure. It requires programming skills to identify all paths through the software. The tester chooses test case inputs to exercise paths through the code and determines the appropriate outputs.* (emphasis mine)
- http://en.wikipedia.org/wiki/White_box_testing (5.31.08)

As a White Box tester, you will be expected to learn the parts of the code that you will be responsible for testing and the tools you will test with as soon as possible. Being able to read through the code well enough to spot weaknesses will allow you to quickly exploit the most vulnerable parts of the program.

The quicker you can do this, the sooner you can deliver a report that will make you look like a rock star!

No matter what type of testing you perform, your end goal remains the same. After submitting your initial reports and bugs outlining the highest risk areas of the program, you can continue through the code. As you do, you can continue to focus your testing on the next weakest area of the program.

Using this progression, you will give the project team the most time, and therefore the best chance, to fix the highest risk and highest priority bugs in the program. You will add the value that only a true Quality Assurance Professional can.

In your travels you will also encounter the term Grey Box testing. This is exactly as it sounds: a mix of Black Box and White Box. It attempts to adapt the strengths of each type and meld them into a “whole” testing that is greater than the sum of its parts. Grey Box can take the ease-of-use, straightforward approach of Black Box testing and leverage it against the in-depth, code targeted testing of White Box.

In your career as a Quality Assurance Professional, you will most likely have the opportunity to experience and execute Black Box, Grey Box, and White Box testing. They each have their place and use.

The principles outlined in this book are applicable to any type of testing. The method used to test depends upon the priorities and capabilities of the company

© Phillip Bailey - All Rights Reserved

that undertakes the task of creating a program or product. Each company has its own way of reaching the point of having a shippable product or service.

For the purposes of this book, the examples that I use will focus on Black Box testing as this is the most common type of testing that you, an entry-level tester, will encounter. Bear in mind that if you learn the principles detailed in this book, you will excel at any type of testing. Being adaptable is the key.

© Phillip Bailey - All Rights Reserved

Testing: Manual vs. Automated

Testing is a skill. While this may come as a surprise to some people it is a simple fact.

~ Graham Fewster: Software Test Automation

In addition to debates about Agile vs. Waterfall and different than strictly White Box, Grey Box, or Black Box, testing is categorized yet another way: Manual vs. Automated.

Manual testing and Automated testing are, quite simply, just as they sound.

Manual Testing requires a tester to execute each test step by hand. Every button to be pressed, each link to be selected, all assets to be verified are done by a human. Thus the term “manual”.

Manual testing can be very time consuming and quite laborious. It is, however, where the Quality Assurance Professional can, and does, most effectively advocate for the end-user. By looking at every page, by reading every Help link, by analyzing every graphic, and by exercising all use paths available to the consumer, you will contribute your expertise to help create the highest quality product possible. A human is necessary to judge whether or not the user experience is satisfactory – no computer can do that yet.

Once you have mastered Manual testing, you may find that Automated testing is a skill you would like to become proficient in. Until then, let me give you a little peek into what Automated testing is.

Automated testing is testing that is executed using a tool (either off-the-shelf or created in-house) to exercise the code. The tests are set up by a QA tester or QA engineer using the tool so that in the future a battery of tests can be run by pressing a single button or entering a single command.

The tool may be complex or it may be a simple point-and-click interface that records the actions the tester takes. Once the test steps and actions to be performed are defined, they are recorded so that they can be executed any time in the future.

© Phillip Bailey - All Rights Reserved

This takes more than a couple of minutes. Once you have defined and recorded the steps, you must run your automated test script and fix any issues that will produce a falsely negative result.

What I mean by this is that you must ensure that all of the results your tests produce are reliable. Each failure your test script reports should be an actual failure – not a false alarm. This is why you must take the time to tweak your test scripts (yes, I am using "tweak" as a technical term).

After you have set up your Automated test scripts, they must be maintained to keep up with the changes that are made in the product you are testing. If, for instance, your scripts begin reporting that a link is not working, be sure that the link itself is bad.

If the placement of the link on the page has changed, but your test script has not been updated to take this change into account, then you will see a failure. But this failure will be a *false* negative. If you report this false negative, you will appear to not know what you are doing.

So, just as with everything else you will do as a Quality Assurance Professional, be aware of your work and attend to all of the details. You will be happy that you did.

If you would like more information about Manual testing vs. Automated testing, check out Wikipedia. Or go to Amazon.com or a library and do some research. I guarantee you that there are hundreds, if not thousands, of books on every kind of testing you can imagine.

A fool with a tool is still a fool.
~ Grady Booch

© Phillip Bailey - All Rights Reserved

Before we get into the meat of this book, I would like to make note of two items to help give you a clearer understanding of its contents. They are: Terminology & Numbers.

Terminology

The last good thing written in C++ was the Pachelbel Canon.
~ Jerry Olson

First, a brief note about Software Quality Assurance terminology: There is a standard that has been used for many years in software and hardware creation that all companies use some form of. However, each company that I have worked for has had slight, or great, differences in how they define and use each term.

What I detail below are the basics; the standard, if you will. You should learn them and know them and you should be able to speak about them in at least general terms. Then you will need to learn what your company calls each item and how they define them.

I have never worked at two companies that defined these terms identically – and some companies even add their own "extra" terms to address their own circumstances. So learn the basics and then be ready to learn the specifics as your company uses them.

© Phillip Bailey - All Rights Reserved

The terms below are listed in such a way that you can easily understand how they relate to one another.

- **SDLC** – The Software Development Lifecycle. This is the framework within which the project team operates. It exists to provide all team members the same perspective of what state the project is in. Its structure helps define each project milestone, from first build to last.
- **Pre-Alpha / Feature Drop** – These are the earliest builds created and delivered to QA for testing. Often they contain very limited functionality. The testing focus during this phase is on specific features, not on the program working end-to-end.
- **Alpha** – The Alpha phase of testing is often focused on validating that all assets are present in the program and that the program is usable end-to-end. At this point the program is nowhere close to complete, but all assets should be present and testable.
- **Beta** – In Software Quality Assurance, the term Beta can mean multiple things. In its simplest form, the Beta phase is reached when the software is feature complete. This means that all assets and functionality are present and testable. They may not all work as desired, but all functionality has been implemented. During the Beta phase, the software may also be released to a small group of external users – this is known as a Beta Test.
- **Release Candidate** – Once the developers believe that the build they are delivering has the potential for release as the final product, and if there are no open "showstopper" bugs, then a build is deemed to have reached the status of Release Candidate.
- **Final Candidate** – Some companies include a step after Release Candidate that is called Final Candidate. This is a build that will be subjected to one final all-out assault by QA in an attempt to unearth any remaining showstopper bugs. A build that clears this hurdle is ready to be shipped.
- **Gold Master** – Once a build has passed all of the final tests, no showstoppers exist, and no Active bugs are assigned to the project, then a build will have achieved Gold Master status. This is a build that QA will bless as shippable. This is the product that will go to manufacturing, and then be released to the public for their enjoyment and abuse.

At the end of this book I have included a much more extensive Glossary that you can reference at any time. **You can also find a glossary that gets updated regularly on my website.**

© Phillip Bailey - All Rights Reserved

Numbers Are Your Friend

We know about as much about software quality problems as they knew about the Black Plague in the 1600s. We've seen the victims' agonies and helped burn the corpses. We don't know what causes it; we don't really know if there is only one disease. We just suffer — and keep pouring our sewage into our water supply.
~ Tom Van Vleck

Another critical piece of information that you should keep in mind while you are reading this book is that numbers are your friend. Numbers are the ammunition you will use to make your point. Numbers are the language you will use to communicate the impact of the issues you uncover.

Even if you are one of the many people that find numbers intimidating, in this book I will show you how to make them work for you. I will walk you through the simple actions you need to take so that numbers are a benefit, not a burden.

Time and again in this book we will return to the impactful ways that you will use numbers. I will show how present numbers so that the bugs you find are taken seriously. I will show you how to capture numbers and use them to improve your team and your daily work life. And I will give you the tools you will need to use numbers to make more money.

The thing to remember here is: **Numbers are your friend**. Numbers are a tool that can work for you or against you. If you master the principles in this book, you will be able to always have the numbers in your favor and working for you.

Using numbers by **leveraging the impact** they will provide you is a skill that will serve you well for many years to come. Knowing how to present numbers so that they emphasize your point is a key to making all of your communication more effective and makes others hold your work in higher regard.

Because they can either be a hindrance that prevents you from advancing in your career or **the cornerstone of unquestionable work**, in this book I will show you how to use numbers for your greatest benefit. If you use the power that numbers provide you properly, you will quickly separate yourself from the average tester.

So remember: Numbers are your friend. Take heart and read on. I will give you all you need to know to be able to use numbers for your greatest benefit.

© Phillip Bailey - All Rights Reserved

Chapter #3: Testing: Science Vs. Art

Once there was a time when the bringing-forth of the true into the beautiful was called technology. And art was simply called techne.
~ Martin Heidegger

Science

Definitions attempting to explain "Science" abound. Here is the one that, to me, is the most applicable to testing:
"The systematic application of knowledge or skill in effecting a desired result"
-http://dictionary.reference.com/browse/science (5.31.08)

The keys in this definition as it applies to testing are:

- Systematic application
- Effecting a desired result

Effective testing is systematic and knows what outcome is expected. It is not random or haphazard. This is why Science is the most important element of testing.

Art

There are even more numerous and varied definitions of "Art". I think that the most apropos definitions for our purposes here are the following:
"Skill that is attained by study, practice, or observation"
"Skill arising from the exercise of intuitive faculties"
-http://dictionary.reference.com/browse/art (5.31.08)

The keys to these definitions as they apply to testing are:

- Skill
- Study
- Intuitive faculties

Your most amazing test results will be the fruit of you exercising your greatest skill learned from in-depth study and trusting your instincts. This is why Art is the most important element of testing.

© Phillip Bailey - All Rights Reserved

Which is more important: Art or Science?

Conundrum? Conflicted? Confused?

As a Quality Assurance Professional, you must master both. You must understand and be able to practice both the Science of testing and the Art of testing. This sounds daunting, but if you study and practice you will soon become a highly valuable tester.

Software Quality Assurance is as much an art as a science. Being able to execute thorough, detailed tests step-by-step is a critical part of the Quality Assurance process. However, just as important and often overlooked is the art of testing. This art comes into play when a tester using their intuition, common sense, and curiosity of the unknown exercises code in a way that the creators and developers did not anticipate.

This QA art is focused on the experience of the end-user. It asks whether a product or service will meet the consumers' needs. The QA professional is the last line of defense for the consumer and therefore must advocate for the consumer long before the consumer ever sees the product.

When a QA professional knows their target market, understands where and why the codebase is weak, and brings to bear their formidable abilities to make the program cry for mercy – *this* is the true art of Software Quality Assurance testing. It takes time to master, but a tester that can do this is, not surprisingly, far more valuable than a tester that can only follow the steps in a test suite to exercise the code.

Let's look at both the Science and the Art of Software Quality Assurance testing, Science first:

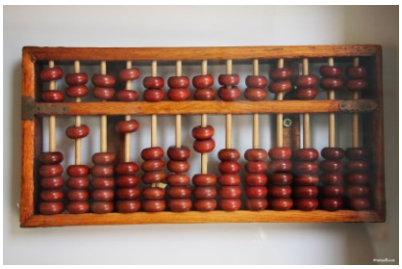
The impossible often has a kind of integrity which the merely improbable lacks.
~ Douglas Adams

© Phillip Bailey - All Rights Reserved

The Science of QA Testing

Quality Assurance testing has its roots in Scientific Methodology. But testing does not end there, that is only where it begins. The science of testing is in the methodology of what to test, in the pre-defined steps to be executed in order to exercise the code, in capturing the results of the testing, and in the measurement of those results. Remember, numbers are your friend.

What you measure matters. This is true in any endeavor where you want to know the status, the performance, and have the ability to improve. If you do not measure, you cannot know with certainty where you are. If you don't' know where you are, you cannot create a plan for improvement. This principle is at the root of the Science of QA Testing.



Using documentation that details the desired end result of the project, a Quality Assurance Professional creates a plan that outlines at a high level what will be tested and what methodologies will be adhered to in order to execute that testing. This is known as a Test Plan. Achieving consensus on this plan between the creator, the developer, and the tester is the first step in being able to accurately measure the project to be undertaken.

Using the Test Plan as an outline, the Quality Assurance Professional then moves into the Test Suite creation phase. Referencing more detailed documentation (technical and marketing specifications) the QA professional creates a detailed suite of test cases that will exercise the program code step-by-step and piece-by-piece. This test suite will also include the way in which the results of the testing will be tracked, and thus, reported.

All of this planning and preparation must be done so that the project can be properly tested and measured. You must know **what** you will test, **how** you will test it, and be able to **capture** and **report** the results of your testing. By doing this you will be able to highlight when the project is on schedule, ahead or behind, and how much more effort will be required to complete the project. This will also give you data to use when planning your next project by using the results from your past projects to forecast the effort involved in future undertakings.

© Phillip Bailey - All Rights Reserved

Now the testing begins! Starting with the highest priority piece, you execute each test case meticulously. You perform each defined step. You track your results. You track the amount of time each batch of tests takes to complete. You capture the total number of bugs found in each section.

This is measuring. You have already determined the plan of attack and now you are executing it. This is how you know that you are on track.

Are there parts of the program you should be able to test right now but cannot because there is a bug preventing further testing? Will this issue delay testing and impact the schedule because the testing you had planned to do will have to be performed later than expected? This is where you will find out.

Plan (define your approach). **Test** (work your plan). **Capture** (the results of your testing). **Measure** (your results against your plan). **Report** (the results and quantify their impact). This is the key to the Science of QA testing. Plan. Test. Capture. Measure. Report. Rinse. Repeat. Repeat. This is the nuts and bolts of QA!

The Art of QA Testing

As true as it is that "what you measure matters", it is also true that the greatest value of a Quality Assurance Professional is the ability to ask "what if..?" and then clearly communicate the impact of the answers you receive.

This is where the Art of Software Quality Assurance enters our story. No matter how thorough a test plan and test suite are, no matter how thorough the automated testing is, no matter how confident a developer is in their code, the greatest Quality Assurance Professionals are still able to come up with insightful, as-yet-unanswered "what if..?" questions. Then, when they unearth the answers to their questions, they can succinctly explain what those answers mean.

What if?

It is in asking the right questions that a true Quality Assurance Professional shows their worth! The innate or developed ability to sense where the program is weak, to feel when they have put the program in a precarious state, to go boldly or stumble into an area of the code that can't keep itself from crashing – this is what separates the truly great QA tester from a warm body that knows how to push buttons and can follow simple instructions. If you want to be a truly great tester, ask "what if..?"

© Phillip Bailey - All Rights Reserved

"What if..?"

- I hold this button down for 5 minutes
- I close the program using key-commands while the program is performing a critical function
- I remove the CD or DVD when it is in use
- I choose to NOT enter required information repeatedly when prompted
- I ***paste*** more than 256 characters into this text-entry field? (This one test, in particular, has crashed more websites and web-based apps in their development process than any other test I have ever seen – and yet many test suites do not include this test. To this day, this test is not accounted for by developers – I last saw this crash an app in 2007…you would think someday, someone might do something about this before QA gets their hands on the program)

And the list goes on. What "What if..?" can you think of?

I should include, here, an anecdote regarding using the above "pasting text" test. To prevent you from looking like an amateur, please learn from the story below.

How Many Databases Can One Bug Crash?

Several years ago I had the honor of working with a great group of Quality Assurance Professionals at a company that existed only to test. All we did was Quality Assurance and we were good! We could test anything at anytime for anyone. This was during the dot-com boom. Everyone had a webpage and money – or so it seemed – and so we spent a great deal of our time testing websites and all of the backend technology that supported them.

One of the projects we were testing was, of course, a website. The website was designed to allow people to enter a question, any question, and receive an answer from an expert in the appropriate field. Supporting this functionality was a huge database that could capture each question and answer for future reference and use. The man assigned to lead this testing was one of my colleagues that we will, for the sake of this story, call Brad.

Brad had been given the enviable task of testing this website and its accompanying technology. This was back in 1999 when what we were testing was the early stages of a new paradigm in how people would use the World Wide Web in the future. Brad got to test the site's user interface, all of the options and compatibility, as well as the database that existed to support the site and become the knowledgebase they were attempting to create.

© Phillip Bailey - All Rights Reserved

One of the most obvious batteries of tests that Brad executed early in the project included stressing the input acceptance capabilities of the question submission forms. The first actions that one can take to test text entry fields uncover whether or not the form has been set up with rules sufficient to protect the back-end systems. Usually these text entry fields are overlooked when developers lock down their code.

The first type of test that Brad ran was text entry via typing. How many characters would he be able to input just by have the cursor in the text entry field and then typing? As expected, there was a hard limit (256 characters, I think). He was only able to type in a pre-determined number of characters before he reached the limit. Happy to see that they had at least set some boundaries on the text entry fields, Brad now moved on to the next stage.

Before executing the next set of tests, however, Brad called the producer and the developer to warn them of what he was about to do and to get their "ok". Since Brad knew that there was a possibility he might crash the database on the back-end, he wanted to be sure that everyone was on the same page and ready to react should the worst happen – to be fair, I suppose that the server catching fire would have been worse, but that would be a different story.

Brad informed the producer and the lead developer of what he was planning. He asked if they were ready should the test produce a severe outcome. They assured him that they were ready. The lead developer also assured Brad that there was "no way" that the planned test would have any adverse affect on the database. They resolved that the testing should go ahead as planned.

Knowing that there was a limit on how many characters he could **type** *into one of the text entry fields, Brad began testing the limits of each field by* **pasting** *characters into it. Pasting characters into a text entry field is a different beast than typing.*

When Brad pasted 100,000 characters into one of the fields and submitted the "question", the website froze. It stopped responding completely. This was not good (well, not good for the website or the database).

Brad immediately got on the phone to warn the developer that something, indeed, had appeared to be adversely affected by the test. Either that or there was a heck of a coincidence going on!

© Phillip Bailey - All Rights Reserved

Brad reached the lead developer before anyone at the website's company was aware that anything had happened. He explained to the developer what he had done (as they had discussed before) and then told the developer what had happened on the user's end. The developer immediately went to investigate the effect of what Brad had done.

It turned out that pasting 100,000 characters into the text field crashed the database and subsequently caused the website to stop responding, as it was waiting for the database to respond, and that wasn't going to happen. This gave Brad and his test team 2 important pieces of information:

1. *The text fields were not yet protected from the end-user (as he expected)*
2. *The website was vulnerable to any adverse behavior from the database it queried. This was not really a surprise, but it definitely hadn't been expected. This would be an area of continued testing as the project moved forward*

Once the website and database were functional again, Brad resumed his testing careful to avoid any text-pasting tests. The project continued.

Now, here I should back up a little in our story and explain that the defect tracking systems (bugbase) we used for most projects were hosted on a server in our office. This gave us the ability to customize each bugbase to most effectively serve each client. This also was a simple solution for clients whose producers, developers, and asset teams were based in separate locales. The entire team could log into their bugbase on our server to track their project with ease.

The only weakness to this setup was that all of our bugbases were hosted on a single server – all thought to be safe from outside malice. Only the project teams had access through the firewall, and it was in the best interest of every team and every client to use the bugbases as bugbases – and ONLY as bugbases.

What happened next was not expected…

Brad's project continued, as did the ten other projects our company was testing at the same time (this meant we had at least 11 (eleven) bugbases active). Then one afternoon, all of our bugbases ceased responding at the same time. All at once we turned to each other, questioning looks on our faces.

Immediately we began checking our connectivity – all fine.

Everything was fine except for the bugbases.

© Phillip Bailey - All Rights Reserved

Time to check the server. Aha!

There was the problem – the server was not responding! This was not a good thing. Eleven bugbases offline and inaccessible to hundreds of people trying to complete their projects on time.

We went about resetting the server and manually restoring each bugbase, doing our best to ensure that no data had been lost in the crash. This was going to take time, but it had to be done.

Just about that time, the phone rang…

By sheer luck, Brad answered the phone. As I was restoring the bugbases, I was peripherally listening to Brad on the phone, as was the rest of the team. Then he got everyone's attention…

"You did WHAT?!?" Brad asked incredulously. Then quickly regaining his composure he asked, "Why? Why did you do that?"

By now we were all curious what Brad was hearing. "I see. Uh huh. Ok, well yes…you did crash the bugbase," Brad was saying. "Now let me explain why I need you to, please, never do that again."

Brad spent the next several minutes explaining, in the most polite terms possible, to whoever was on the other end of the line that we had all of our bugbases on one server. He continued to patiently explain that this meant that the actions the person on the phone had taken resulted in ELEVEN project bugbases going offline, not just one. Having made his point as clearly and kindly as possible, he asked to speak with the caller's boss.

Brad then patiently and politely explained the same thing all over again until he was sure that he had made his point. Then he hung up…

"Well?" we all asked, "Who was that??"

*Turns out that the lead developer, from earlier in our story, believed he had a fix for the crash Brad had caused, but wasn't sure that the problem was unique to his database. In order to test his theory, he decided the best course of action would be to paste 100,000 characters into OUR BUGBASE…**that** is what caused the crash.*

© Phillip Bailey - All Rights Reserved

Had he asked, we would have told him that our bugbase supported only 60,000 characters in each text entry field. This was a known limit. We also knew that it was possible to paste more than that into a field. We could have answered his questions had he consulted us. Instead, we spent the rest of the afternoon fielding calls from producers, developers, artists, and other team members all asking about the apparent instability of their bugbase. We assured all of them that the issue had been addressed.

There are some important points to take away from this story:

- Always make sure that you get clearance before crashing a mission critical piece of a project and clarify that the team is ready to act if necessary
- **Always be nice.** No matter how upset you may be, be nice. It will help you make your point more effectively. It shows how professional you are. If you cannot be bothered to be nice, you will present yourself as someone who lacks professionalism. Be nice. The producer in our story was so impressed with Brad's handling of the situation (in addition to the expert testing they received) that he sought out our services when he moved on to a different company. Be nice.
- When a mission critical system is affected, jump to the task of fixing it as soon as possible. Fixing it tomorrow is too late – these systems need attention ASAP. If you are not the one to fix it, find out what you can do to help.

Communicating Impact:

The Art and the Science of QA merge not only when testing, but also in the way that you communicate. Using the Science, you will begin to build numbers. Through the Art, you will most effectively reach your audience.

As a Quality Assurance Professional, one of the tasks that will make you highly valuable to any project is your ability to communicate impact. You can become proficient in this by merging 2 abilities:

- Your ability to write and speak in such a way that your audience will understand the information you are delivering
- Your ability to quantify the affect of the effect

To truly be an effective Quality Assurance Professional, you must be able to clearly explain the results of what you have measured. You must be able to put in perspective the answers you have found when you asked "What if..?"

© Phillip Bailey - All Rights Reserved

Don't assume that others will come to the same conclusions that you did when they review your results. They are starting from a different perspective. They don't know all that led you to take the steps you took. They may not make the connections you have about the systems that are affected.

It is up to you to communicate this impact in your emails, bugs, reports, and meetings. You must state not only what the outcome was, but you must lead them to the "correct" conclusion at which you have already arrived.

You very probably will have to patiently lead your team members down the path that you followed. You may need to explicitly point out the meaning of the results you are presenting. This is not difficult, but it does take practice for most people to become proficient in.

Take the time to become proficient. You will separate yourself from the crowd. You will be able to make your point more effectively in any situation if you are able to communicate the impact of the information you are delivering.

You have executed the outlined test steps and have asked "what if..?" All of that work has given you results that you need to deliver. The way in which you deliver those results determines whether you are viewed as an amateur tester or as a Quality Assurance Professional.

The average tester will simply submit a report that contains no details other than "this is what I did."

In this book you will learn how to deliver reports that include:

- The plan you are executing
- What portion of the plan you have executed
- The results of your work to date
- The effect those results are having on the project
- The effect those results will have on the project
- When your work will be completed

And you will deliver all of this information in a format and language that will be easy for everyone else to understand.

As you learn the principles presented in this book, you will begin to master both the Art and the Science of being a Software Quality Assurance Professional. There is more to this career than testing. Testing is integral, and you must learn

© Phillip Bailey - All Rights Reserved

to test at an elite level, but that is only the beginning of what can be a rewarding future in the Software Quality Assurance field.

As you read on, think about what you will measure and when you will ask “what if..?” Absorb the information I give you in detail and analyze it holistically. By taking both a detailed and a wide-angle view of the principles in this book, you will quickly gain an understanding of each process and the insight to see how they work together.

Alone, each lesson presented here can help you overcome previously insurmountable obstacles. Together, they offer the opportunity to write your own ticket in the high-tech industry. When you master the principles in this book, you will have joined an elite few that understand the true value of QA and bring to the table the highest level of professionalism.

© Phillip Bailey - All Rights Reserved

Chapter #4: 3 Essentials Skills

In order to become a software tester, you must have 3 skills. Without these, you may never have the chance to get hired. Since this whole book is about helping you succeed as a tester, having the skills to create that possibility is priority one!

The 3 Essential Skills

1. Testing Skills
2. Computer Skills
3. Communication Skills

If you think you have all three of these skills aced, feel free to skim this section and move on. But if you're not sure, if you think you may not have these skills mastered just yet…stay with me and read on.

First let me explain why you have virtually no chance of securing a software tester position without these skills. Why are these specific skills so important?

These 3 Essential Skills form the basis on which you will or will not be considered for a testing position. If you can demonstrate in your resume and in your interview that you have a basic proficiency in these areas, then you will have taken the first step in creating and controlling your career as a Software Quality Assurance Professional!

Let's get you started on your journey…

© Phillip Bailey - All Rights Reserved

Essential Skill #1: Testing

As a Software Quality Assurance tester, your main job each and every day will be to execute tests. This requires more than simply pushing a button and waiting. You must know which tests to execute, how to execute them, and what to do with the resulting information.

If you are able to demonstrate that you have already developed this skill when you apply for a testing position, then you will be ahead of your fellow applicants. If you can show potential employers that you already have an understanding of what will be required of you, then they will know you are a better investment than your competition.

How do you show that you understand software testing? What can you put on your resume? What information should you use in your interview to impress an employer enough to take a chance on you if you don't have previous testing experience?

Find a good place on your resume to highlight that you have read this book. Start there (we will cover resumes and related resources later in the book). Your potential employers in the software development industry will know how to find out if I'm legitimate. Anyone worth working for will know how to evaluate the information in this book and they can look up my credentials (as can you).

Then, of course, make sure you actually do read and understand this whole book. That is a start.

Once you get to your interview (which I am confident you will if you use the tools and resources in The Job Hunt chapter of this book), you can detail your understanding of software testing. You can demonstrate your **working knowledge of the principles and techniques** any beginning software tester should know.

What are those techniques? Glad you asked.

Following the links, you can find more in-depth information about **types of software testing** and to various **software testing techniques** on my website.

© Phillip Bailey - All Rights Reserved

That will help to give you a well rounded introduction and understanding of the basics.

Then you can tell them about where you would begin your testing under two different scenarios:

1. If given a test suite to follow
2. If left on your own to perform exploratory testing

Here are some techniques to get you started. Learn the principles behind each of these tests and you will be on your way understanding fundamentals.

© Phillip Bailey - All Rights Reserved

Scenario #1: Following the Script

When given a test suite to execute, your job at first is just to follow the instructions in each test case and compare the outcome you witness with the expected outcome listed in the test suite. These two results should be the same.

The product design and technical specification documents establish a product's requirements. To make sure that each and every requirement behaves as intended, test cases are written to cover each requirement. The outcome of the test cases is then tracked so that the QA team knows if and when all requirements have been tested.

Each test case you execute should give you **exact steps** to follow. By following the stated steps, you will be verifying the behavior of a feature requirement – or at least one piece of it.

That is why I say that testing is simple – all you have to do is follow directions. This is crucial if you want to learn software testing, *you must be able to follow directions*.

Unfortunately this assumes that the test cases you have are given have been written clearly. This is not always the case.

If the test cases you are trying to execute are not crystal clear, make sure you receive clarification as to what the intended testing is to cover. You can either update the test case yourself (so it is easier to follow in the future) or get it on the to-do list of whoever is responsible for that.

Until you learn software testing well enough to compensate for poor project set up, the test cases must be clear and accurate or your testing will suffer. You may not test the right thing; you may miss requirements that need to be validated – in short, get your tests accurate or risk participating in bad QA.

While executing test cases, **take each one a step at a time**. Perform each step of each test with diligence and attention. Make sure you aware of the outcome of every move you make. Compare the results you witness with the expected results as stated in each test case. Anytime there is not a match, ***you have found a bug!***

© Phillip Bailey - All Rights Reserved

But what if you don't have a test case in front of you? What if you have been tasked with performing exploratory/ad-hoc/real-world/end-user testing? How will you be able to find bugs?

Follow me…

© Phillip Bailey - All Rights Reserved

Scenario #2: Low-Hanging Fruit

Whenever you have a chance to freelance, you should jump at the chance! This is a fantastic opportunity to further learn software testing.

This is your best chance to hone your skills. This is where you really learn software testing.

Where should you begin your exploratory testing? How can you make an impression on your boss and show that you have a knack for finding weaknesses in software? Being able to quickly and effectively identify where a product is weak, is really the art of QA testing.

Common software weak points:

- Data Capture
- Selectors
- ReadMe.txt
- Transitions
- Interrrupts
- Defaults
- Un-Install

Data Capture

Any time a program or web page have data capture fields, you have a chance to exploit them. Data captured goes into databases for future use so if you can affect the data capture field, you may be able to affect the database.

Most data entry fields allow you to enter whatever you want. Their rule checking happens upon submission. Anytime you can get around the rules that the database on the back end needs enforced so that it can do *its* job, you can put the whole system in jeopardy. So here are some things to try:

- How many characters can you type in a field?
    - This is usually limited to 256 or less. So, if they limit the amount you can type in, then go find a copy of War and Peace and see if you can Copy and Paste. Often you can paste more into a field than you can type
- Which characters can you enter by typing?
    - Again this is often limited, so go back to Copy and Paste. Now which characters can you enter into the field?

© Phillip Bailey - All Rights Reserved

- If there are any rules stated next to the field (minimum number of characters, etc.) see if they are being enforced
- If there are multiple data capture/entry fields, use the [Tab] key to navigate from one to the next. See if they go from one to the next in a logical order

Selectors

Much like data capture/entry fields, selectors can be very weak in the early stages of development. This is often due to the development team using non-standard, basic "rules" that they expect to work. It is thought to be such mundane work, that this is often a rich area for bugs.

Radio buttons, Drop-down menus, or any other selection mechanism should be toyed with as though you were a halfway intelligent monkey.

- Left and right mouse click very quickly on a selection
- Double-click a selection
- Quickly one and then another selection
- Click and drag prior to selection
- Select any blank line item option

The other thing to check is what happens if you don't select any of the available options. Are there defaults selected already? Are the defaults functional even if you don't select them?

ReadMe.txt

R.T.F.M. = Read the Manual (Don't ask what the "F" stands for)

Part of your job as a QA tester is to ensure that the information made available to the user is accurate. It is very common to use an old ReadMe.txt file on a new product or updated product version. Often the ReadMe is updated *very late* in the project, but it will have to be (or should be) updated prior to launch.

The ReadMe is often a great place to continue to learn software testing. You will find information about the product that you may not have known!

Get into the ReadMe immediately and see if it is accurate. You can often find several bugs to log just in outdated information, information that wasn't updated prior to the previous release, or even copyrights. In the real world, not many users actually R.T.F.M., but if you have a ReadMe that is shipping with your product, then it had better be accurate.

© Phillip Bailey - All Rights Reserved

Transitions
Programs are often very weak during any and all transitions. Transitions from one screen to another, from one scene to another, from one user state to another, or from one game level or state to another. Make selections that can engage the program during a transition.

This means, make a selection (whether or not you can see it onscreen) while a transition is occurring. Even if you can't see the selection you are making, click on the location of where selector *would be normally*.

Interrrupts
This is a classic step in your ability to learn software testing. Just as stated above with transitions, interrupting the program while it is trying to do something can often give it problems. This is especially true in the earlier stages of development (hopefully this becomes less common later in the project).

Create an interrupt while the program is not "expecting" one:

- While it is loading
- While reading information
- While writing information
    - One example would be to unplug a memory stick (flash drive, USB drive) from a computer while it is reading/writing – bad things happen. Apply this test to devices and memory reads/writes other than external USB, that is only an example so you understand the principle

Defaults
Verify that a program's default selections function properly if you don't select anything. If you really want to learn software testing, play with default/non-default settings. You will learn more about testing *and* about software and traditionally standard weaknesses all at the same time.

Another issue is to install to a non-default location. There are thousands of programs on the market right now that do not function properly if they are installed anywhere other than the default (sad, but true).

Even if the main body of the program works, check the peripherals. Do the installed shortcuts recognize that the program was installed to a non-default location or is the shortcut path hard-coded? Does the program save information to a non-default location?

© Phillip Bailey - All Rights Reserved

Un-Install

UnInstall the program and see if it really does "Un-Install". Upon completion of the Un-Installation, does the program inform the user that everything has been uninstalled (or does the program imply it)? Another major area of knowledge if you want to learn software testing is to master the ins-and-outs of Installation/Un-Installation.

If, upon completion, the program tells the user what was not removed – OR – that not everything was removed and perhaps tells the user what was left and/or where, then that is ok (technically – although it is always feels like amateur hour to me). But if the user is led to believe that everything has been uninstalled, this should be true.

Often, however, a program will tell the user (or strongly imply) that everything has been removed when this is simply not the case. This can get the company a great deal of bad press depending on what is left behind. News travels fast on the internet.

Run a standard search on all hard drives for the name of the program, shortcuts, etc., See what is left. Search for directories that the program installs; save information, preferences, etc.

If you are on a PC you can also run "regedit" and search the registry for program remnants (from the [Start] menu, select "Run" – and then type "regedit" without the quotes, then select [Ok] or press [Enter] – your registry will open. F3 will launch search function). If a program tells the user all has been removed, then all really should be removed.

And don't forget to combine Non-Default installation and full Un-Installation. Make sure a program can remove itself after installing somewhere it is not expected to. Until you can perform Install/Uninstall tests in your sleep, you still have much work to do to learn software testing.

© Phillip Bailey - All Rights Reserved

And You're Off!

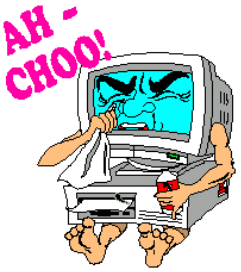
Each of the tests above embodies one (or more) principles of software quality assurance testing. Your task is to learn the tests. Execute each one and get a feel for what they affect and for the outcome. As you do this, you will learn software testing.

By practicing tests that are known to reveal bugs is a very quick way to get a feel for and learn software testing. Execute effective tests and learn what they do. This is your shortcut to understanding the art of QA testing.

Once you have practiced those techniques and feel comfortable in your execution of them, you should be able to speak about them in an interview with ease. You should feel comfortable enough to converse about why you would perform each test, what you would be looking for, how you would execute them, and what you would expect to find.

Warning!!

Before you begin installing software and poking at it on your computer, heed this warning: ***Testing software and attempting to create an unstable situation may cause your computer to perform poorly – or worse! Do this at your own risk.***



It's not that you shouldn't practice, but you need to understand the risks you are taking. Here's how to reduce your risk and take care of your computer.

Prepare before you begin. Take the suggested actions below and you should be able to keep your computer from harm:

******Note: None of this guarantees 100% computer safety******

- **Learn how to restore your computer using a backup image**
  - Every operating system is equipped with a method for restoring a prior image
  - Learn how to use yours
  - Create a backup image
  - Make sure you can restore this backup image

© Phillip Bailey - All Rights Reserved

- Get software that removes remnants that your testing leaves behind
    - After performing an Un-Install, make sure that everything is removed and your computer is back to normal
    - This may keep you from having to use the backup/restore image you made
    - If you do this right, you can integrate this into your Un-Install testing practice…can you figure out how?

Here are several options to help you stabilize, fix, clean, and repair your computer. Look at them and find an option that you feel comfortable with.

You should find a solution that you can work with and have confidence in. Find one that works for you. If you already use a tool that you can control and are confident with, then use that. If not, look at the selection below and choose one to master. It will help you take care of your computer *now* as you practice and later when you have a QA lab to work in.

- **Disk Doctors**
- **Error Killer**
- **iolo System Mechanic**
- **PC Drivers HeadQuarters**
- **PC Repair Doctor**
- **Registry Defender Platinum 2010**
- **RegZooka**

- **Develop basic functional IT skills**
    - This will help you troubleshoot more effectively
    - You will know if you can perform a simple fix or really need to use your backup/restore image
    - Know what free software will aid you in keeping your computer healthy as you abuse it by testing software on it

Before you risk losing any data you have on your computer, make sure you back it up. Always have a safe backup of any critically important data. That way, just in case you have an unforeseen crisis, you still have all you important files.

© Phillip Bailey - All Rights Reserved

Depending on the amount and type of data you have to backup, there are several options.

- Email it to yourself on a free email service where it will remain on the server. Gmail and Yahoo can serve very well.
- Back up to a removable media format:
    - USB stick (flashdrive)
    - MP3 player
    - If you're old school, a Jump or Zip drive
    - If you're ***really*** old school, you could use about a billion floppy discs

You can also back your data up to a hard drive on another computer – where it should be safe should anything happen to the machine you are practicing on. And if you want to start opening up options for yourself in the emerging global economy, you can begin using online data storage.

With an **online data storage system**, you can access your important data from *anywhere* in the world as long as you have an internet connection. That means you can travel anywhere in the world, work anywhere in the world, or live anywhere in the world and still have access to your critical data without having to drag around computer.

This can open up worlds of possibilities if you think about it. If you plan to join the emerging technology workforce, this is the kind of tool you should know how to use.

Here are two options to get you started:

- **Elephant Drive**
    - "Unlimited online storage for your photos, music, video, documents, and all other valuable files you have! Pre-made categories and our simple wizard make sure that all your important data gets automatically backed up"
- **Live Drive**
    - Add Livedrive Briefcase to access your files anywhere. Once you've backed up your files with Livedrive you can do so much more. Add Livedrive Briefcase to get all of this:
        - Add an L: drive to your PC - use your online files like normal files
        - See the same files on all of your computers
        - Access your files anywhere - online and your mobile
        - View and edit files online - watch movies or play music
        - Share your files with friends and family in seconds

© Phillip Bailey - All Rights Reserved

Once you feel like you have adequately prepared your computer, have a plan in case you need to clean the registry or the hard drive, and have a plan to restore in case of emergency…it's time to begin practicing.

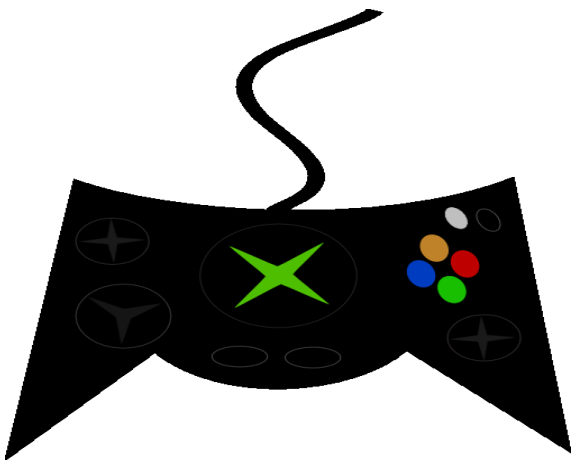
If you don't have a ready-made beta testing opportunity in front of you, you can still practice finding bugs. You can pick out almost any game off the shelf (or online) and find bugs.

Yes, these games have already shipped and thus should be fairly complete, but the sad truth of the software industry is that software gets shipped with bugs. This is especially true of games. There is such a profit to be made from a game that works "good enough" that you should be able to find a decent bug (or several) in just about any title.

This does take practice, but if you are able to identify bugs in these completed games, you will be very well prepared to find tons of bugs in software that is still in development.

You can choose just about any title you like. I recommend starting with games and genres you are familiar with and then as you get an idea of what testing feels like, you can branch out into others that are not your favorite. As a game tester, you may be required to test games that are not your favorite (but more on that later).

Although you can go get the latest and greatest game off the shelf for $50 or so, I recommend less expensive alternatives. There are several options for you to choose from. There are game rentals, inexpensive online games, and even free trials. Who knows, you may end up finding a gold mine online of gaming selections and options.



© Phillip Bailey - All Rights Reserved

Here is a brief list to get you started. Take a look and find an option that you would enjoy practicing with. Once you get familiar with testing, then branch out into to other options:

- **Aces High WWII Interactive Game**
- **Bigpoint**
- **GameFly - Online Video Game Rentals**
- **GameHouse**
- **Gamemine**
- **GameStop**
- **Lord of the Rings**
- **Metaboli**
- **Microsoft Store**
- **Nick Arcade**
- **Nick Jr. Arcade**
- **PopCap Games**
- **Second Life**
- **Shockwave**
- **Yahoo Games**

Find a game you want to practice on and go to it! Practice these simple testing techniques and you will be able to say that you have testing experience before you even get hired!

In fact, I'll even give you a place to post the bugs you find. That way you will have a record on a reputable website of bugs you have found.

When you find a bug, write it up – bug writing is covered completely in this book, so everything you need is right in front of you – and **submit it.** If your bugs pass my quality standards, they will be posted on my website for all to see!

With bugs that ***you*** have found in **existing** games posted on my website, you can direct any prospective employer to a sample of your testing work. Then you can do more than *claim* you practice testing, you can show it!

Let me make this even better…

If you submit a bug that does not measure up, I will critique it and respond directly to you (make sure to include a valid email, ok?). We can work together to get your bugs polished so that they show the kind of high quality work you are capable of. Then when they are ready, you **submit them and they will be posted on your own page on my site!**

This will help get you practice *and* training. And then you can **showcase your bugs for all prospective employers to see.**

The software industry needs more high quality QA Professionals. Are you ready to start your training? How much is a high tech career worth to you?

© Phillip Bailey - All Rights Reserved

Essential Skill #2: Computers

To work as a Software Quality Assurance Professional, you will need to have some computer skills. You don't necessarily need to be able to build and configure a server, but you do need to understand your left-click from your right-click.

Unfortunately, computer skills can take a long time to learn if you have to figure it all out yourself with no help. That's why I am offering some help.

Now, I don't have the time or inclination (yet) to write a whole How-To book on basic computer use. But help is out there. There are books, night classes, maybe even a friend who can get you up to speed on what you need to know.

If those options appeal to you, great! Go for it. Whatever you do, get started with your computer training. Keep reading about QA here and develop your computer skills elsewhere.

But if you want a quicker way to get there…if you want ready-made, useful, applicable, real-world computer training...

What kind of computer skills should you develop?
It really depends on what jobs you will be applying for. The demands of different QA jobs will dictate what software you should learn.

But no matter which programs you end up studying, you should know how to troubleshoot a computer. ***You need to develop basic IT skills.***

Why? Because you will be creating unstable situations daily in your testing. That's part of what you do as a software tester. But once you have crashed your test computer or made it unstable, you must know how to do the reverse.

Breaking your computer (or the program) is only the first part. Once you have created an unstable computer, you need to be able to return it to a testable state. That takes a little know-how and skill.

Most testing computers don't come with an "Undo" button, or an "Easy" button. Once you have mangled them, you have to be able to fix them.

© Phillip Bailey - All Rights Reserved

Sometimes this can be accomplished by restoring a backup image. But sometimes, restoring a backup image take far longer than a simple fix.

To learn some fantastic IT skills, look at this very affordable program. Rich Pryor is giving away so much for free just to start with that you really can't go wrong. His newsletter provides tips and tricks, and his program is filled with useful real-world IT computer knowledge that you can learn and apply today!

Take a look at what Rich is offering. If you need a hand developing your computer skills, this is a great choice! To work in QA you *need* to understand how to use computers – the logic, the OS, the inside scoop. Take a look and decide for yourself.

Learning Software Applications

Some people can pick up any kind of software and just seem to understand how it works. They have the experience to recognize the menu options, can troubleshoot its behavior, and have a knack for getting it to do what they want.

This kind of person is also very rare! If you are not that kind of person, then let me offer you an alternative to spending years in a trade school learning one specific program. That can be very expensive, take way too much time, and when you're done you might as well be a specialist in that program.

Where can you go for affordable software training? Check out this online training!

© Phillip Bailey - All Rights Reserved

TrainingCenter.com offers a long list of programs that you can train in:

- Microsoft Office Suite
- Java
- AJAX
- SQL
- CompTIA A+
- Cisco
- Visual Basic
- Visual Studio
- Unix
- XML
- Windows Systems
- TCP/IP
- And the list goes on…

Their courses are simple and immediately applicable. They even offer printable certificates and transcripts (in case that's the sort of thing you like to add to your resume information).

They let you sample free courses so you can see the value for yourself. Go check them out. If you need Flash or Java or SQL knowledge for jobs you plan on applying for, then this may be your quickest solution.

Another option is Total Training. With a complimentary guest pass, you'll enjoy access to the first 20 minutes of every tutorial in their library. What are you waiting for?

Of course, you could go buy a book on whatever technology you want to learn and have at it. Maybe you can find a book that comes with a handy-dandy help line too…good luck if that's what you choose.

Bottom Line

To work as a tester in a QA lab, you need to have computer skills!

You must be able to understand the programs that you are tasked to test. You have to be able to understand the behavior you see in the software you are testing.

© Phillip Bailey - All Rights Reserved

You should have a working knowledge of hardware and software that make up the computer(s) you use. You should be able to troubleshoot and fix simple computer issues. If you can't, then you will be a burden to your team because you will only be able to create the problems…not diagnose or fix them.

QA Professionals that last in the industry all develop fluent computer skills. You don't have to know everything when you start, but you must be able to demonstrate basic working knowledge. **They want to hire you to test, not to train you in how to use a computer.**

© Phillip Bailey - All Rights Reserved

Essential Skill #3: Communication

To get a job as a QA Professional, entry-level, mid-career, or controlling quality for an entire company – you must be able to communicate. This skill is so important that I address it here and *also* as **Elite Skill #1**.

If you can't communicate in the language of the company, you won't be able to impress in your interview. You won't even be able to create a useful resume yourself. You must be able to write and speak in the language of the company.

Now, at this point, I'm not talking about being able to persuade, convince, cajole, or otherwise sell any particular point – I am talking about being able to communicate clearly.

Before you can interview, you usually have to submit a resume – or at least an application. How are your writing skills? Do you write in a clear professional manner?

Professional Writing:
Do you know the difference between "**then**" and "**than**" or between "**effect**" and "**affect**"? Many people *say* they do (and maybe they actually do) but often when it comes to writing, these are two very common errors.

Are you confident in your writing? If you submit your resume, cover letter, thank you letter, or even a report are you confident that it will sound professional? If not, you should do a little work. You won't get paid like a professional unless you are a professional.

Develop your basic written communication skills so that you will be viewed as a professional. **No one wants to hire a liability. Show that you have command of your writing!**

This can take a little time, but the return for you will last a lifetime. More doors will be open to you. You will see and be able to take advantage of opportunities that otherwise would not be possible for you.

© Phillip Bailey - All Rights Reserved

Speak Professionally:
The other form of professional communication in which you must demonstrate basic competency is verbal. **That is – being able to speak professionally and be understood.**

Again, I am not claiming that you need to be able to sell snow to a penguin, just that you must be understood. There are many would-be testers that I could not hire over my many years in software testing simply because they were not able to communicate clearly.

They seemed to be intelligent, and had credentials that looked as though they had computer literacy…but I could not understand them. If I was unable to understand them, how would they write a bug that would be useful to me or a client?

If I couldn't understand what they were saying, how were we going to work together? I freely admit that I have only the feeblest command of any language other than English, so I wasn't helping them (I was admittedly not prepared to take advantage of any potential opportunity – a point I am rectifying). But I work with software developers and companies that are English speaking.

Therefore my language skills need to demonstrate competency (or fluency) in English. I don't believe English will remain a top language of commerce over the next 100-200 years, but I wanted a job without waiting that long…how about you?

Do want to work with companies in the U.S.? In Silicon Valley? In the San Francisco Bay Area? How about in Ireland or the U.K? There are ***so many*** opportunities for QA in the English speaking world! Are you prepared to take advantage of them?

Even in a worldwide economic downturn, the technology industries (software, hardware, green tech) continue to grow. There will always be opportunities in the tech industries…when will you be prepared to seize them?

Maybe English is not your native language. Lacking fluency can greatly hinder you job prospects when searching for game or software testing jobs in the U.S. Since it is so critical in QA that people understand what you say and what you write, your language skills must be developed well enough that you are able to employ impactful methods of communication.

© Phillip Bailey - All Rights Reserved

Since I have met several testers over the years that had great difficulty with English, I suspect that there may be others (this may be you). So, I have gone out and found some resources to get you on track if this is an issue for you. This is nothing to be embarrassed about – not all of the testers I am referring to were from outside the U.S.

Sadly, many educational systems do not adequately prepare their students for life in the "real world". This doesn't mean that you can't have a career as a software tester; it just means that you must continue to learn and grow.

If you have language shortfalls, trouble in **conversational speaking** or with **proper grammar**, take advantage of these resources. They should help you overcome the initial language challenge you face. Once your language skills are properly prepped, then your effort as an impactful communicator will produce much greater results.

© Phillip Bailey - All Rights Reserved

Your Best Investment

A simple investment in yourself to develop these **3 Essential skills** is the best way to become a viable, hirable software tester. You will stand out above other candidates and will appear to be a sound hiring investment.

Throughout this book, I will tell you time and again that you should do a little work on yourself. This is because none of us is perfect, there is always room for improvement, and most of all because you are the best investment you can make.

Investing in yourself is the only investment you can truly control. **YOU** control the amount of the investment, the type of investment, and your return on the investment. When there is a profitable return, **YOU** are the recipient!

You are your best investment and YOU are worth it! But you have to give yourself the opportunity. The Roman philosopher Seneca said (or, more probably, wrote):

"Luck is what happens when preparation meets opportunity."

It was true back then and it is true now. If you want to be "lucky" you must be prepared to see an opportunity and then be prepared to take advantage of it. That is luck. Once you are prepared, you will be amazed at how "lucky" you become.

If you are surrounded by people bemoaning their fate; constantly complaining that things are not fair – do your best to even the odds for yourself. This is not about "fair" – this is about ***YOU*** *taking control of your career and your life.*

Remember: Luck = Preparation + Opportunity

With the information and resources in the rest of this book, you can become a star. You can become an Elite Software Tester…but first you have to get hired.

Having the basic skills will create the opportunity for you to apply and be considered for a position. Once you have a position, then you can really show

© Phillip Bailey - All Rights Reserved

your stuff! You can demonstrate the highly developed sense of quality that will separate you from your colleagues.

The skills detailed in this book and the lessons directing you to successful action…they won't mean much if you can't create the opportunity to be considered. Later on in the book we will look at your resume, your cover letter, and your overall job search (I have some ***absolutely fantastic answers for you***…when we get there). I will show you how to create the opportunity to be considered…

But if that opportunity is going to do you any good, you must be able to demonstrate these **3 Essential Skills**. You will need to show on your resume and in your interview that you are the right choice.

Develop your testing skills…and show that you understand how to find bugs.

Develop your computer skills…and show that you know how to troubleshoot and fix a computer. Then show that you are conversant in the technology (or technologies) that your potential employer uses.

Develop your language skills to be taken seriously as a professional.
Develop your skills as an effective communicator to take control of your career.

Develop the 3 Essential Skills and you will have taken the first step to creating and controlling your career as a Software Quality Assurance Professional. This book will tell you all you need to know to succeed as a tester. It starts right here!

The 3 Essential Skills

1. Testing Skills
2. Computer Skills
3. Communication Skills

The choice is yours…the investment is yours…the future can be yours…

How much is your career worth to you?

Essential Skill Resources

Learn IT Skills from a Pro:
Learn to configure test environments. Know how to troubleshoot computer issues that YOU will create as you practice and work as a software tester. Become an asset in the QA lab!

Software Training:
Get yourself affordable online training in the software of your choice.

- **Training Center:** Join over 1 million people from over 100 countries who learn online, with these complete interactive training courses.
- **Total Training:** With a complimentary guest pass, you'll enjoy access to the first 20 minutes of every tutorial in their library. Getting started is easy and takes just a moment.

Clean Your Computer:
Keep your test computer(s) stable and ready for anything!

- **Acronis**
- **Disk Doctors**
- **Elephant Drive**
- **Error Killer**
- **iolo System Mechanic**
- **Live Drive**
- **PC Drivers HeadQuarters**
- **PC Repair Doctor**
- **Registry Defender Platinum 2010**
- **RegZooka**

© Phillip Bailey - All Rights Reserved

Gaming Options:

- **Aces High WWII Interactive Game**
- **Bigpoint**
- **GameFly - Online Video Game Rentals**
- **GameHouse**
- **Gamemine**
- **GameStop**
- **Lord of the Rings**
- **Metaboli**
- **Microsoft Store**
- **Nick Arcade**
- **Nick Jr. Arcade**
- **PopCap Games**
- **Second Life**
- **Shockwave**
- **Yahoo Games**

Writing Help:

- **WhiteSmoke**
- **Confident Grammar**

Speak with Ease:

If your conversational English is stilted, you will make others work harder to communicate with you. If you are not comfortable communicating verbally, you will handicap your career.

© Phillip Bailey - All Rights Reserved

Chapter #5: The 5 Keys When Getting Started

Quality is free, but only to those who are willing to pay heavily for it.
~ Lister, DeMarco: Peopleware

A crash is when your competitor's program dies. When your program dies, it is an idiosyncrasy. Frequently, crashes are followed with a message like ID 02. ID *is an abbreviation for idiosyncrasy and the number that follows indicates how many more months of testing the product should have had.*
~ Guy Kawasaki

What do you need to know first? As a Software Quality Assurance tester, there are **5 Keys you need to know**.

The 5 Keys When Getting Started

1. **The Testing Process**
2. **How To Execute Tests and Find Bugs**
3. **What To Do When You Find A Bug**
4. **How To Write A Bug**
5. **How To Deliver Test Results**

That's it in a nutshell. It's simple. Just those five things. Very simple, right? Note I didn't say "easy", I said "simple". Testing is (or should be) simple. The more complicated you make things, the more difficult they will become. Your testing will become enough of a task on its own, so don't make it any more complicated than it needs to be.

This book will show you how to keep your testing simple and effective. Always remember **KISS**. You have probably heard this before, but just in case you haven't, KISS stands for: **Keep It Simple, Stupid**. There is no place where this principle is truer than when testing…KISS. Disregard this principle at your own peril.

Learn it…Live it…Love it!

© Phillip Bailey - All Rights Reserved

Key #1: The Testing Process

In its simplest form, here is what a project looks like:
A company comes up with an idea that they are sure their target market wants. The product is put to paper (or electronic doc form) to detail what the desired behavior and end-user experience will be. The product is then developed (coded, manufactured, etc.).

- Once in a stable enough state, the product is tested
- Testing uncovers issues (bugs) that need to be fixed
- These issues are reported
- Developers fix the issues and the product is tested again
- This developing, fixing, and testing continues until the product has reached a pre-defined state of completion, or is close enough and the pre-determined ship date has arrived
- The product is completed
- The team hold a meeting to wrap up the project (this is called a Post Mortem meeting)

To be the most effective tester possible, you will need to understand the types of testing that will be needed. There are many types of tests that will need to be executed; including Functional, Automated, Boundary, Stress, Compatibility, Regression, End-User, and Ad-Hoc to name only the most common.

In the beginning you will not need to know all of the detailed permutations and definitions of each test, but you should have an understanding of the basic idea behind each test type and why it is important to execute them. You will find more technical definitions of test types in the Glossary, but below are some basic explanations that will get you started.

Keep in mind that just about every company has slightly different terminology and definitions that you will need to adapt to. But if you have an understanding of what the tests are, why they are run, and what they test for, all you have to do is change the name that you associate with each test and you will be off and running.

© Phillip Bailey - All Rights Reserved

Basic Test Explanations:

- **Ad Hoc** – This is testing that is improvised. The tests run are at the discretion of the tester, therefore the experience and skill of the tester is very apparent when this type of testing is executed. Ad Hoc testing is performed without following predefined test steps
- **Boundary** – These tests attempt to "break" the program from within by changing or breaking the program's own rules. Pushing the program from within to check for correct error handling. Are the error messages that it gives intelligible to the end user? This includes tests like: Selecting buttons when you shouldn't, interrupting dialog as it's playing, using both mouse buttons at once, what input will text fields accept (foreign characters, "too" many characters), etc.
- **Compatibility** – This testing answers question regarding whether or not the program will work on all of the operating systems that it claims to? What sorts of errors are given when you try to run on an unsupported OS? Are you given a helpful, clear error message?
- **Functional** – Exercise every single function in a program step by step, piece by piece.
- **Real-World** – This is testing that is focused on the user paths that are most probable.
- **Regression** – This is testing that focuses on validating pre-existing functionality. It is also the term used when testing bugs that claim to have been fixed
- **Stress** – These tests attempt to "break" the program from the outside. Changing external factors to see how the program responds. Checking the error handling capabilities of the program. Are the error messages that it gives intelligible to the end user?

© Phillip Bailey - All Rights Reserved

Key #2: How To Execute Tests and Find Bugs

Follow the instructions. I know that sounds simplistic (and it is) but that really is the first key. Follow the instructions detailed in each test, one step at a time.

Okay, maybe there is a little more to it than that. **First**, find out **what test**(s) are assigned to you. **Then** clarify the **testing priorities**; what test(s) should you run first, which second, etc.

Know **how much time** the tests you've been assigned are expected to take, if possible (this is the first step in Managing Expectations). This is a piece not to be overlooked. If you want to set yourself up for success, you had better **first know what is expected of you**.

So you get your tests, now what? Now you establish what environment you should be testing in.

- What computer should you be testing on?
- What state should it be in?
- Should you have a fresh factory installation?
- Are there specific programs that should be installed and/or running while you execute your tests?
- Are there any specific testing tools you should be using?

You have your tests and their priorities and know the environment you should test in…now what?

Well, where are you going to enter the results of your testing? Is there a test matrix that you should track your results in? Is there a database for entering your test results? A spreadsheet? You have to capture your results somewhere and it's best to know this before you jump in and have several bugs to keep track of.

Knowing where you will enter your test results, you are now almost ready to begin your testing. The last thing to do (for best results) is to note the time you start your testing. This is the first step that you can take to begin measuring. When you can report with certainty how long a test takes to execute, you are able to effectively manage expectations because you have quantified the effort involved with solid numbers. So note the time you start and then, note the time you stop.

© Phillip Bailey - All Rights Reserved

This serves three purposes:

1. Gives you the information to show what you have worked on each day
2. Shows how long the tests actually take to complete, which will help you estimate how long subsequent execution of the same tests will take in the future
3. As you improve your testing, this will show the effect of the efficiencies and improvements you have made. This will become apparent as you compare how long the tests took the first time you ran them against how long they took the 10th or 20th time you ran them. Documenting this improvement will be a key piece of ammunition for you to use when you show off what an amazing tester you have become

Now, ***follow the instructions detailed in each test case***. This is a simple, yet key, action that begins to separate the average testers from a Quality Assurance Professional. Any test can be run successfully by following clear instructions. If the instructions in the test case are not clear, make note of this and update them so that they are simple, straightforward, and comprehensible to all Quality Assurance personnel.

It is your **attention to detail that makes the difference**. So be very clear what steps you are to perform, and then execute them meticulously. As you progress through your test cases you will undoubtedly encounter some behavior that is not what is supposed to happen. If you have followed all instructions correctly up to this point, chances are you have found a bug. Now what do you do? Write it up! Everyone will celebrate! You will be seen as a genius and the accolades will be lauded upon you!! Your boss must now feel the need to give you a raise!

But wait…not so fast.

© Phillip Bailey - All Rights Reserved

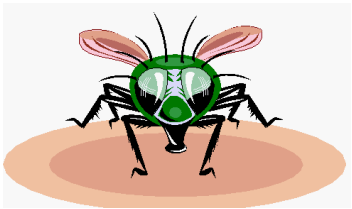
Key #3: What To Do When You Find A Bug

This is where you really make your money as a tester. Just finding a bug isn't enough. Anyone who has ever used a piece of software has encountered a bug (whether they know it or not). What you do once you find a bug is what separates you from the uninformed citizen that thinks anyone can excel as a tester.

Now you add the value that only a true Quality Assurance Professional can. Now you define the bug.

Once you have found a bug you must determine:

- What happened
- Why it happened
- How often it will happen
- Under what circumstances it will always happen
- If there are circumstances in which it will never happen
- What is the real impact of the bug occurring



You can establish all of these points by following just three steps. In just three simple steps you can gather all of the information you will need to write great bugs!

How to Define a Bug in 3 Simple Steps:

1. Eliminate the Variables
2. Establish the Reproducibility
3. Define the Impact

Having found a bug, you must now eliminate the variables and boil the bug down into its most fundamental parts. You must determine what the exact steps are to make the bug happen again and eliminate all else. This way the bug that you write up will have value. If you don't do this, then you have alerted people that something is wrong, but otherwise have only created more work for someone else, either your team or others, and that is never a good thing.

© Phillip Bailey - All Rights Reserved

Step 1: Eliminate the Variables:

So first, back to the bug that you found: **What did you do to make it happen?** Since you have been documenting all of the tests you have been performing, you should have a clear idea of how to reproduce the bug. You should have steps you can follow to recreate the bug. Do that now. Run those same steps again. Did the bug happen?

If the bug did happen, review your steps to see if there are any that can be omitted and still get the bug to occur. Selectively leave out steps that likely don't affect the outcome and attempt to reproduce it again.

Were you able to skip any steps and still reproduce the bug? If so, repeat this process of removing steps until you have broken down the sequence needed to reproduce the bug into the fewest steps possible and make sure you document them for use when entering the bug.

If the bug did not occur again when executing all steps, you will have to delve deeper into what you did to make the bug happen in the first place. Try using the exact same steps and path that got the bug to occur originally. Did it happen yet? It should have. If it didn't, chances are you are forgetting something because although it is true that "sometimes computers just do that", there is usually a reproducible cause.

Try it from the top. Be curious; **are there other things that you did that may have caused the bug? Are there other steps you could *add* that would make the bug occur?** This is where the great tester is separated from the average tester. Test and experiment until you have firmly established the precise actions and sequence with which you can reproduce the bug.

Step 2: Establish the Reproducibility:

Once you are able to reproduce the bug, you need to discover how often the bug occurs. You must be confident that you have determined without question how often the bug will occur when you follow the steps you have outlined.

The way that you do this is to attempt to reproduce the bug several times. Follow the steps that you know will make the bug occur and execute them multiple times. Do this until you are certain you clearly understand how often the bug will occur.

© Phillip Bailey - All Rights Reserved

If the bug occurs every time that you execute the steps, great! This is the simplest and most straightforward scenario. This is a bug that will always occur.

But what do you do if the bug doesn't occur every time you execute the steps? What you must do is repeat the steps over and over until you can establish with certainty how often the bug does occur.

Execute the steps 20 times and make a note or a check mark each time the bug occurs. After you have attempted 20 reproductions, look at how many check marks you have made. Is there one check mark? Are there 2? Are there 10?

You should now have the information to state with conviction how often the bug occurs. Your testing shows that the bug occurs:

- 1 out of 20 times
- 2 out of 20 times
- 10 out of 20 times
- 18 out of 20 times

You can now report with **accuracy** what the **reproducibility** of the bug is. You can state how often the bug occurs when the steps are followed. Being able to report with this level of accuracy will help the bug get prioritized properly and will show that you bring integrity to your work.

If the bug has such a low reproduction rate, like 1 out of 20, that you don't feel confident reporting its reproducibility, what should you do? Test more! Attempt to reproduce the bug 20 more times. Then you will have the results of 40 of the exact same test steps. Test until you feel confident enough to report your bug and stand by its accuracy.

It is very important in this phase of your work to **test until you are confident**. You are the one that unearthed the issue and are the one that is reporting it. You must know that what you are reporting is accurate and must feel confident enough to defend it.

What do you do if you can't get the bug to occur a second time? What do you do if a bug has occurred once, but you are unable to discover how to make it happen again? Don't fret. We will cover this scenario (the "Seen Once" bug) in the chapter titled: The 5 Keys to a Bulletproof Bug.

© Phillip Bailey - All Rights Reserved

Step 3: Define the Impact:
Once you have **Eliminated the Variables** so that you know exactly what steps to follow to reproduce the bug, and after you have clearly **Established the Reproducibility** by executing multiple replications, you will need to ascertain what the ultimate effect of the bug is. Your goal is to be able to understand what the true outcome is when the bug occurs.

This goes beyond bug itself. The bug is the result that was incorrect. The bug is the reason you are logging the issue. The bug is the element that will need to be fixed. For all of these reasons, the bug receives the focus – and rightfully so. But in order to excel as a tester; you must learn to **Define the Impact** of the bug.

In Defining the Impact, you will **communicate what is affected** due to the occurrence of the bug. What I mean by this is that you must do more than state the first thing that happened (the Result of the bug). Once you state what happened (the Result), you will explain what has been adversely affected by that result.

It is not enough to report that the program crashed. Once you state that the program has crashed, you must state what that means. There are many outcomes that can occur.

Once you tell the reader that the program has crashed, you then **quantify** that crash for them. Your explanation could be either of the following examples:

- That the user just needs to restart the program and then can continue to use the program

– OR –

- That the computer became unresponsive and the user has to reboot the computer before they can continue

Can you see the difference in those two scenarios? In both cases, the program crashed. However, in the first case the computer was unaffected. In the second scenario, the computer had to be rebooted before anyone could do anything with it.

Is it clearer to you now how being able to **Define the Impact** and **clearly communicate** it to the project team would affect the priority they place on your bug? Can you see how this gives the bug deeper meaning? A crash that affects the entire computer is far more severe to the end-user than a crash that only

© Phillip Bailey - All Rights Reserved

affects a single program. This is the type of differentiation you need to be able to provide for others.

Once you know how to make the bug occur and know how often it will occur, you will determine what the effect of the bug is to the end-user. When you can communicate this impact clearly, you will add a great deal of informed weight to your bugs.

Being able to describe the bug is very important. To take your bugs to the next level, explain what the real-world impact will be to the consumer. This will show that you understand the true nature of the bugs you are reporting.

© Phillip Bailey - All Rights Reserved

Key #4: How To Write A Bug

A more detailed, step-by-step explanation of each of the critical parts of a bug and can be found in the sections The 10 Critical Elements of a Bug and The 5 Keys to a Bulletproof Bug, but let's start with a basic introduction to bug writing. Below is your beginning guide to bug entry.

So you've found a bug and have determined exactly what steps to take to reproduce it. Now you get to enter it. A very important part to keep in mind when you write bugs is this: *You are about to communicate to someone that their creation is not perfect.* **The way that you communicate this is *very* important** as you strive to build a relationship of trust with other members of your team.

When you tell someone that there is something wrong with the work that they have done, they may become defensive. The first reaction that most people have (even developers) is to claim that they have not done anything wrong at all. There must be something wrong with the way that you have used their creation! Their creation must be error-free. That's what they get paid for and they are extremely good at what they do, therefore the problem must be you.

By writing a bug in the bugbase, you are telling not only the developer that something is amiss with what they have created, **YOU ARE TELLING EVERYONE!** You have created a permanent record in a database that the entire project team or company uses that documents one person's (potentially) perceived shortcomings. When this happens, the developer can become very defensive toward you and may not trust your work in the future.

We will delve more deeply into the most effective ways that you can communicate with your coworkers in 10 Skills of Elite Testers: Skill #1: Communication. But let's take a look at the first steps so that you will begin to understand how to communicate with your development staff.

How can you avoid a defensive response? I'm glad you asked. In order to avoid this reaction, you need to communicate issues with objectivity. Your bug must be a statement of fact, assigning no blame, and clearly stating the behavior you expected to encounter – which is clearly documented in the specs used to create the test cases you are executing.

If after entering a bug in this way you still get a defensive or otherwise unkind response, respond with objectivity, clarity, and politeness. No matter how snarky

© Phillip Bailey - All Rights Reserved

a response you get to a bug you have written, be nice. If someone responds to your bug by writing that it is stupid, be nice. Never get caught up in any personal debates or attacks in your work. Respond as a professional even if the person you are dealing with won't. Be a professional and be nice.

In the section The 5 Keys to a Bulletproof Bug, I include the necessary steps you can take to report a bug with clear objectivity, but you must still remember to always communicate in a way that builds trust and avoids creating an adversarial relationship with other departments.

Let me make this as clear as possible:

When communicating:

- **Always build trust**
- **Never create animosity**

If you can achieve this in your communication, you will be on your way to being able to convince others that you know what you are doing and are doing the right things.

Below is a brief explanation of the steps to take when you write up and submit a bug. You will find a more in-depth explanation of each element of a bug in the section The 10 Critical Elements of a Bug as well as detailed information in The 5 Keys to a Bulletproof Bug that will explain the information you must include and why (look at the example bugs included in the back of this book for reference).

Now you are ready to write up a bug. What should you do? The first thing to do is to open the Bug Template that you created (copied from the example included in this book). It is best to open the Bug Template using a word processing program that has a feature that will check your spelling.

Open your bug template and begin writing your bug.
Start with the Brief Description. Your Brief Description should be a concise, one sentence description of the issue.

Then you move onto the body text of the bug and write your Expanded Description. Here you will use as much verbiage as necessary to clearly state the issue you have found.

Next, write the Setup. Here you will explain any circumstances that must be present before anyone attempts to reproduce the bug.

© Phillip Bailey - All Rights Reserved

Now you will write your Steps to Reproduce. Clear, numbered actions that the person reading your bug will take.

After the Steps you will write your Result. This is the issue at hand. This is what happened.

The last part of the body of the bug is your Expected Result. Here you clearly state what exactly should have happened and, if appropriate, why.

At this point you should have a complete bug. So what do you do? Run a check on your spelling. This cannot be overlooked! Any detail that QA does not pay proper attention to sheds doubt on the rest of QA's work, so **ALWAYS SPELL CHECK**. Few things look worse than a bug that QA has entered that has spelling mistakes throughout it. This is a preventable mistake – don't make it.

Fix any errors that are identified by the spell checker. And now…

Once you have written your bug, go back and review it. Read over it to see if there is anything missing or if you included information that should be removed. Think "less is more". As you read over it, make sure that you have a clear, succinct Brief Description. Make sure that your Expended Description makes sense.

When your spell checking is complete, copy the bug from your word processing document (the bug template) into the bugbase. Once you have done that, select the correct Reproducibility, Status, Assignee, etc. and, *voila*, you are done entering this bug.

During your first week or two on the job, it's a great idea to have your lead (or another very experienced tester) review your bug before you submit it. Once they give you the green light, submit your bug! Then onto your next tests.

© Phillip Bailey - All Rights Reserved

Key #5: How To Deliver Test Results

No matter what your particular position in QA, whether you have been doing the job for one week or for ten years, to be viewed as a true Quality Assurance Professional you must be able to cohesively report the results and the impact of your testing. This may be as simple as a weekly report to your QA lead that details what you have accomplished the preceding week or may be a year-end wrap up of all the projects undertaken and completed by your department; broken down by cost, effort, efficiencies gained, cost savings, proficiencies added, and more. Either way, you must be able to communicate this clearly.

All of this data comes from the daily, weekly, monthly, and quarterly documenting of your progress. You should be able to whip out a report in a matter of minutes if you have done your day-to-day work efficiently and effectively.

For a tester, your report should contain:

- **Any** issues that are **impeding** your testing or other **high priority issues**. Be sure to list these priority issues at the top of your report so that they are obvious and get the attention of the reader
    - Your lead should already be aware of these issues, but this is another opportunity to document the status of the issues and allows you to be on record regarding:
        - What testing cannot be done
        - Why you cannot complete it
        - What not being able to execute these tests is impacting
- **Status of the testing** that was assigned to you:
    - Detailed outline of **tests completed** – shown as a percentage as well as stating which tests were completed
    - Detailed outline of **testing** that is active or **remaining** – also show this as a percentage as well as stating which tests are not yet completed
    - Estimated **effort remaining** to complete the unfinished testing – also include the day that testing will be completed

Be sure to save all of the reports that you send so that you have them available for future reference (we will use them later). There are four sample Weekly Tester Reports in the back of this book. Use them as a reference to get an idea of what a weekly real-world report may look like. You should feel free to copy the format for your own use when writing your reports. That's what this book is about – targeted actions and usable tools!

© Phillip Bailey - All Rights Reserved

The 5 Keys When Getting Started

1. **The Testing Process**
2. **How To Execute Tests and Find Bugs**
3. **What To Do When You Find A Bug**
4. **How To Write A Bug**
5. **How To Deliver Test Results**

These are **the 5 Keys to your day-to-day work as a Software Quality Assurance tester**. If you can understand each of those sections and can master them, then your daily work life just became much easier. You now know what the tests are, how to test, and how to communicate your testing. These are the Keys to the daily life of a QA tester. The more you excel at these 5 Keys, the sooner you will be ready to lead a team of QA testers and to make sure that they can do what you just learned at least as well if not better than you.

I know that seems like a lot, but as you learn these skills and practice them you will progress until all of this information and each of these activities become second nature. You will not need to refer to this book any longer. You will not wonder what the right thing is to do next. You will know what to do. You will have the foundation to confidently state the proper course of action and know how to execute it. Keep practicing and learning and you will excel!

© Phillip Bailey - All Rights Reserved

Chapter #6: 10 Skills of Elite Testers

Vision without action is merely a dream. Action without vision just passes the time. Vision with action can change the world.
~ Joel Barker

There are many disciplines and methodologies to master in order to become an elite tester, and that mastery can take many years. But the key to developing these masteries lies in your fundamentals.

Just as with any endeavor, developing the fundamentals necessary to excel in your field is crucial. You may be able to get by for a year or two without grounding yourself in the true foundation of your craft, but you will quickly hit a ceiling that you will not be able to overcome without spending effort on the key ingredients that make a great tester, and you will never become a great Software Quality Assurance leader. So I encourage you to spend the time it will take to master these fundamentals that will empower you to rise to the top of your profession.

The 10 Skills of Elite Testers

1. **Communication**
2. **Bulletproof Bug Writing**
3. **Managing Expectations**
4. **Attention to Detail**
5. **Asking the Right Questions**
6. **Be Solution-Oriented and Proactive**
7. **Master Your Bugbase**
8. **Deliver Stellar Reports**
9. **Deliver on Your Word**
10. **Never Stop Learning**

Because each of these skills is so important, I have broken each into its own subchapter. **Don't rush through these.** Take your time and learn each one. They are all important and will all require practice to master. As you are learning these skills, keep in mind that knowing what to do is not enough, you must practice what you learn.

© Phillip Bailey - All Rights Reserved

A quick note about practicing what you learn.
It is said that "practice makes perfect." I disagree. This is a nice sentiment, but this is not quite accurate in my experience. I learned many years ago that how you practice is key to perfection.

Practicing results in training you to perform as you have practiced. If you practice ineffective behavior, you will perform ineffectively. The more you practice ineffective behavior, the more a habit being ineffective will become.

The key to mastery lies in *perfect* practice. Perfect practice makes perfect. Although you may not perform perfectly every time in every situation, if you strive for perfection, to do better each time, your performance will improve without fail.

So please, learn each of these skills and strive to practice as perfectly as possible. You will be amazed at the progress you will make in just a short time.

© Phillip Bailey - All Rights Reserved

Elite Skill #1: Communication

A positive attitude may not solve all your problems, but it will annoy enough people to make it worth the effort.
~ Herm Albright

Think like a wise man but communicate in the language of the people.
~ William Butler Yeats

Communication is the lifeblood of QA; communication between one tester and another or within a group of testers, communication between tester and lead, communication between QA and Engineering, communication between QA and the Producer/Project Manager, communication between QA and the Product Manager, communication, communication, communication!

There are two types of communication at which QA must excel; written and verbal. Communication with others on a daily basis is the most effective way for QA to build trust.

Trust in QA so that other departments know what QA is doing and are confident that QA is doing the right things and doing them in the right way. Trust with other departments that everyone involved in the project is on the same page and working in the same direction toward the same goals. Below I will outline the key pieces of communication most available to QA.

As I stress throughout this book and on my website, effective communication is a vital skill for any and all QA Professionals to develop. A skilled communicator can influence, convince, and persuade. If you expect to be taken seriously as a software tester or as a QA Lead then you need to develop and refine your communication skills (you probably won't even be given the opportunity to be a QA Manager unless you already have).

Skillful communication can be a tricky thing. It requires far more than simply stating your needs or what is on your mind. You must be able to deliver your point to anyone at any time in a way that they understand and react favorably to.

© Phillip Bailey - All Rights Reserved

This can be in writing (reports, bugs, impact analyses, etc.) or verbally (meetings, hallway chats, desk/cubicle drop-ins, etc.). Either way you must know how to communicate effectively.

Written Communication:

!!SPELL CHECK ALL WRITTEN COMMUNICATION!!

This is such a preventable mistake or omission and I just cannot emphasize it enough. Few things lower the trust that others have in QA more than a shoddy email or report. If QA can't be bothered enough to spend the effort to make sure that their bugs, emails, and reports are properly written, what trust should anyone have that QA is doing anything else correctly?

Some would (and have) argued, "But a quick 'heads-up' email to an engineer or project manager is just a courtesy I am extending them. Why does it matter if I punctuate it correctly or if I spell all of the words correctly?"

It matters because EVERY piece of communication that comes from QA represents the entire QA department. It is QA's job to take responsibility for *all* of the little details as well as any flaws in major functionality. If you can't communicate in a professional manner, no one is going to take what you have to say seriously.

If, on the other hand, your written communication is flawless, they will trust the testing you perform all that much more. This may not seem fair, but it is the perception you will have to live and deal with.

Any communication you make, whether written or verbal, **may be communicated up the chain**. Because of this, you had better ensure that what you say and write is accurate.

Spell checking your emails and reports takes only a few seconds and can make the difference between the information that *you* provide being trusted or not (and thus whether *you* are trusted or not). The few moments you take to ensure that your communication is as clear, accurate, and properly spelled will pay off ten-fold as you progress in your career as a QA professional!

© Phillip Bailey - All Rights Reserved

Trust me, you won't be sorry if you spend the time to do it right the first time, because you never know when a Vice President or CEO will be forwarded a key excerpt from your report.

If you have taken the time to proof it, you will be looked upon as a solid performer. If you can't put a coherent sentence together and haven't taken the time to ensure you have spelled everything correctly, then you will be seen as a possible liability that is in a high-risk position in the project – this will not bode well for your future.

Bug Writing

Writing bugs is the most common, everyday communication that a QA tester has with the rest of the project team. It is therefore my most fervent hope that you make sure that your bugs are clear, **have all of the necessary information** (and no extraneous, useless data), and are **easy to understand**.

If the person assigned a bug has to decipher what you have written, you have just made their job harder and created more work for them. Make their job easier and build the knowledge that they can trust you by writing clear, concise bug reports. And for the love of Joe – **SPELL CHECK** them before you submit them to the bugbase – you may not have the option of editing your original submission later!

Status Reports

Daily, weekly, or per project, these **reports are critical to the continued success of your team and your company**. Your reports that detail the status of your testing will be used by the company to gauge whether or not their product will ship on time and whether or not it will have all planned features included and functional. Be sure you are clear in advance about what information the recipient needs from you and then provide them that information in a clear and easy-to-understand manner.

When you send a Status Report keep these **3 simple keys in mind**:

1. List the most critical issues first at the top of the report
2. Articulate clearly the current status of your testing
3. Include the date you estimate your testing will be complete

© Phillip Bailey - All Rights Reserved

If you can deliver those 3 items clearly in your report, then anyone should understand where you are in the project. Your lead will be easily able to compile the information from the reports of you and your fellow testers into a report that will be useful to anyone in the company who wants to know the state of the project.

Test Metrics Reports

These **reports can detail several key pieces of information**. Among them:

- The amount of testing completed to date (often expressed by as a percentage)
- The number of bugs found (grouped by Priority level)
- The number of bugs closed (also grouped by Priority)
- The status of the testing (whether on schedule, ahead of schedule or behind)
- And help to set expectations regarding the timeline for completion of testing

In addition to delivering the hard data numbers listed above, **a Test Metrics report will communicate the impact of the numbers being shown**. It will explain why the numbers are important, from whence they are derived, and what they mean in real terms to the project.

As a tester, you will most likely not be asked to compile one of these reports. However, all of the reports that you do write and deliver will help to enhance the Test Metrics report that your Quality Assurance Lead or Manager creates and uses to support the project team.

Please **remember that all QA reports** whether they are bug reports, status reports, or metric reports **must explain the impact of the information they are delivering**. It is very easy to simply send someone a list of numbers and hope that they can make sense of them – but if they are not in context, the recipient may not find the numbers very impressive or even understand why you sent them.

To ensure that your reports are received with thanks and greeted by a trusting audience, **state your issues clearly and succinctly and explain the impact of the information you are imparting**. People will come to trust your reports and trust you. This will enhance others view of you and your QA team.

If you really want to ensure that your written communication is not only professional, but also persuasive…I've got some help for you.

© Phillip Bailey - All Rights Reserved

I spent many years honing my report writing skills. I suggest that you **shorten that learning curve.** *My* expertise is Quality Assurance.

I communicate effectively when writing my own reports, but the only way I have ever been able to truly teach report writing is by critiquing reports one at a time, one-on-one with my team.

Since I don't have the time to do this with everyone…and since it would take you several months and/or years to do this with me, I offer you an alternative. I give you a writing expert!

Learn to write effective reports, emails, or anything else! Learn in a matter of days, not months or years! Treat yourself to the life you deserve – learn to write effectively!

Dr. Ugur Akinci is a professional with over 20 years experience writing copy. Don't just take my word for it, go and see for yourself! What is your career worth to you? Use me as an expert on QA, but use Dr. Akinci as your expert on writing!

© Phillip Bailey - All Rights Reserved

Verbal Communication:

The other communication you will engage in is verbal communication. This sounds simple, but you would be surprised how many people cannot communicate clearly when they speak to others. Let's look at some simple, yet often overlooked, skills to develop so that you may enhance the efficacy of your verbal communication.

Be nice!

It is amazing how many people cannot seem to be bothered with asking questions politely or by phrasing their statements in any way other than "I am right and thus you must be wrong". Be nice and see just how many people will go out of their way to give you all of the information you need whether you ask for it or not. Be nice and note how many people will give you the benefit of the doubt if you ever make a mistake.

Listen

None of what else I have written will be useful to you if you do not learn to listen. There are volumes of books that cover the subject of, listening, learning to listen, being a good listener, etc. and so I will keep this portion brief and to the point.

If you are going to become a skilled communicator you must not only be able to send information, but you must be able to receive it. That means that when the other person is talking, you listen.

- Don't form an answer while they are talking. *Listen*
- Don't focus on how soon you can leave. *Listen*
- Don't interrupt them and finish the speaker's thought. *Listen*

Listen to understand what the speaker is saying. Listen to learn what the speaker means. Become an effective communicator by honing your listening skills. If the person speaking to you doesn't think you are listening – **they won't trust you.**

Think Before You Speak

Before you begin asking people questions that may be irrelevant, inaccurate, or only partially formed, make sure you know what it is that you want to know and phrase your question accordingly. Know what information you are seeking before taking up someone's time. Know the message that you want to deliver before you get someone's attention.

© Phillip Bailey - All Rights Reserved

Get To The Point
Tell them what you need to tell them. Ask them for the specific information that you need to know. Don't waste their time with a long, drawn-out story about the last week of your testing and why it is important to the world. They have a job to do too, so let them do it. Get to the point and move on.

Give Respect
Do you only speak kindly to people that can fire you? Are you impatient with others because you're sure they're not as smart as you, as detailed as you, as good looking as you? If so, you're going to have problems. Give respect. Work with everyone on your team as though they deserve it – because they do. If you work with your team, they will work with you. If they don't, they won't stay around very long because you will have helped create an environment where those that are not team players will stick out as liabilities.

If You Don't Know, Ask!
It's really as simple as that: If you don't know, ask. The quickest way to find out if the bug you just found is already in the bugbase? Ask! If it's in there, chances are that someone on your team has already seen it, so ask. They will tell you if it's there.

The quickest way to get clarification about a test that you are running and don't understand 100%? Ask! The person that wrote the test should be able to clarify what they meant – and this gives you both an opportunity to improve the test so that the next person understands it when they encounter the test.

If a developer has made a fix for a bug but their explanation of the fix is unclear to you – Ask them about it. If you don't know or understand how to test a fix – Ask.

I have found that most developers are more than happy to talk about the code that they have written. When you ask them about it they have a chance to talk about themselves and the cool kung-fu that they practice.

In addition to that, they are usually thrilled to have you, the tester, gain a deeper understanding of the program so that you can test it more effectively and maybe the next time you ask them a question they will get to talk about the ultra-super-secret kung-fu they have been working on…**so Ask. And ask politely.**

© Phillip Bailey - All Rights Reserved

Elite Skill #2: Bulletproof Bug Writing

Continuous effort--not strength or intelligence--is the key to unlocking our potential.
~ Winston Churchill

Being able to write a bug that anyone can understand is a key skill of being a true Software Quality Assurance Professional. If people cannot understand your bugs, if they are incorrect, if they contain misspelled words, or if they are not clear, then other people will have to do more work just because you haven't done your job effectively. This is never a good thing.

Since entering a bug is something that you will be doing for the majority of the days that you are testing, this is the communication that will be seen the most often by all of the members of your project team. Because of this, **you will be judged** (fairly or not) **by your skill at writing and entering bugs**.

The best feedback you can get with your bug writing skills is for someone to say "that was a good bug!" "That one was a good find!" This doesn't happen very often, but is rewarding when it does.

The most common reaction to a bulletproof bug is almost no reaction at all. The bug is reviewed and then, without question, the conversation turns to what to do about the bug; whether the project can withstand the cost and risk of fixing it. There is no recognition of the bug other than what to do about it now.

Although no praise is spoken, this is a great result! This means that there are no questions about what you wrote, only what is to be done about it.

Once you can do this consistently, you will have achieved something that many testers I have seen have never accomplished. You will have become a trusted member of the QA team that can be counted on to do their work right the first time.

You will be someone whose work is not questioned by other competent project team members. You will have built trust in your work and people will want you to

© Phillip Bailey - All Rights Reserved

test the most critical parts of the project. This is a key in being viewed as a true Quality Professional!

Be viewed as a professional and you will gain respect for yourself and your QA team.

Keys to writing a bulletproof bug:

Check For Duplicates FIRST

Once you find a bug, the first action you must take is to determine whether or not the bug has already been entered. Entering a duplicate of a bug that already exists only shows that you are not paying enough attention, you don't have a firm grasp of the bugs that are in the bugbase, and/or that you don't communicate well with your QA team (because someone should have known if the bug was already in the bugbase). If you skip this step and enter a duplicate without checking, you will be viewed as a liability.

So **first**, always check with your team and then in the bugbase to be sure that the bug was not entered previously. This will prevent the developers wasting time attempting to fix any issue that has already been fixed. It will also prevent you, the tester, from having to test the same bug multiple times just because someone didn't do their work properly the first time. Don't waste your time and the company's money duplicating your effort.

KISS

KISS stands for: **Keep – It – Simple – Stupid**. I didn't invent this, it is a long-standing principle. As you write your bug, do your best to write it as clearly and as simply as possible.

Projects become complicated enough on their own with their changing schedule, mutating features, late deliveries, prioritization meetings, etc. Make sure that your bug is an oasis of clarity among the complexities of the project.

Your bugs should be a welcomed respite for people to read and fix amid the confusion that will ensue in other parts of the project.

© Phillip Bailey - All Rights Reserved

Include An Unquestionable Reproducibility

I cannot stress this enough: **Make sure that the Reproducibility you state in your bug is accurate and complete**. Your bugs should have as a Reproducibility (as often as possible) of either Always or Intermittent. If your bug does not always happen, ask yourself: Did I do all of the work I could to determine for sure that this bug is Intermittent? Do your best to verify the Reproducibility you state in the bug.

If the bug is, indeed, Intermittent: You must state how intermittent it is. This can be stated as a percentage, but only if you include the "X-out-of-Y". The <how many times it occurred> out of <how many attempts made>. Lack of clarity in this area can lead to people not understanding the true risk involved in fixing or not fixing the bug.

Make Your Expected Result Crystal Clear

Your **Expected Result needs to state the exact behavior** that you had anticipated experiencing. With this expectation clear, it will be easy for anyone to understand exactly why the issue you entered is, indeed, a bug.

It may also point out where documentation is incorrect or incomplete since the test case you were running when you encountered the bug was based on some documented requirement or described behavior. If this is the case, then your bug may not be a functional issue at all, but instead a documentation issue that needs to be resolved.

For added emphasis, cite the requirement and/or document from which this bug has arisen. In this way, people can check the veracity of your bug against the documentation, and not against how thoroughly they believe you did your job. Send them to the specs instead of asking them to challenge their beliefs – this is a much more successful route to having questions answered. This will also help pinpoint where the documentation is in error or is incomplete.

© Phillip Bailey - All Rights Reserved

Always Spell Check

You may notice that I have mentioned this several times and here it is again: **ALWAYS SPELL CHECK!** It is easy, it takes only a few seconds, and it helps make you look like you know what you are doing.

A bug with errors in it, whether they are incorrect steps, an inaccurate result, or spelling mistakes, always make you appear as though you did not do a complete job. If you want to be viewed as a Quality Professional that can be counted on to be aware of even the finest details, you must ensure that you attend to those details. This includes spell checking your bugs.

Trust me…just run everything you type (bug, email, or report) through spell check. You can email me with the story of the first time you get caught not spell checking a bug if you like. Everyone forgets this eventually when they are under pressure from deadlines or a screaming, hyperactive boss, but the great ones don't make this mistake more than once a year.

Do yourself the favor and…**spell check** everything you write – THIS INCLUDES **YOUR BUGS.**

Review Before Submitting

After you have written up your bug, go back and re-read it. Look for information that can be removed without compromising the integrity of the bug. Once you hit <Submit> or <Enter> and your bug becomes a permanent record in the bugbase, you may not have a chance to edit what you originally wrote, so read through your bug before you submit it and make any last minute adjustments that you think would help your bug. Below are a few suggestions to keep in mind as you review your bug.

When reviewing your bug before you submit it:

- Think "Less is More" – can you make it more succinct?
- Make sure you haven't left out any Setup or Steps
- Be sure you have included an accurate Reproducibility
- Does what you have written make sense?
- Do you need someone else to read it and give you feedback?

Once you have reviewed your bug and are confident in it, submit it. Then listen to any feedback you are given when the bug is reviewed and learn what you can do to improve your bug writing skills.

© Phillip Bailey - All Rights Reserved

Once you are adept at writing and submitting a bug that no one questions, you will have enhanced how others on the project team view you professionally. They will become accustomed to the reliability of your work and how you report your data.

This is the most fundamental step for a Software Quality Assurance Tester. You must gain the trust of the developers and managers on your project so that they will trust you in helping to create solutions instead of questioning the validity of your bugs. Master this discipline and you will be well on your way to excelling as a Quality Assurance Professional.

© Phillip Bailey - All Rights Reserved

Elite Skill #3: Managing Expectations

The superior man is modest in his speech, but exceeds in his actions.
~ Confucius, *The Confucian Analects*

Another key component in your ability to build trust is your skill at managing the expectations that others have of you. If you want to increase your odds of putting yourself in a position to succeed (and no one else can be counted on to do this as well as you), you must have a clear understanding of what others expect of you and ensure that you can meet those expectations.

It is up to you to be sure that you can do what is being asked of you. If you can meet these expectations, you will be seen as a Quality Professional who can get the job done. This will lead to advancement, more responsibility, more opportunities, and more pay.

You can control what you agree to deliver. If you agree to deliver on something that you are not able to, you will be seen as someone that cannot be counted on. It does not matter to most people why you didn't deliver, only that you didn't. So, you must be aware of what you agree to.

It is imperative that people expecting results from you can count on them being delivered when they expect them to be. Make sure you know when your results are expected and be very clear about what those results should contain.

When testing is assigned to you, be very clear about when the testing needs to be completed and your results delivered. If someone is expecting your results in the next two hours and you take two days, then they will question what you were doing. They will question why they didn't receive the results they were expecting when they expected to have them. They may not say so aloud, but they will nonetheless feel that they cannot rely on you.

The first step in managing expectations is to clearly communicate *your* understanding of the expectations people have of you with them so that you are both on the same page. If you say you will deliver results by a certain time, be sure that you do. This will build trust. If you will not be able to meet that deadline, inform the people that will be affected as soon as you realize it. If your

© Phillip Bailey - All Rights Reserved

results are due by the end of the day, then 5:00pm is the wrong time to figure out that you won't be able to make the deadline.

As soon as you can see that you will not be able to complete your assignment and deliver your results within the agreed upon timeframe, inform the person expecting those results. Then make sure you tell them why your results will be delayed, when they can expect the results, and ask if it would be helpful for you to send them the results you have so far.

Although it is not the best thing to miss a deadline, people understand that life happens and that unexpected issues arise. So, if you cannot keep your commitment, the next best thing you can do is tell the person as early as possible, explain the delay (if appropriate), update your deadline with them (or for them), and offer your up-to-the-minute status (that I am sure you have, since you have been documenting everything).

By doing this, you will build a relationship of trust with those that you work with. They will trust that you will deliver on time because you will have ensured that you can do so when you first agree to the deadline for your results. They will trust that if something goes awry that you will keep them informed and not try to sweep it "under the rug". Since you are keeping them informed, they can communicate the status of the project as accurately as possible – this also makes them look good. They will trust you with more critical parts of projects because you have shown that they can count on you. **Build this trust with them by managing expectations effectively.** If you do, you will never regret it.

© Phillip Bailey - All Rights Reserved

Elite Skill #4: Attention To Detail

Weakness of attitude becomes weakness of character.
~ Albert Einstein

For a Quality Assurance Professional, this cannot be overstated. You are paid to be attentive to every single detail (whether this is a realistic expectation or not). Since there is no way that you will remember absolutely everything off the top of your head (many have tried – all have failed) you must take the only step that will save you when you are asked to recall the most abstract piece of minutiae – **Document Everything!**

I have a friend who used to say, "I really do know everything…just not all at the same time."

You have a choice:

- Make a few notes here and there and **hope** that when the roulette wheel of questions spins and it is up to you to answer that you will have the most up-to-date, accurate information

– OR –

- **Document Everything!**

Which do you think gives you the best chance of being able to answer a Vice President of Engineering's question regarding the status of a bug you entered last month or how long it will take to complete a suite of tests? If you document what you do then you will have this information.

If you don't have the information with you in the meeting when you are asked, you can say, "I don't have it in front of me, but I can either go get it for you right now or immediately following the meeting." This is a fair response. As nice as it would be to have all information in front of you and reference-able at will, the next best thing is to be able to deliver it within a few minutes after the meeting has adjourned.

The only way to be able to answer any and all of the myriad questions that will come your way is to have documentation that you can reference. If you are able to answer the many questions thrown your way and every time you are asked a question you are able to deliver an accurate answer, then you will be viewed as a

© Phillip Bailey - All Rights Reserved

trustworthy professional – no matter the amount of experience you have in your current position!

So do yourself a favor and…wait…for…it…**Document Everything!**

You will also use this documentation when compiling and writing reports and when asking for a raise. You did say you wanted to make more money didn't you?

When you attend to and document these details, you will have the information needed when it comes time for you to show off how valuable you are. You will be able to answer any question about your project, about your contribution to it, and about what you have done in your time on the project to add value. You can showcase the efficiencies you have enacted, you can chart the progress you have made as an employee, and you can explain in detail exactly why you should be given a promotion and a raise.

© Phillip Bailey - All Rights Reserved

Elite Skill #5: Asking The Right Questions

We are a bit of stellar matter gone wrong. We are physical machinery - puppets that strut and talk and laugh and die as the hand of time pulls the strings beneath. But there is one elementary inescapable answer. We are that which asks the question.
~ Sir Arthur Eddington

Each member of a project team has a role to play. One part of the team has to decide what to create, another is responsible for tracking the project's deliverables and milestones, yet another group will focus on creating the program, web page, database, etc. As a Software Quality Assurance tester, your daily role is to work through the program to validate and report its stability. This part of QA is done through daily testing of the program. In order to test most effectively, QA must ask the questions that the other members of the team have overlooked.

The development team has been tasked with creating a working program based on the marketing requirements they have been given. Their job is to work out the logistical issues that arise from taking an idea and making it a working reality. They are not going to be focused on all of the infinite possibilities that will affect their creation in the real world. This responsibility falls on QA.

Ensuring that all of the variables are accounted for is the area where QA can add yet more value to the project. This is an often overlooked ability that QA can bring to the table.

Most run-of-the-mill QA testers may complain that there is no way a program will work as originally envisioned or will whine that there is no way they would ever purchase the product. **This is a golden opportunity that many QA people miss.**

If there is something wrong with what is being created, if there is question as to the value of the product to the target market, QA is the team that has this information readily available to them. This is because QA is using the program EVERY DAY! No other part of the team is as intimately aware of how the program is performing!

© Phillip Bailey - All Rights Reserved

The idea must have looked good on paper. And the developers probably wrote some very elegant code to tackle all of the complex logistics of making the product viable. But only QA is using the program every day. Only QA will see every time there is a resource conflict with another program, or if the sign-up method creates a path for a user to *not* be able to upload their videos, or any number of myriad circumstances that make the product a burden for the end-user to use in some way.

When this happens, QA must bring it to the attention of the project team. When the implementation that made sense on paper results in an issue for the end-user, it is the duty of the Quality Assurance Professional to bring this issue into the light. QA must alert the project team that the planned implementation is not as seamless as was assumed nor as joyful as had been hoped.

To put the issue in the proper light, the Quality Assurance Professional must ask the right questions for the end-user. It is possible that the implementation matches the product requirements, but maybe the requirements did not take all the variables into account. This can result in a product that matches the specifications, but does not meet the end-user's needs.

QA must ask the right questions:

- What need is this product intended to meet?
- Knowing the target market: Does this implementation truly meet the need as intended
- Does the implementation create a conflict with other software that the target market is known to use?
- How could the usability be more intuitive for the end-user?
- Does the program create a user path that will be confusing for the end-user?
- Will this product truly add value for the end-user or does its use create a burden?

This goes beyond just logging bugs! These are issues that will affect how the buyer views your product! Certainly these issues should be logged as bugs, but be aware that you only set the Severity of the issue, not the Priority. If you have logged an issue with a moderate to low Severity and you see that it has been given a low Priority, question that prioritization! Bring this issue to the attention of your Lead or your Manager or the Project or Product Manager. Explain to them the true end result of the bug and clearly state why you think the issue is more important than the Priority it has been given.

© Phillip Bailey - All Rights Reserved

They may not have thought through all of the possibilities that you see. Maybe they have but they forgot about them. Maybe they think that the issue will affect only a very small percentage of their target audience. If you have done your homework, you will know what percentage of the buyers will be affected and this is why you are bringing it to their attention.

This does not mean that all of the issues that you raise will be fixed in the current version of the product you are testing, but it does give the team the opportunity to look at the situation as you see it. If it is important enough, they will fix it in the current version. If the risks associated with fixing it are too great, at least they will have the opportunity to plan to fix the issue in the next iteration of the product. Being seen as a member of the team who cares enough to as the right questions will gain you trust and respect with your team members.

You must also have what I call a **"What if..?" attitude.** As you are performing your testing, you must always be thinking of "what would happen if..?" This will enhance your ability to test more effectively. No single test suite can cover every possible circumstance and so it is left to the tester to think outside of the box, to explore the Art of QA, to go beyond the bounds of the stated testing, to poke and prod the program, and ask "What if..?"

Make asking "What if..?" your mantra. Embody this question in all of the testing you perform and you will find more bugs, find bugs that are more deeply hidden, and add value to all that you do by being the one who asks the questions that others have not.

© Phillip Bailey - All Rights Reserved

Elite Skill #6: Be Solution-Oriented And Proactive

When trouble is solved before it forms, who calls that clever?
~ Sun Tzu

It's not enough that we do our best; sometimes we have to do what's required.
~ Sir Winston Churchill

To separate yourself from the crowd, you must be a person that finds a way to get things done. This is true in any undertaking and is a golden opportunity to the Software Quality Assurance Professional.

Often a QA environment is filled with people that wanted to get a job that paid something and they knew how to use a computer and they thought "anyone can test" so they got a job with a company testing games or apps or databases. There is nothing wrong with this, but they often don't take their job more seriously than something that allows them money to eat, pay the rent, etc. When they encounter difficulty at work, they do their best to minimize this pain by passing the problem off to someone else.

As bothersome as this attitude can be to deal with every day from multiple team members, this is a golden opportunity for you to shine. Be the part of the team that sees these obstacles as challenges, as opportunities, as a chance to learn something, to improve your team, to deliver superior results, and be viewed as one who gets the job done no matter the challenge!

Let's face it, in today's economy the chances of getting a job with one company, working there for 40 years and retiring with the company paying your pension are just about zero! So as long as they are paying you to test, take advantage of every opportunity that comes your way to expand your knowledge. Every challenge you encounter is an opportunity to improve yourself through the educational experience of learning a better way to do things. You can take this knowledge with you wherever you go and it will make you more marketable.

© Phillip Bailey - All Rights Reserved

It is not uncommon when there are deficiencies in the test suite for a tester to complain that:

- We're not testing everything
- We're not going to know if this thing really works like they want it to
- They suck

This "who cares…it's somebody else's fault" attitude is unfortunate, but not uncommon. It does nothing to make one's daily work or life more rewarding. It puts the blame for the world's ills at someone else's feet and absolves the speaker of all responsibility. But you have a chance to do things differently.

If you see that a test suite is not as complete as it should be, tell your lead. Ask if you can help update the test cases so that they are more complete for the current round of testing and for the future. This will show that you are aware of what is not getting covered, helping to build trust in you that you know how to do your job, and will provide you the opportunity to get your hands dirty learning a new skill (if you have never written a test case before).

Or, before approaching your lead, you could do the testing that is being missed by the test cases. Then (since you document everything) you will have ammunition to use when you speak to your lead and say, "I noticed that there were no cases that covered <circumstance X>, and so I tested that scenario and logged 6 bugs. Do you think it would be ok if I added the test cases that I executed to our suite of tests so that we are sure to cover this area the next time?" That is a much better encounter than leaving your boss to answer questions about why an area of the program is not being tested.

You will make your manager's life easier, make them look better at doing their job, and be seen as a true Quality Professional who is out to do the right things and do them in the right way. You will be viewed as a person who brings solutions to the table and is proactive about implementing them.

To be proactive is to be able to identify issues, small and large, before they trigger an alarm across the company. It is in developing the ability to point out where the hurdles will be, alerting the appropriate party, and then suggesting ways in which you can minimize their impact.

© Phillip Bailey - All Rights Reserved

A proactive Quality Assurance Professional:

- Reviews appropriate project documentation without being directed
- Brings questions to light that need clarification
- Notes where documentation is not complete, knowing this will result in lack of thorough testing coverage, and motivates completion of the documentation
- Is alert to others that find a bug they know is already logged – and save the team time by directing them to it
- Asks what they can do to help their QA Lead or Manager improve the team and its processes

Being proactive is an attitude that can be learned. It is a behavior that can become a habit. But it takes practice.

Look for opportunities to create solutions before problems get out of control, before everyone must drop all that they are doing to address the problem. Then when you identify these problems, see them as challenges and take action! Identify the issue, create multiple solutions, and then present them to your Lead or Manager. They should be happy to hear that you have identified a hurdle and come up with an answer.

Practice being proactive and it will become an invaluable habit!

This is a skill that you can use in any circumstance, at any company, in any job! Learn it and take it with you! In the meantime, take every opportunity presented to you as a chance to learn and to improve. If you take this lesson to heart, you will never regret it, I promise!

© Phillip Bailey - All Rights Reserved

Elite Skill #7: Master Your Bugbase

The sweat of hard work is not to be displayed. It is much more graceful to appear favored by the gods.
~ Maxine Hong Kingston, *The Woman Warrior, 1976*

Great ability develops and reveals itself increasingly with every new assignment.
~ Baltasar Gracian

There are few things that you can spend your time on that will yield more information about the state of your projects than your bugbase. Therefore it is your assignment to master it! Being able to use your bugbase effectively will be key in being able to show your true worth to your team and your company.

Mastering your bugbase will allow you several opportunities to showcase your value:

- Comprehensive knowledge of all the issues preventing your product from shipping and their status
- The ability to enhance your reports with bug statistics
- Prevent you and your team from entering duplicate bugs
- Decrease the time you spend searching for issues to determine if they have been previously entered
- The ability to separate yourself from the average tester by doing all of the above

Although no two bugbases are identical, mastering them is really a simple process. It is not rocket science, even if you are testing rocket components.

In my years working in Software Quality Assurance I have used no fewer than 10 different bugbases. I have used FileMaker 3.0, 4.0, 5.0, DevTrack, TestTrack, Bugzilla, FogBugz, and Jira to name just a few. The principles are always the same, only the setup and search features really differ. Below I will give you a few simple tips that, once mastered, will make you king (or queen) of your bugbase and have people coming to you to help them massage it as only you can.

© Phillip Bailey - All Rights Reserved

Review Bugs Daily

This is very simple. The first thing that you must have command of are the contents of the bugbase. This means being knowledgeable about every Active bug. Although this may seem a daunting task as first, break it down into bite-sized chunks and you will have all of this information in your head in no time.

Your first step is to spend 30 minutes each morning before you begin testing, reviewing the Active bugs. If you do this, you will know them all in no time.

Where do you find these 30 minutes? Either speak to your Lead and tell them what you are trying to accomplish (having a clear understanding of all the Active issues logged against the project so that you can be a more useful member of the team) and assure them that this will not take more than 30 minutes each morning, or get to work early. This is simple, but it is all on you. Get to work before you are expected to be working on anything else and study the bugbase for 30 minutes. For most effective use of your time:

- **Begin by reviewing the most recently entered Active bugs** and work your way backward in time by date entered. This will get you up to speed on the issues that are currently assigned to be fixed in the current release. You will quickly learn where the program is the weakest, what issues have not yet been addressed, and which bugs are being given what Priority level. This will give you the informed opportunity to spend your testing time the most effectively and efficiently. You will not spend time testing areas that have been completely covered or have no weaknesses in the code (as if that were possible); you will know how to attack the program to unearth the greatest vulnerabilities.
- **Review the highest priority bugs first** and work your way down the prioritization ladder
    - Thus review all of the Priority 1 bugs first from most current to oldest
    - Then review all of the Priority 2 bugs from most current to oldest
    - Select the Priority 3 bugs…rinse…repeat
- Once you have completed the review of the Active bugs, **move onto any Deferred bugs**. By having and understanding of what bugs have already been Deferred, you will have insight into the issues that are slated for a subsequent release. You will know what features are not considered a priority in the current release and can minimize the time you spend exploiting those known weaknesses. Or if you feel strongly that a Deferred bug should be addressed, you will have the information you need

© Phillip Bailey - All Rights Reserved

to be proactive and show in what other ways that Deferred bug will adversely affect the end-user.
    - Review all Priority 1 and Priority 2 bugs that have already been deferred (if any)
- Then **move onto the Closed bugs**. Once have inventoried the Closed bugs, you will have a more complete history of the project. You will see what bugs have been logged and fixed and can see what the fix was. This should help enhance your ability to effectively test the program going forward since **you will have researched the lifecycle of the bugs** that have already been logged and addressed.
    - Review Priority 1 and Priority 2 bugs that have been closed within the last month or go back 100 bugs, whichever comes first. Don't go back any farther than that during this exercise – this will get you the information you need to get started
    - Pay attention to what the fix was (this will help you understand the current behavior of the program and give you an idea of how your development team addresses bugs)

Once completed, you should be up to date with all of the bugs in the bugbase. You should have a high-level understanding of the major issues that have been logged against the project so far. Your next assignment is to **set up two searches (or filters) that you will run every day**. These two searches are:

- Bugs Opened today
    - Include all bugs that are newly Opened or that were Closed but got Re-Opened today
- Bugs Closed today
    - This includes (you guessed it) all bugs Closed today

Set these searches up in your bugbase and save them so that you can run them every day. Then make sure you run them each and every day. Run the searches and review all of the resulting bugs. Since you are up to speed on all the Active issues, this shouldn't take you more than 15 minutes each day.

Is that worth it to you? It should be. Spending 15 minutes a day to know the Status of all bugs that affect the project is one of the highest return activities you can invest in for yourself as a Quality Assurance Professional.

© Phillip Bailey - All Rights Reserved

With the knowledge you gain from this daily review, you will:

- Be able to speak accurately to questions about any bug affecting your project
- Know when you encounter a bug whether it has already been logged or not – thus preventing you (and your team) from entering duplicate issues
- Have an insider's view into how issues are prioritized and addressed during a project
- Be able to focus your testing on the what you know are the weakest areas of the program
- Have the ability to compile your reports with detailed bug metrics much quicker

With this knowledge and these abilities, you will:

- Be viewed as a go-to person on the project
- Increase your productivity by spending your time testing more effectively and efficiently
- Improve the overall performance of your team, thus making your Lead and Manager look better in the eyes of *their* bosses
- Deliver reports that will show what an outstanding performer you are
- Be seen as an asset that should be developed so that the company will reap even greater benefits by employing you

© Phillip Bailey - All Rights Reserved

Master the Search Functions

If you have been able to get listings of bugs as described above then you have begun your education about the search functions in your bugbase. Each bugbase has slightly different search capabilities and methods. Some allow you to enter multiple search criteria at once for a single query that will yield the results you desire. Others will require you to learn the filtering system that the bugbase uses. If this is the case, you will need to understand what each filter selection that you enter will yield. Fortunately all bugbases have a help feature or link that you can use to familiarize yourself with its functionality.

Read through the Help provided by the bugbase to get an understanding of how its search and filtering capabilities work. Explore these features and practice until you are able to create searches with the following parameters:

- **Active Bugs:**
  - Opened by a specific tester
  - Assigned to a specific tester or developer
  - Opened during a designated timeframe
  - Opened due to testing a specific section of the program
  - Opened due to executing specific portions of your test suites
  - Opened by software (the program you are testing) version number
- **Resolved Bugs:**
  - Resolved by version number
  - Sorted by Resolution Type (Fixed, Deferred, Will Not fix, etc.)
  - Resolved within a designated timeframe
- **Closed Bugs:**
  - Closed by a specific tester
  - Closed within a designated timeframe
  - Closed by version number
  - Sorted by test suite section
- **Deferred Bugs:**
  - Deferred by version number
  - Deferred within a designated timeframe
  - Sorted by test suite section

****Note:** Not all bugbases will give you the option to search and sort by all of the criteria listed above, but you should still be able to have it yield most of these requests.

© Phillip Bailey - All Rights Reserved

Practice until you are able to get your bugbase to give you each of the results listed above. Once you do, **save those searches (if possible)** so that you don't have to remember how to do the search each and every time.

Having these searches readily available to you will give you the ability to:

- Be fully aware of the Status of your project at all times
- Know if you are testing a build that is particularly feeble or robust
- Track your bug finding, bug logging, and fix rate
- Answer questions about the Status of all bugs related to the project when *anyone* asks
- Give you the data you will want to include in your status reports to show off how valuable you are to the project and to your team

Learn the Export Feature

Once you are able to get the search results that you desire, the next step is to learn how to have the bugbase export them into a graph. This graph can show how many bugs you have found and gotten fixed and how many have *not* been Deferred, sorted by Priority, by version number, by test suite section, etc. You will want to add this information to your reports and keep track of them for use when you are ready to ask for a raise.

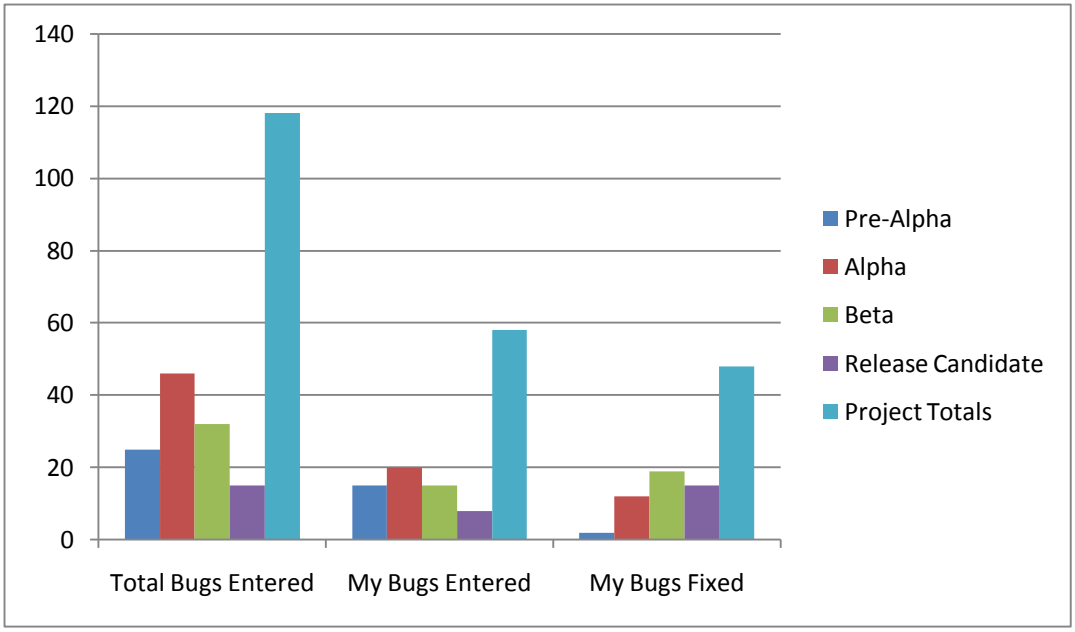
When someone sees a number like 48%, they know that they are looking at a number that represents almost half, right? But do they really see the impact of what that means?

When you are able to combine your hard-data numbers along with a pretty little graph to illustrate what those numbers mean in relation to the whole, it is much more impactful to your audience. In the example below, you have culled data from the bugbase showing in each phase of the project what sort of impact you had.

Here you show:

- How many bugs were entered total – this gives the baseline that you will be measuring against
- How many bugs *you* entered – in this example, almost half of all bugs during the project
- Of the bugs you entered, how many were fixed – this shows that the bugs you entered were of high enough severity and priority, and were written well enough to quickly be addressed

© Phillip Bailey - All Rights Reserved



Can you see how being able to actually *show* your boss how many bugs you have found, what percentage they represent of the total, and the fix rate of your bugs would have an immediate impact in showing off your worth to the team? You can show this as a chart (as above), or a line graph, or a pie chart, or whatever style you like. The key is to be sure that it is comprehensible to those you will show it off to.

Which do you think would give you a better chance of getting a raise or of being promoted?

- "Look boss, I work hard here every day and really help out the team. I find great bugs! Just ask anyone, they'll tell you."

– Or –

- "In the last 5 weeks on this project I've been responsible for logging 43% of the bugs that have been found. Of those, 74% were Priority 1, and to date 87% of them have been fixed. I find and log more bugs per hour spent testing than any other tester on the project." And as you say this, you are showing your boss each graph/chart that you have prepared from the searches above

Do you see how the second scenario would give you a better chance to impress? You are using the numbers that are available to you and backing up everything you say with hard evidence. You boss may question you, but if you have done your homework correctly all of your data will be accurate.

© Phillip Bailey - All Rights Reserved

Keep in mind that your boss will probably simply be impressed that you are able to present your arguments in this way. This is how QA should present their data, show the numbers. Your boss will be impressed, as well he/she should be. You should now be top of the list for more money, responsibility, promotions, and/or development by the company.

*See? Numbers **are** your friend…*

Elite Skill #8: Deliver Stellar Reports

When your work speaks for itself, don't interrupt.
~ Henry J. Kaiser

Part of the inhumanity of the computer is that, once it is competently programmed and working smoothly, it is completely honest.
~ Isaac Asimov

There are several types of reports that you may be called upon to deliver. Daily, weekly, or per project, these reports are critical to the continued success of your team and your company. Be sure you are clear in advance about what information the recipient needs from you and then provide them that information in a clear and easy-to-understand manner.

Daily Reports

These reports are generally a summary of your day's activities. They answer the questions regarding what you did and what happened. This can include:

- Highlight any major issues you have encountered (which I am sure you communicated previously to your lead at the time of the event). This can include major new bugs entered (6 new crash bugs found today), issues that are preventing testing (can't test the Help menu because it won't open: Bug#123), etc.
- Tests you performed and completed
- Tests you performed and are still in progress
- The results of the testing you performed that day; number of bugs found (newly entered and existing bugs updated), number of bugs regressed, number of bugs closed, etc.
- Communicate the impact of any impediments to testing. If you can't test an area of the program but should be able to (and were assigned to test that area today), then let your lead know how much time and effort remain to complete that testing.
    - For instance: You were assigned testing the Help menu today. This Help menu testing is forecast to take 2 days. At this point, your lead is expecting that testing to be done by the day after tomorrow. Since you are unable to test this area due to Bug#123 – be sure that your lead knows that the testing will not be done for at least one day beyond their original expectations

© Phillip Bailey - All Rights Reserved

- What percentage of your assigned testing is completed
- When you expect to complete your assigned testing

Remember that you can always include the hard data that your bugbase reports give you for added emphasis. This will make your reports more impactful and raise the opinion people have of you when they read your reports

Weekly Reports

These reports are the summation of the status of the parts of the project that you worked on that week

- Highlight major issues
- State your information as objectively as possible
- Add the bug report graphs that show your progress
- Include what is on your plate for the upcoming week
    - What parts of the project are you expected to work on
    - Estimated completion date for each assignment
    - If you have nothing assigned to you, point it out here and be sure that you have your lead assign you tasks so that you can plan your work

Again in this report, be sure to **add hard numbers data.** Create an expectation that this is the information people will receive when you deliver a report. This will accomplish two major things:

- You will be seen as a reliable source of information that can be trusted
- You will raise the bar for your team to aspire to. This creates an environment where you and your team will be viewed as professionals. Anyone who is unable work as a professional will be weeded out sooner than later. This will make your work day more rewarding and enjoyable.

As a tester, those are the two main reports that will probably be asked of you (if even those). The other report that you will want to be able to deliver (whether they tell you this at the outset or not) is an End of Project Report.

© Phillip Bailey - All Rights Reserved

This report should be used for the Post Mortem meeting:

- An **End of Project Report** should be a high-level summary of the project upon its completion
    - Include total number of bugs found
    - Total number of bugs fixed
    - Total number of bugs deferred
    - Make them sortable by Priority, by test suite section, etc.
    - Detail what went right with the project (from a QA perspective)
    - Detail what could be improved for the next project
        - And list suggestions to achieve this improvement
    - Detail what you learned during the project

As you can see, you will add hard numbers data that you have by now mastered acquiring from the bugbase. The rest of the information you should use for two purposes:

- You have a ready report for a Post Mortem meeting
- You have an record of your contribution to this project

Having all of these reports **readily accessible will be the foundation you use when tracking your progress.** It will be accessible for you to reflect on when trying to remember how you did certain things during the project. Use this information to show your boss why they should promote you – if you can compile this report, you obviously are a keeper and the company had best "pony up" if they plan to keep you around.

© Phillip Bailey - All Rights Reserved

Elite Skill #9: Deliver On Your Word

People forget how fast you did a job - but they remember how well you did it.
~ Howard Newton

To be viewed as a true Quality Assurance Professional, you must be someone that others can count on. This is a perception of you by others that takes time to develop. To do this you must consistently do what you say you will within the timeframe you set forth.

As easy as this seems, many people are not able to deliver on their word. People often say they will do things that for whatever reason, they then do not do. There are several reasons for this:

- They want to be liked, thus they make a promise on which they cannot deliver
- They want to please, to be seen as someone who can do whatever is asked, so they agree to anything without regard as to whether or not they can meet the request
- They want to be thought of as smart, so they agree to whatever is asked
- They don't understand the request and so cannot deliver what has been asked of them
- They think that the more they take on, the more valuable they are – regardless of whether they can do any of what has been asked of them in an effective manner

These pitfalls can be avoided if you are willing to manage expectations and really take a hard and honest look at what you are capable of delivering. An honest self-assessment is something that trips-up most people when it comes to the ability to meet expectations.

Self-assessment is beyond the scope of this book, but is a key factor in truly being your most effective self. Here is what I will offer in this area: As part of your commitment to yourself to continue to learn, get some books or take a course on self-assessment so that you can understand your real capabilities.

Here is some of the material that I have found very useful in this arena:

© Phillip Bailey - All Rights Reserved

- The 7 Habits of Highly Effective People – by Stephen R. Covey
- Now, Discover your Strengths – by Marcus Buckingham and Donald O. Clifton
- The Success Principles – by Jack Canfield and Janet Switzer

As for Managing Expectations I wrote earlier that you must:

- **Understand the request**
- **Know when you must deliver your results**
- **Have clarity regarding the type and content of the results that are desired**

With the three above pieces clearly defined, you should be able to deliver what you have agreed to without fail. Once you have those elements, the last piece is to ensure that you do, indeed, deliver.

One other action to take that will help in the perception people have of you is to **"under-promise and over-deliver".** This is much more impressive than the reverse. If you over-promise and under-deliver, people will not have patience with you for very long, won't trust you or your work, and you may be headed out the door sooner than later.

Simply agree that you think you can do the assigned task within the parameters set forth and then deliver your results **sooner** and **more completely than expected**.

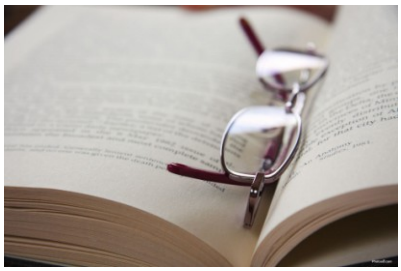
For an example of this behavior, go look up some old Star Trek episodes (the original series) and watch Mr. Scott (Scotty) the engineer. He *always* under-promised and usually over-delivered – the true key to that character's success.

© Phillip Bailey - All Rights Reserved

Elite Skill #10: Never Stop Learning

Seize the moment of excited curiosity on any subject to solve your doubts; for if you let it pass, the desire may never return, and you may remain in ignorance.
~ William Wirt

Employ your time in improving yourself by other men's writings so that you shall come easily by what others have labored hard for.
~ Socrates



This skill is critical for you to advance in your career as a Quality Assurance Professional. You must always strive to learn more. Learn something new every single day. Read, explore, pick a skill to master, do something! In the world today, if you don't continue learning, the world will pass you by.

As I noted in the beginning of this book, there are boundless opportunities to be had in the world of Quality Assurance. For every piece of technology that becomes "obsolete", three more seem to takes it place. The only way you can keep up with the pace of technology is to always be learning something. Continually add to your knowledgebase.

Most, if not all, of the information you acquire will be applicable in a number of different ways – if you learn the principles. If you understand how one database works, it's simply a matter of modifying your terminology or learning the newest wrinkles to master a second one. Technology is ever-changing, but the human mind isn't evolving as quickly.

The technology you will be asked to test, and guide the testing of, will be conceived of and implemented by humans. If you therefore learn the principles, you will be able to adapt in today's global economy. So start somewhere…anywhere.

Pick something and start learning. Learn about web marketing. Learn about your company's target market (something you should always do anyway so that your testing is focused most effectively). Learn about the newest ways to secure a network. Learn about email usage trends. Learn about audio mixers. Learn

© Phillip Bailey - All Rights Reserved

about process improvement. Learn about Project Management so that you can relate to the Project Managers on your team more effectively. Learn something!!

It really doesn't matter what you choose to focus on first, just pick something and start. By doing this, you will be WAY ahead of most people. Learning takes effort. It's work. It can be fun work, but you will have to make the time and spend the effort…you will have to THINK! This is something that the majority of people avoid if at all possible – sad, but true.

Henry Ford is credited with saying two things that I find appropriate for this chapter. They are:

- *Anyone who stops learning is old, whether at twenty or eighty. Anyone who keeps learning stays young. The greatest thing in life is to keep your mind young.*

– And –

- *Thinking is the hardest work there is, which is probably the reason why so few engage in it.*

I have found both of these sentiments to be true.

Often, unfortunately, in a QA environment there are many people who see their QA job as a burden and not as an opportunity. They miss the opportunity to learn new things that could open doors for them so that they could find an area where they would truly excel.

If you decide to take on the challenge of continual learning, you will be rewarded many times over! You will find avenues open to you that never would have been otherwise. You will have the ability to assess a situation or a job opportunity with greater perspective and make a more informed decision about what you choose. So just pick something, start somewhere, anywhere and start learning.

If you don't like the first subject you choose, then move on. You probably learned something from the experience as well as the beginnings of some expertise, so you haven't wasted your time. Now you are simply closer to finding what really does interest you – these are the things that are fun to learn. If you choose something that you find you can't stand, you have eliminated one thing. By process of elimination, you have increased your odds of finding something you really do enjoy.

People I have worked with over the years know that I always have a book with me that I am reading. It might be Stephen Covey's "7 Habits of Highly Effective

© Phillip Bailey - All Rights Reserved

People", or "Marketing that Matters" a book in the Social Ventures Network series, or "Top 10 Challenges of Software Testing".

It doesn't matter where you start, just start! This will enhance your life, open more doors for you, and help you understand the world at large with greater depth and clarity. The greatest Quality Assurance Professionals I have worked with all do this and they never stop. Invest your time and money in yourself – you will make the world a better place.

If you want to give yourself the chance to be the best you that you can be – discover your true self and learn to use all that you possess to excel…**check out one of my favorite writers**. He offers the ultimate way for you to invest in yourself.

The more self-aware you are, the better you can harness your strengths. If you give yourself the chance, you can improve every day.

One of the things I most appreciate and respect about Dr. Wayne Dyer is that he gets better year after year. I don't love everything he has written – but he has improved with every work.

That's the goal isn't it? To be better than you were. I believe he has done (and continues to do) that. He practices what he preaches.

I find no greater goal: *Be better than you were yesterday* – each day. If you can do that, then your professional life can remain in synch with your personal life.

Don't worry about the next "guy" – focus on you. If you can become just a little better every day – better than you were yesterday – you will set your own standard of quality and become one of the truly Elite.

Give yourself the opportunity to achieve greatness. Open your life to the wonders that the world has to offer. Prepare yourself to see the opportunities and seize the moment. Start now and join the coming Innovation Economy!

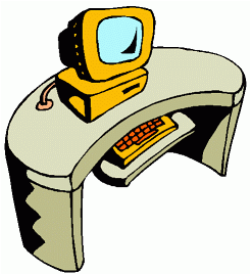
© Phillip Bailey - All Rights Reserved

These are the 10 Skills that you can master to build a solid foundation. Strive to excel in each of these disciplines. It takes time to turn these behaviors into habits, but as you work toward mastering them, you will find that even the incremental improvements that you make will be rewarding.

I know that it seems like a lot of behaviors to change and many skills to master. You may be wondering if this is something that you really can do. What I can tell you with certainty is that whether you succeed or not; it is completely up to you! The choice is yours. You have the option of success or not. You are the one in control.

The 10 Skills of Elite Testers

1. **Communication**
2. **Bulletproof Bug Writing**
3. **Managing Expectations**
4. **Attention to Detail**
5. **Asking the Right Questions**
6. **Be Solution-Oriented and Proactive**
7. **Master Your Bugbase**
8. **Deliver Stellar Reports**
9. **Deliver on Your Word**
10. **Never Stop Learning**



I know that these skills can be mastered because I have worked with multiple teams of testers that have succeeded in doing just that. I have had the distinct honor to work with three different teams at three different times that took these skills seriously and achieved mastery of each. It is up to you. Only you can choose for yourself to succeed.

© Phillip Bailey - All Rights Reserved

Chapter #7: The 10 Critical Elements of a Bug

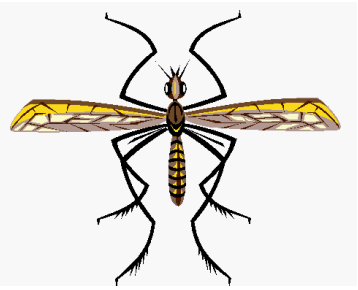
It doesn't matter how beautiful your theory is, it doesn't matter how smart you are. If it doesn't agree with experiment, it's wrong.
~ Richard Feynman

"I don't care if it works on your machine! We are not shipping your machine!"
~ Vidiu Platon

Every bug is made up of multiple individual parts. Singularly, each piece (or Element) contributes to the integrity of the "whole" of the bug. Leave out one Element and your bug is incomplete. Omit just one Element and your work will look like that of an amateur. But if you can master the content and use of each Element detailed in this section, your bug writing skills will make you and your QA team proud as you demonstrate the unerring accuracy of a true Quality Assurance Professional.

The 10 Critical Elements of a Bug

1. **Brief Description**
2. **Expanded Description**
3. **Reproducibility**
4. **Steps to Reproduce**
5. **Result**
6. **Expected Result**
7. **Severity**
8. **Priority**
9. **Status**
10. **Assignee**



There are other elements that may be necessary for your bugbase, but these are the most important ingredients to start with because you will need all of them no matter which bugbase you use. Let's take them one at a time.

****Note:** You can use the Sample Bugs in the back of this book as reference while you read through this.

Element #1: Brief Description

This is a succinct, one line description of the issue you are entering. It should be short enough that anyone browsing a list of issues by just this one piece of information is able to determine what the issue is. Think of a single computer screen full of single sentence descriptions. The list may be 100 or 500 bugs long. This is how the Brief Description will appear when anyone does a query in the bugbase. The goal to writing the Brief Description is that anyone scanning a list of 500 bugs can easily tell one from another just by the Brief Description.

Sometimes it is easiest to write this piece last, after you have the bug plainly defined in front of you and clear in your mind. Even if you write this part first, be sure to review it after you have finished writing the rest of the bug to ensure that it is still accurate and clear.

Element #2: Expanded Description

Positioned at the top of the "Bug Description" field, the Expanded Description should be a precise explanation of what has gone wrong. This can be the exact same as the Brief Description, but is also the place where you can go into more detail to make the issue as clear as possible to anyone reading the bug.

This is where you can elaborate on the issue and use more than a single sentence to describe what is happening. Even though you can be more verbose here, keep in mind that simplicity should be your guide. Explain as much as is necessary, but no more than that. K-I-S-S.

© Phillip Bailey - All Rights Reserved

Element #3: Reproducibility

Here you describe as accurately as possible how often the bug occurs by following the exact Steps to Reproduce. **There are 3 standard choices:**

- **Always**
- **Intermittent**
- **Seen Once**

Those should be the only options that you choose from. Let's define each option one at a time:

- **Always:**
  - By following the Steps to Reproduce, the bug will happen every single time *without fail*
- **Intermittent:**
  - This means that the bug does not occur 100% of the time when following the steps, but has occurred more than once
  - When using Intermittent as a Reproducibility, it is imperative that you state how many times a bug has occurred out of how many attempts to reproduce – "X out of Y"
- **Seen Once:**
  - The bug has only occurred on a single occasion and has not yet been reproduced. This could be because you have not yet figured out exactly how to reproduce the bug despite your best kung fu, but the time has come to enter the bug and so "Seen Once", although not preferred, is an acceptable selection

Element #4: Steps to Reproduce

Just like it sounds, these are the exact necessary steps that anyone should follow to be able to reproduce the bug at will. Begin with any bug-specific setup information like "Using Windows98 SE:", then number your steps.

Each step should contain **a single action** that the user is to perform – **only ONE.** When the next action is called for, that is another, separate step. Write each step necessary up to the point where the bug occurs. **DO NOT write the bug in this sequence, only the steps!**

© Phillip Bailey - All Rights Reserved

Element #5: Result

This is the bug. This is what happened that caused you to write up the issue in the first place. Something like: "Program crashed". Then add the information that is needed to truly understand the bug, because as clear as "Program crashed" is, more information is needed.

- Is the computer still in a usable state?
- Did the computer reboot?
- Did an error screen appear?
- What happened when the program crashed?

So add information like: "Mouse and keyboard no longer responsive. User has to reboot computer."

Element #6: Expected Result

This is what ***should*** have happened. Often this is simply the opposite of what *did* happen, but it needs to be explicitly called out so that there is a clear understanding of whether or not the expected behavior is accurate.

This is the place to quote the specs that your test cases were based on. Something like:
"Program should open the Help menu (per page 6, section 4.1 of tech doc #1.8.6v1)"

Element #7: Severity

First things first: QA is responsible and accountable for the Severity field. Let me state that another way: **QA owns the Severity field**. No one else, only QA. This is because QA is the team that is logging the majority of the issues, the team that is using the product day-in and day-out, the team that has the most in-depth understanding of how *Severe* an issue is for the end-user. QA is the team that knows whether the issue, however simple or complex to fix, has a major adverse impact for the end-user or is only a minor inconvenience. Because of this: **QA owns Severity.**

The Severity of a bug speaks to the impact on the user experience and is NOT the same as Priority. As the consumer's advocate, it is up to you to state how bad this issue would be to the end-user.

© Phillip Bailey - All Rights Reserved

Now, no two companies use the exact same definitions for the Severity of a bug, but this will give you a solid background regarding the use of this field. Use these as guidelines and then adjust the titles once you learn how your company designates its Severities.

Here are **some guidelines to get you started:**

- **Crash or Freeze**
  - This is the most severe. This would be BAD for the end-user and thus is the highest level that can be selected
  - The program crashes or freezes. The computer crashes as a result of the program you are testing. This is the highest severity
- **Major Functionality**
  - A very important piece of functionality does not perform at all as it should. Either it doesn't do anything, or it does the wrong thing. You are able to continue working in the program – it has not crashed or frozen – but it is not doing the right thing and it's not a small thing
- **Major Cosmetic**
  - A glaring visual issue that does not affect the functionality of the program would be considered Major Cosmetic. A single pixel width off is not major. An outdated copyright, however, could definitely be considered major, especially by a legal department
- **Minor Functionality**
  - Much like Major Functionality, this is a piece of the program that is not behaving as expected. However, in this case there may be an easy workaround available to the user that is obvious or is something like a help page opening but the text is not positioned exactly where it should be on the page for a user to see it best
- **Minor Cosmetic**
  - Here is where a single pixel-width issue would fall in terms of severity. If the colors don't match exactly or there is some other art issue that is not obvious or screamingly ugly
- **Suggestion / Feature Request**
  - This is something that you think should be added to make for a better user experience. Some companies like to categorize these not as "bugs" in their bugbases, but as "features". This is fine, as the same rules still apply
  - There are **three things to keep in mind when entering a Suggestion** bug (also known as a Feature Request)
    1. Write it up exactly as you would any other bug

© Phillip Bailey - All Rights Reserved

2. Be as clear as possible about what is, or is not, happening (the Result) from a **user's perspective** so that when reviewed, people will understand what it is you think is missing or should be changed
3. ALWAYS state a very clear Expected Result. Here you will describe WHAT you think should happen and WHY. If you can do this, then your suggestion will have a much better chance of being considered. Your suggestion will not be taken seriously if you cannot state the reason(s) that would make your suggested change worth undertaking. Your suggestion may have been considered in the design meetings and not included in this release due to the risk or complexity it would entail. Or it may not have been considered at all. If you can state clearly why it would enhance the end-user's experience, then someone can make an informed decision as to whether they think this change is worth the effort

Element #8: Priority

The Priority field is best owned by the Producer or Project Manger. This determines how important the bug is to fix. The most common designation for this field is Priority "X" where X= a number. Priority 1 (P1), Priority 2 (P2), etc.

QA should not set the Priority of a bug – ever! That being said, each company does things a little bit differently and you may work for a company that wants QA to set the Priority of the bugs that it enters. This is a mistake, but plenty of companies make it. **If you are called upon to set the Priority, get CLEAR definitions of each Priority** from your QA Lead or Manager before you begin using this field – and then be sure to follow those definitions as best you can.

Once entered and reviewed, each bug is given a Priority designation. This represents how important the bug is to in the current release. Factors taken into consideration when setting Priority are: complexity of the fix, time to implement the fix, risk level of implementing the fix, time needed to test the fix, etc.

The Producer or Project Manager or Engineering Lead will set the Priority of bugs as they are reviewed. This helps clarify to the whole team what work is to be done next.

© Phillip Bailey - All Rights Reserved

Element #9: Status

The Status field shows the current state of a bug (see Chapter #10: The Life of a Bug for more detail). A bug begins in an Active state ("Active" or "Open") and is assigned to the appropriate party for Prioritization.

The bug remains in this Active state until it is fixed. Once Engineering implements a fix for the bug, the status is changed to a Resolved state ("Fixed" or "Resolved – Fixed" or "Fixed\Not Verified") and is assigned to QA to test.

After testing the fix, if QA determines that the bug has not actually been fixed, they change the bug back to the Active state and assign it back to Engineering to attempt another fix.

Once a bug has been verified as Fixed, QA changes the Status to a Verified State and Closes the bug. Once Closed, if a bug is found to have been re-introduced, it is Re-Opened with the Status being changed to an Active state or to "Re-Opened" (depending on the company) and assigned to Engineering.

© Phillip Bailey - All Rights Reserved

States

- **Active Status:** When you first enter a bug, it is an "Active" state. It remains in this state until its Status is changed to "Resolved" (Engineering has implemented code that is believed to fix the bug) or "Deferred" (when the fix for the bug has been scheduled for a later release).
    - Common terms for Active Status bugs:
        - Open
        - Active
        - Unresolved
- **Resolved Status:** When Engineering implements a fix, they change the bug's Status to Resolved and assign it to QA for verification of the fix.
    - Common terms for Resolved Status bugs:
        - Resolved
        - Resolved – Not Verified
        - Resolved – Implemented
        - Resolved – Will Not Fix
        - Fixed
        - Fixed – Not Verified
- **Verified Status:** Once QA has tested the implemented fix and is certain that the bug is no longer occurring, QA changes the bug's Status to Verified and then they can Close the bug.
    - Common terms for Verified Status bugs:
        - Verified
        - Fixed – Verified
        - Implemented – Verified
- **Closed Status:** Once a bug has been verified as fixed and no longer occurring, the bug's Status should be changed to Closed by QA. The bug will no longer appear when searching for "Open" or "Active" issues.
    - Common terms for Closed Status bugs:
        - Closed
- **Deferred Status:** If the decision is made to **not** address a bug in the current release, the Producer or Project Manager will change the Status to Deferred. At this point, the bug will not be considered a part of the current release. These bugs should be addressed during the planning phase of the **next** release to determine whether they will be fixed, deferred to a later release, or not fixed at all.
    - Common terms for Deferred Status bugs:
        - Deferred
        - Known Shippable
        - TBD (this stands for "To Be Determined")

© Phillip Bailey - All Rights Reserved

- **Re-Activated Status:** When testing, if QA encounters a bug that was previously Closed, they change the bug's Status to a Re-Activated state.
    - Common terms for Re-Activated Status bugs:
        - Re-Activated
        - Re-Opened
        - Active
        - Reoccurs
- **By Design Status:** If it is determined that a logged bug is not truly a bug at all but is a feature that is behaving as expected, the bug's Status is changed to a By Design state by the Producer or Project Manager. This occurs because either the documentation was incomplete or missing, the behavior was not documented in the specs, the tests built based on the provided documentation are incorrect or inaccurate, or because the tester did not clearly understand the desired behavior.
    - Common terms for By Design Status bugs:
        - By Design
        - Not a Bug
        - As Designed

Element #10: Assignee

The Assignee field is used to ensure that every bug is owned by a project team member. This helps build in accountability and makes it possible to accurately track the progress of every bug.

Upon submission, each bug should be assigned to the appropriate party. This usually begins as an assignment to the Producer or Project Manager. On some projects, the default assignment is given to the lead developer. Either way, once submitted each bug gets assigned to the person who can do something with it.

When QA enters a bug, it is assigned to the Producer or Lead Engineer. They, in turn, set the Priority and Assign it to the Engineer that will fix it.

Once Engineering fixes the bug, they Assign it to QA. QA tests the bug and then Assigns it back to Engineering if it needs further work. In this way, the bug is always Assigned to the appropriate party who is in line to perform their job on the bug.

© Phillip Bailey - All Rights Reserved

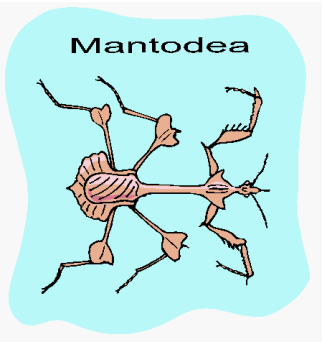
Two very important actions you must remember **regarding the Assignee field:**

- **Commenting the Bug:** Every time you Assign a bug to someone, **be sure to make a comment in the bug stating why you are changing the Assignment.** This is so that the person you Assign the bug to will have some idea of why you did so and what action they should take. When you don't clearly comment a bug, you create confusion. How would like to have a bug Assigned to you and have no idea why? Take the time to state in the bug why you are Assigning it.
- **My Bugs Filter: Create a search in your bugbase that will show you** at a glance **all of the bugs that are Assigned to you.** Make sure you remain current with this list and do your best to keep it as short as possible. Once a bug is Assigned to you, take the action you need to and Assign it to the next person or Verify and Close it. If you keep the bugs Assigned to you to a minimum, you will be seen as an efficient and Quality Professional.

Learn these Elements. Understand each Element individually and what part it plays in a bug. Then using a wider view, gain an appreciation for how they fit together. If you can do this, soon you will be able to write great bugs every time!

The 10 Critical Elements of a Bug

1. **Brief Description**
2. **Expanded Description**
3. **Steps to Reproduce**
4. **Result**
5. **Expected Result**
6. **Reproducibility**
7. **Severity**
8. **Priority**
9. **Status**
10. **Assignee**



© Phillip Bailey - All Rights Reserved

Chapter #8: The 5 Keys to a Bulletproof Bug

The winds and waves are always on the side of the ablest navigators.
~ Edward Gibbson

Quality is not an act. It is a habit.
~ Aristotle

In the section The 10 Critical Elements of a Bug, I introduced you to an overview of the anatomy of a bug. You should now understand what general elements you must include when writing a high-quality bug.

We are now going to take this to the next level and get down and dirty with the detailed information you need to write great bugs. Below I describe, in as much detail as you will need, each of the 5 Keys so that you can master the art of writing high-quality, value-added, **Bulletproof Bugs**.

The 5 Keys to a Bulletproof Bug

1. **Descriptions (brief and expanded)**
2. **Reproducibility (X-out-of-Y)**
3. **Steps**
4. **Result**
5. **Expected Result**

Learn the importance of each Key and strive to understand how they work together to complement each other. Once you have an in-depth understanding of each Key and the interrelation of all the Keys, you will be on your way to knowing how to write a bug that your team will understand and respect.

Strive to practice as perfectly as you can. Soon these 5 Keys will be second nature to you and every bug you write will be unquestioned. You will be able to write any bug, at any time, about any issue that everyone can understand and trust.

As you read through this chapter, use the attached sample bugs as reference. This will help enhance your understanding of what to write and how to write it as well as increasing your comprehension speed.

© Phillip Bailey - All Rights Reserved

Key #1: Descriptions

Brief Description

As noted earlier, the Brief Description is a one line summary of the bug you are entering. Write it as clearly, succinctly, and concisely as possible. The idea here is to encapsulate the issue as best you can in the fewest possible words.

Remember, this is often the only information displayed when searching a bugbase until the user opens each bug one at a time. So make sure that anyone scanning a list of 500 bugs can easily differentiate the bug you are writing from every other one line descriptor.

If you find it too difficult to write this Brief Description first, use an old QA trick and write the Brief Description last. This may seem a little counterintuitive, but once you have finished writing the rest of your bug, it will be much easier to break it down into a simple description.

Practice improving the brevity and clarity of your Brief Description until anyone can easily understand the bug from just this one sentence. You will save your project team time and your company money if you are able to explain your bug in just one clear sentence.

People will notice that your bugs are easy to understand and trust your work. This is because you will be saving everyone time in the many meetings they must attend to sort through all of the bugs. The less time they need to take understanding your bugs, the better. Be a professional.

Expanded Description

The Expanded Description is your chance to explain the bug you are writing in greater detail. It may be the exact same as the Brief Description as long as it is descriptive enough – OR – You can copy the Brief Description and use it as the beginning of your Expanded Description.

The Expanded Description should go at the top of the text field in which you are writing the rest of the bug. It should be the first piece of information a reader sees when they open the bug and begin reading it.

© Phillip Bailey - All Rights Reserved

Use as much verbiage as you need to clearly state the issue. Since the reader has already had a chance to scan the Brief Summaries of the bugs, this is the place you can explain the issue in more depth.

You may find that you will need to edit the Expanded Description before officially submitting the bug, but it is good to write out what the issue is to help keep you focused on what you are writing about.

Write clearly what the bug is. Just write it. Get it all out. It is easier to edit a bug to have less information than to attempt to write the perfect explanation in the first place. So write out all that you need to. Once you are ready to submit the bug, review your Expanded Description and using a "less is more" mentality, delete or edit your text so that it is as clear and succinct as possible.

© Phillip Bailey - All Rights Reserved

Key #2: Reproducibility

After you have written your Expanded Description, you will want to define your bug's Reproducibility. You should write this directly under the Expanded Description. This way the reader will immediately have an understanding of how often the bug you have just described occurs. This is the order that a reader can most easily understand the Severity of your bug and the Priority that should be assigned to it.

Reproducibility has *only* three options:

- Always
- Intermittent
- Seen Once – OR – One Time

Let's look at when you would use each selection.

Always

The bug you are writing will happen 100% of the time, **without fail**, when the Steps that you write are followed. In this instance "Always" really means always.

When you write up a bug and state that it has a Reproducibility of Always, be certain that this is true. Do enough testing to determine that the bug will happen every time. Not most of the time, ALL of the time when the reader follows the Setup and Steps to Reproduce that you detail.

It is possible to have a bug that will only occur under specific circumstances or only when certain conditions are met. If you include these details in your Setup and Steps, then you still will have an Always bug.

Conditions that may affect, create, or contribute to a situation-specific bug include (but are not limited to):

- Operating System
- Browser Version
- Media Player Version
- Internet Connectivity Type
- CPU Speed
- Available RAM
- DirectX Version
- Any combination of the above

© Phillip Bailey - All Rights Reserved

When this is the case, you must include these situation-specific factors in your bug. If you detail these factors correctly, your Reproducibility should be solid. It is ok to write that a bug Always occurs when using Windows 98SE and then following the Steps as written.

Intermittent

In your journeys you will encounter bugs that occur sometimes, and only sometimes. This can be frustrating, but it does happen and you will have to accept it as a reality of testing.

However, make sure in your testing that you don't fall into the trap of accepting a bug as Intermittent too quickly. Many novice testers too easily accept that the bug they have just encountered is Intermittent. Often they are wrong. They just haven't spent enough time and effort to determine the Reproducibility with surety.

When it comes to light, as it always does, that the bug is not Intermittent but occurs more frequently (see: Always) and the tester simply didn't complete the work to do their job then the perception of QA suffers. The team loses trust in the tester's skill and attentiveness. As I have stressed throughout this book, this is the opposite outcome you are working to achieve. So, be thorough. If a bug is truly Intermittent, so be it…just make sure that you are certain it is.

To reiterate, **when you encounter a bug that appears to be Intermittent, you must remember two things:**

First: Test and re-test until you are certain that you are unable to find the exact circumstances and Steps that will give you a Reproducibility of Always. Once you have done that and are sure your bug is truly Intermittent, then move on to the second thing.

Second: Whenever you write up a bug with a Reproducibility of Intermittent, **you must detail the Intermittency.** What I mean by this is that you must state that the bug occurs "X-out-of-Y" times. You must clearly state the bug occurred <how many times> out of <how many attempts>. This way everyone can get an informed understanding of the true impact of the bug you have written.

If requested or if you like, you can state this Intermittency as a percentage *also*, but you must state "X out of Y". This clearly quantifies the current impact of the bug.

© Phillip Bailey - All Rights Reserved

Only stating that the bug occurs 20% of the time is *not* an impactful statement. This could mean:

- 1 out of 5
- 2 out of 10
- 20 out of 100

A sample size of 100 is a far more accurate count of the bug's impact than a sample size of 5, therefore "20 out of 100" is a more impactful statement than "1 out of 5" – **so be sure to always include "X out of Y" ANY time you enter an Intermittent bug.**

Seen Once – OR – One Time

It will happen, but I hope for you it will happen only rarely. You will encounter bugs that, despite your best efforts, you will have to enter as Seen Once. What this means is that you were able to produce a bug once, but only once. You have tried and tried, but are unable to establish a circumstance and Steps that will entice the bug to occur a second time.

Although sad, this happens to everyone. Everyone that stays in testing long enough, that is. When this happens to you, be a professional and do the right thing

"Pretend it never happened?!?"

As nice as that would be, no…don't do that. You can do the next best thing.

As a Quality Assurance Professional, you must perform due diligence and ensure that you are unable to find a way to reproduce the bug. If you can do this, then you can move on and enter the bug.

When entering a bug with a Reproducibility of Seen Once, be sure to state how many attempts you have made to reproduce the bug. This will help others understand whether or not attempts have yet been made to reproduce the bug – or if you had to enter it because it was closing time and you had to get the bug into the bugbase.

© Phillip Bailey - All Rights Reserved

Write it up as you would any other bug, but make sure that you include all the information you have about what the circumstances were when the bug occurred.
Include the details of:

- What actions led up to the bug occurring
- What environment were you working in when the bug occurred
- What testing did you perform when attempting to reproduce the bug

That last piece of information is key: What have you tried that did NOT result in the bug occurring again? With this information, it will be easier to narrow down the variables going forward if the bug is ever seen again. If nothing else, it will give the only other data you have about the issue – this can be key if an Engineer is to have any chance at tracking down the issue.

Whenever possible, this Reproducibility should be upgraded to a more precise entry as soon as possible. However, there are times when you simply will not be able to figure out what particular circumstances caused an issue to occur – these should be few and far between, but it does happen. So after giving it your best effort, update the bug with as much information as possible and move on.

© Phillip Bailey - All Rights Reserved

Key #3: Steps to Reproduce

Once you have established a definite Reproducibility, it's time to move onto spelling out to the reader the exact Steps they will need to follow to be able to reliably replicate your bug. This Key begins with the Setup and then lists the necessary Steps to Reproduce.

The need for a Setup depends on the bug. You may or may not need to include a Setup, but just in case your bug calls for it, we will cover it here.

Your Setup should tell the reader about any conditions that must be present, any prerequisites that must be met, and/or any environmental factors that are necessary to be able to reproduce your bug. This is separate from the Steps. This is how to Set Up so that once they follow the Steps they will be able to experience the joy that is your bug.

In your Setup, **explain where the user needs to start.**

For instance:

- In Windows 98SE:
- In MacOS 10.3.9:
- Using IE 5.5:
- Running AOL 10 on Win2K with Screensaver set to 1 minute:
- After spinning around in your chair three times:

Ok, that last one you will probably never use, but you get the idea. Your goal when writing the Setup is to help the reader get their testing environment ready so that all they must do after the Setup is follow your Steps.

I know this sounds simplistic, and it is, but can you see what would happen if you didn't include a Setup when one was called for? If a bug only occurs in Windows XP and the reader is attempting to reproduce your bug on a Mac – because you didn't state that the bug only occurs in Windows XP – what will they think when the follow your Steps but your bug doesn't occur?

The first reaction is always to question the validity of the bug. As a result of that reaction, the reader will also question the person who wrote the bug. This again is not the result you are looking to achieve on your way to becoming an elite Quality Assurance Professional.

© Phillip Bailey - All Rights Reserved

When a developer or producer spends an hour of their time futilely trying to reproduce a bug that was written inaccurately, they will feel that their time was wasted. And since it occurred while they were trying to understand a bug *you* wrote, they will blame you for wasting their time. Let's avoid this.

In order to avoid this unpleasant scenario, just include the necessary Setup when it is appropriate. When you do this, it is often not noticed initially. No one says, "Good Setup!"

But you know that you have provided the information they need to reproduce the bug…even if they don't see it. And sometimes the reader does miss it. It's right there in front of them, but since *they* are not as detail oriented as you, sometimes they miss the little things.

It can be entertaining to politely set a developer straight when they claim that they are unable to reproduce your bug. They have tried but can't replicate it and ask what you forgot to include. Then you ask them if they are testing it using XP – as stated in your bug. The inevitable pause will occur…time passes slowly…then they tell you…their machine is running Win2k, not XP…and they ask, "Why doesn't the bug happen the way you wrote it? Do I have to use XP?"

Now it is up to you to **not** laugh, to **not** say I-told-you-so. Now you politely explain that yes, indeed, the bug only occurs in XP.

They will either go away quietly and eventually notice that you had included that information in the first place, and they may apologize for the misunderstanding (and ask to borrow an XP machine). Or they might ask you right then why you didn't include it in the bug in the first place. Again, this is your chance to **not** laugh.

You now should ***very*** politely, really, as politely as possible, point out *where* in the bug you included this information. They may now apologize, or look sheepish, or just say, "Oh" (which means "oops"). Either way, you did you job like a professional and when challenged you were able to point the questioner to the exact place they needed to find the information they were missing.

As I said before, you may not need to include a Setup. If your bug occurs on every operating system, every time – then you probably won't need to include a Setup. But if you are in doubt, go ahead and include it.

© Phillip Bailey - All Rights Reserved

So, you should now have a clear idea of why to write a Setup, when you would need it, and how to write it up. Let's move to the next piece of this Key; the Steps.

The Steps to Reproduce. This is very simple, but many people still do this incorrectly. Here is the best way to write them:

1. Always number them (and start with the number "1")
2. Make sure each step is a single action
3. As soon as the next action is needed, make another step
4. Write all steps needed until the bug occurs
5. DO NOT INCLUDE THE RESULT (THE BUG) IN THE STEPS

Your **Steps to Reproduce should be simple**. That is imperative. The more complicated you make them, the more difficult your bug will be to understand. If your bug is difficult to understand, your work will look less professional – so make it simple. **Remember KISS.**

Take a look at the sample bugs included to get an idea of the proper format and professional verbiage that comprise quality Steps. You should make the bug sound like you wrote it, but be sure you use a vocabulary that is objective. When I write bugs, I attempt to keep them as clinical as possible (as you may notice from my sample bugs).

The Setup and Steps to Reproduce should look like this:
On a Mac running OS 10.3.7:

1. Launch <program>
2. Navigate to <location>
3. Select <button>
4. When <window> appears, select <link>

And that's it. Just what they should do and where they should do it. As soon as all of the action Steps are listed, right up to the moment that the bug occurs, move on to the Result.

This is very important: The Steps just comprise the path that lead to the event that triggered you to write up the bug in the first place. Do not include this event in the Steps – this event is the Result.

Does that make sense?

© Phillip Bailey - All Rights Reserved

Here is what your bug looks like so far:

- You have summarized it in a single sentence for easy scanning and understanding
- You have explained in more explicit language the full effect of the issue
- You have stated a clear frequency of occurrence
- You have listed any environmental variables that must be in place to make the bug occur
- You have walked the reader through the exact path they must follow

And now for the payoff: The Result…

Key #4: Result

The Result should be a clear, simple explanation of exactly what happened and, therefore, what is now affected. State the bug. Write the exact event or behavior that caused you to write the bug.

You need to **state** this **Result** as well as **what is affected** because of the Result. If the program crashed, state that. That's a good start. But make sure you include the "then what does that mean?" If the program crashed, did the user lose any information they were working with? Did the crash cause the computer become non-responsive?

You must include the details of what the effect of the bug is. Here you are **quantifying the impact** of the issue you are reporting. **This adds value to your bug**. This is where you will separate yourself from the average tester.

An average tester will state that the program crashed. That's all. That's it. They will write no more.

You, on the other hand will use this opportunity to show why you are elite, not average. You will write that when the program crashed, the computer had also frozen. The keyboard and mouse were unresponsive and the computer had to be manually restarted.

Of course, only write all of that if it is actually true. I'm sure that's how you would do it.

The Result is also the place to reiterate two things (if they apply):

1. **The Intermittency of the bug** or if it has only been Seen Once. If this bug does not occur 100% of the time, then in the Result you should state this. It is a prime location for you to remind the reader how often the bug occurs so that if they don't see the outcome they expected, they will know why
2. **The environment in which the bug occurs.** If the bug only happens in Win98 SE, this is where you should remind the reader. Again, this is so that if they missed it the first time or forgot, it will be available for them here. Also, some people scan bugs and read the Result first – when they do this they will miss all of the great information you wrote earlier, so here is your chance to put all that information in one place

© Phillip Bailey - All Rights Reserved

Here is an example of one poorly written Result and one well written Result:

Bad Result:
Program crashes intermittently

Good Result:
Program crashed. Computer froze. Mouse and keyboard are not responsive. Hard reboot necessary and program must be re-launched.
**Note: Bug occurs in Win98SE only
Bug has occurred in 6 out of 10 attempts

Show your worth and write a great Result. Separate yourself from the crowd. Make the rest of your project team happy to deal with your bugs and raise the level of trust they have in you and your work by writing a comprehensive Result.

Once you've written a great Result, move on to the Expected Result.

© Phillip Bailey - All Rights Reserved

Key #5: Expected Result

The Expected Result is where you state, without ambiguity, what you expected to happen. Very often this is just a way to clearly state the opposite outcome than what you witnessed (the Result). Whether the Expected Result is simple or complex, you must articulate it with as much clarity as you have the rest of the bug.

Your Expected Result needs to be as clear as the rest of the bug, not just "That shouldn't have happened" or "The program should do what I want it to do". Your Expected Result needs to state specifically what should have happened.

Remember that the test you were running had a pre-defined outcome – this is the Expected Result. This outcome was expected for a reason; either because the link you clicked read "Help" and so you expected the Help Menu to open, or you changed the armor on your character and expected to see a change in your defense rating. Whatever you did, there was an expectation. This is your Expected Result.

If there is a document that you can cite to point the reader to the reason for your expectation, **cite it**. Tell them which document, where in the document, and quote it if at all possible.

When you are writing your bugs, **be sure to give as much attention to** this Key, **the Expected Result**, as you give to the other Keys. **If you do not include an Expected Result, you don't have a bug.**

This seems simple, but I have encountered many testers who do not spend the necessary effort when writing this part of their bug. They write "verify above result" or "this behavior seems wrong". That is not good enough. In this Key, you must state what *should* have happened and, if at all possible, why it should have happened.

Although I will delve more deeply into this Key in the section 5 Most Common Bug Writing Errors, here is a glimpse into one of the reasons that the Expected Result is so important: Re-testing the bug.

When a bug has been fixed, QA must perform a type of testing known as Regression Testing. In this scenario (commonly known as "Regressing a bug"), you will be testing to see if the applied fix actually worked.

© Phillip Bailey - All Rights Reserved

Now think about that. If you are testing the fix, how do you know if the fix was successful? You would know because the current behavior would match your test's expected outcome – it would match your Expected Result.

If you have not identified and described a clear Expected Result in your bug, how can you validate the fix? Here a tip: YOU CAN'T! So, let me repeat: **If you don't write an Expected Result, you don't have a bug**. Does that make sense?

I am guessing that you get the point. You are here to add value and quality to the project – this is just another detail at which you have a chance to excel.

Here is an example of poorly written Expected Results and well written Expected Results:

Bad Expected Results:
Program should keep running
– OR –
Program should not crash

Good Expected Result:
Help Menu should open and the program should continue running
– OR –
FAQ page should open (per requirement 4.1.2 in Technical Specification Doc 6.1a, page 32, paragraph 4)

Remember to clearly state what should have happened so that there is no question as to whether or not the behavior you encountered was a bug. A good Expected Result will leave the reader knowing that you have done your job with proper attention to detail.

And there you have it! Simple really. The 5 Keys that you need in order to write a Bulletproof bug!

I know that it seems like a lot of detail to keep track of at first, and might even seem a little intimidating, but don't fret. Just review the sample bugs included in this book to get an idea of what a Bulletproof Bug looks like.

© Phillip Bailey - All Rights Reserved

Then when you need to write a bug, just refer to the sample bugs to jog you memory. Use the included Bug Template. When you write your bugs, just make them match the samples. You will get the hang of this in no time, it just takes practice. Strive to practice writing your bugs as perfectly as possible and before you know it, you will be writing Bulletproof Bugs all the time, every time!

The 5 Keys to a Bulletproof Bug

1. **Descriptions (brief and expanded)**
2. **Reproducibility (X-out-of-Y)**
3. **Steps**
4. **Result**
5. **Expected Result**

You will have mastered the fine art of bug writing and can help others improve their skills at it too! You have the knowledge, now put it to work!

© Phillip Bailey - All Rights Reserved

Chapter #9: The 5 Most Common Bug Writing Errors

You can be discouraged by failure, or you can learn from it. So go ahead and make mistakes, make all you can. Because, remember that's where you'll find success - on the far side of failure.
~ Thomas J Watson Sr.

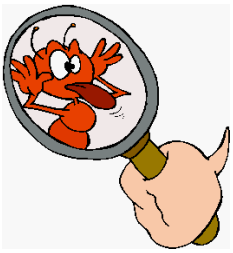
You now know The 10 Critical Elements of a Bug and The 5 Keys to a Bulletproof Bug. You are on your way!

In this chapter I will detail the most frequent mistakes that testers make when writing bugs and how to avoid them. This will help you NOT make the mistakes that so many before you have. This puts you WAY ahead of the average tester. Learn what these errors are and how to avoid them and you will save yourself a great deal of time and frustration in your testing career!

There are many errors that Quality Assurance testers make daily when writing bugs. Keep these points in mind when you enter your bugs and you will sidestep the most common pitfalls that many testers make.

The 5 Most Common Bug Writing Errors

1. **Not Enough Information**
2. **Unclear Reproducibility**
3. **Omitting the Expected Result**
4. **Selecting the Wrong Severity**
5. **Assigning Bug to the Wrong Person**



Now let's break each of these Errors down so that you can understand and avoid them.

© Phillip Bailey - All Rights Reserved

Error #1: Not Enough Information

A bug is useless if it does not contain enough information. It is not a rare occurrence for a bug to be missing the necessary Setup information or to not list all of the Steps. As I said before, if a bug does not include the necessary information for Setup and Steps to Reproduce, there is no way for anyone to ever be able to replicate the bug with accuracy.

Remember that **the goal is for anyone to be able to reproduce the bug** with only the information contained within it. Any other scenario is going to make someone else do more work to be able to understand what you have done. Don't be the person who creates more work for others. Be someone who creates solutions, not obstacles.

As noted before, simply stating in the Result that the program crashed is not enough information. You must add the important details of the result(s) of that crash. Add what the effect of the bug was: "Result 'X' means that the user cannot do 'Y'" – that sort of thing.

After you write a bug, read through it. Read it as though you know nothing about the bug. Did you include all of the necessary information? When you read the bug, does it make sense?

© Phillip Bailey - All Rights Reserved

Error #2: Unclear Reproducibility

If you don't state the Reproducibility of the bug clearly, then the team will not be able to establish an accurate Priority for it. You must state very clearly whether the bug happens Always, is Intermittent, or has only been Seen Once.

Attempt to reproduce the bug enough times to feel confident in the Reproducibility that you state. If it occurs every time, great! **Always**!

If it only occurs sometimes or most of the time, you must state "how-many-times-it-occurred" out of "how many times you attempted to reproduce it"; the "X out of Y".

- 2 out of 10 = 20%
- 20 out of 100 = 20%

Both of these examples show the same percentage. Which numbers would you have more confidence in? I hope you chose the second one. A sample size of 100 should give you far more confidence than a sample size of 10.

If the bug has only occurred once and you have tried to reproduce it several times, state that. State how many times you attempted to reproduce the bug and were unable to. Then keep this bug in mind so that if you see it again, you can update the information in the bug with your latest results.

As **a bonus tip:** Keep a ready "Most Wanted" list at your workstation of bugs that you have only Seen Once. This way, if you ever encounter them again, you won't have to search laboriously through the bugbase to find them. You can just look at your list, navigate to your bug, and update it with your new information. This will save you a great deal of time and effort as you progress through your projects.

Saving you time from endless searching will mean that you are able to focus on the more important testing and reporting issues. This will make you appear even more efficient and productive! Always a good thing for the aspiring Quality Assurance Professional.

© Phillip Bailey - All Rights Reserved

Error #3: Omitting the Expected Result

If you don't have an Expected Result, you don't have a bug! Period (or exclamation point). There is no way you could have found a bug if you didn't have a clear expectation of what should have happened. There is simply no way.

Maybe the program is supposed to crash when you pressed <Enter>. Without a quantified, desired outcome, how do you know if the program you're testing is not functioning correctly?

Hell, you can't even test properly without a clear idea of what is supposed to happen when you run your tests. How can you?

I can make ***any*** program crash (trust me, I've done it) – does that mean it shouldn't? Under what circumstances should a program remain stable and when should it crash? If you can't tell me what is supposed to happen when you press the big, red button, then don't tell me you have a bug if something or nothing happens!

In order to execute your testing, **every test needs to state the desired outcome.** *Then* if the functionality of the program differs from what is stated, you can log a bug. And when you do, please state what was supposed to happen.

This will save you time and frustration when you have to regress the bug and it will help the developer know what you thought should happen. It is possible that what you expected wasn't accurate. If that is the case, and you have stated your expectation clearly, then the documentation, test case, and program can all be updated so that everyone has a clear vision of what is, truly, supposed to happen.

Regardless of where the error is, whether in the code or in the documentation, you must always include a clear Expected Result in every bug you write. Remember: if you don't include an Expected Result, then you haven't written a bug.

© Phillip Bailey - All Rights Reserved

Error #4: Selecting the Wrong Severity

Even with your most diligent effort, sometimes you may choose a Severity that is not the best choice. You may select a Severity that someone questions. When this is the case, be sure that you have a rational argument for why you selected that Severity. You may become convinced that your choice was not the best, but you should be able to defend it if challenged.

If this happens, learn from the feedback you get so that you can choose more wisely in the future. This can be useful a learning process if you take it in the right way. Do your best, but if you choose inaccurately, learn from it and move on.

However, there is also a scenario that can catch people off guard because of the way in which this Error is made. Some bugbases remember the settings of the previous bug you entered. When this is the case, each time you enter a new bug, it retains (by default) your selections for the last bug.

This means that if you select the highest Severity for bug#1, then when you enter bug#2 it will begin with this same, highest Severity unless you change it. This means that if you don't pay attention, you can log bugs all day that have the highest Severity possible – whether you are logging a crash bug or a typo.

So, do yourself a favor and **be sure you review your bug** and all of its fields **before you submit it.** You don't want some developer or manager landing on your desk demanding an explanation of why the typo in the 3rd line of the 2nd paragraph of the ReadMe was logged as a crash bug – trust me, this is not fun.

I have learned the hard way. You don't have to.

If the bugbase you are using *does* reset all fields to an unselected, default state every time that you enter a new bug, this will not be a major concern of yours. Just keep in mind that every bugbase behaves a little differently. Know your bugbase, and keep yourself safe.

© Phillip Bailey - All Rights Reserved

Error #5: Assigning Bug to the Wrong Person

Another of the fields you may be required to set when entering a new bug is the Assignee field. This determines who has responsibility for the bug once you have entered it.

Some companies set up their projects so that all new bugs go to one person in particular who then reviews and assigns them to the appropriate person. If this is the case, just make sure that the default Assignee is correct when you enter a bug.

If you have the responsibility of assigning your newly entered bugs to one of a number of people, **make sure you do 2 things:**

- Have a list of which bugs get assigned to whom
- Assign your bugs to the appropriate person

Again, with any bugbase that remembers your most recent bug entry, be careful that you are not assigning every bug to the same person if they should be assigned to different people. People find it *very* frustrating to have each and every bug assigned to them if they shouldn't be. This forces them to figure out who the bug should be assigned to and interrupts their work. Assign your bugs to the correct person and be viewed as a Quality Professional.

You have been given a walk-through that will allow you to enter any bug with clarity and confidence. Now you are also armed with the information you need to avoid the most common pitfalls that hamstring most testers.

The 5 Most Common Bug Writing Errors

1. **Not Enough Information**
2. **Unclear Reproducibility**
3. **Omitting the Expected Result**
4. **Selecting the Wrong Severity**
5. **Assigning Bug to the Wrong Person**

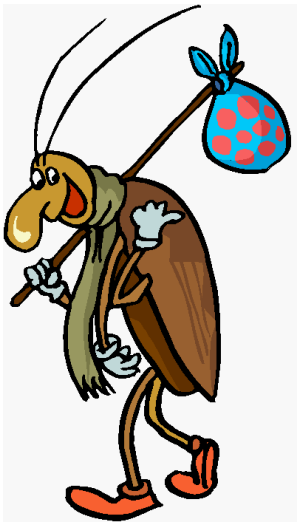
You know what hazards to avoid and how to avoid them. I have given you the tools to succeed in your bug writing. Now it is up to you. Go forth boldly and dare to excel!

© Phillip Bailey - All Rights Reserved

Chapter #10: The Life of a Bug

Any sufficiently advanced bug is indistinguishable from a feature.
~ Rich Kulawiec

A key to understanding Quality Assurance and its place in the Software Development Lifecycle is knowing what happens once a bug has been entered. Below I will guide you through all the questions you may have about what happens once a bug is found.



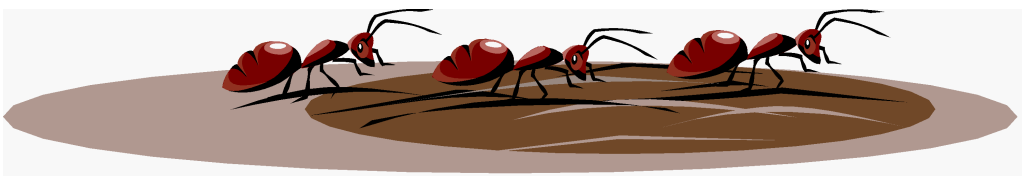
As you go about your day testing, you will encounter behavior in the program that is not as designed. As stated previously, when this happens, it's time for you to enter a bug. Once entered, how does the bug get fixed? Who does it get assigned to and what should that person do with it?

Below is an explanation that follows a bug's life. Follow along on the proceeding flowchart as you read through this section to gain a clear understanding of why and how a bug is assigned to different people and what they do with it.

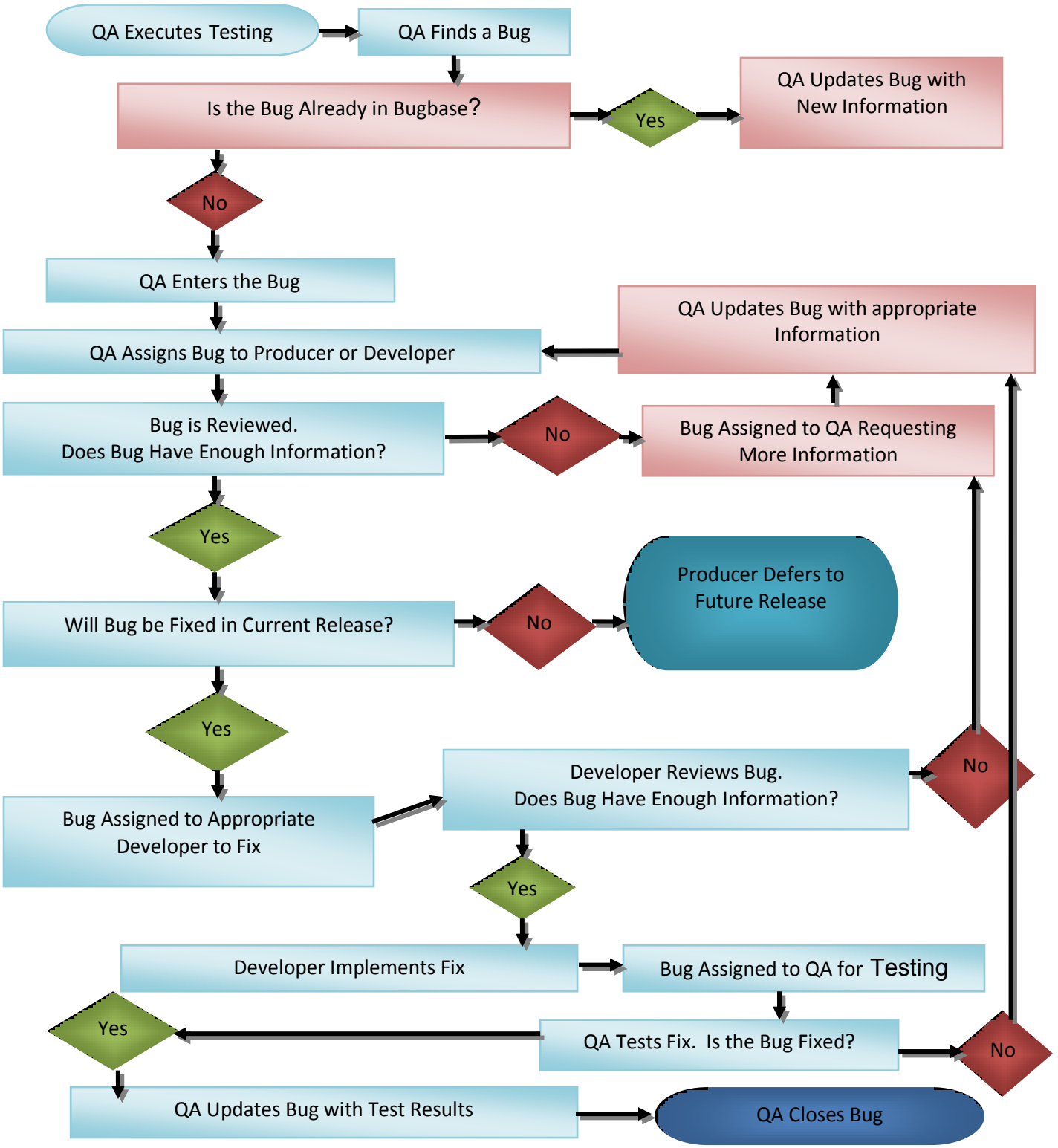
1. **You find a bug:**
    a. Being the responsible and thorough tester that you are, you search the bugbase to see if this bug already exists. **You search through Open and Closed bugs.**
        i. **If the bug already exits and is currently in an "Active" state**, add any new information that you have uncovered (if applicable) and move on to your next test
        ii. **If the bug already exists but is in a "Closed" state**, Re-Activate the bug and assign it to the appropriate person. This is often the person who fixed it in the first place or the Producer/Project Manager
        iii. **If the bug is *not* yet in the bugbase**, you determine the exact environment and steps necessary to reproduce the bug. Once you have done that, you are ready to enter it

© Phillip Bailey - All Rights Reserved

2. **You enter the bug:**
    a. Employing all of the knowledge you have gained to date, you enter a bulletproof bug. You fill in all of the necessary fields and then assign the bug to the appropriate person.
    b. At this point the bug is in the Active state
3. **The person assigned the bug now reviews the bug, prioritizes it, and either:**
    a. **Fixes it**, changes the Status to a Resolved state and assigns it to QA
    b. **Assigns it** to the appropriate party **to be fixed**
    c. **Assigns it back to QA** because the bug needs more information or some other sort of clarification
4. **Once the bug is assigned to the appropriate person to be fixed:**
    a. A fix is made
    b. The bug's Status is changed to a "Resolved" state and is assigned to QA
5. Having determined that they **need more information** before attempting a fix, **the bug is reassigned back to you** in the Active state, requesting more information
    a. You update the bug with the necessary information and reassign in back to the person that assigned it to you, clearly noting that you have added the information requested. The bug is still in the Active state.
6. **A Resolved bug is assigned to QA:**
    a. You review the fix for the bug and determine 2 things:
        i. The stated fix is clear to you
        ii. The build in which you will be able to test the fix
7. Once you have a build in which you can test the fix, **you test the bug:**
    a. If the testing fails and the bug still occurs, change the Status to an Active state and assign the bug back to the appropriate party
    b. If the testing verifies that the fix worked and that that bug no longer occurs, change the Status to a Verified state and then close the bug



© Phillip Bailey - All Rights Reserved



© Phillip Bailey - All Rights Reserved

There are two ways to write error-free programs; only the third one works.
~ Alan J. Perlis, Software Metrics

To this point you have learned the foundational skills you will need to master in order to excel as a Software Quality Assurance Professional. You have also learned a great deal about the Software Development Lifecycle and QA's place within it.

As you can see, you still need to know more. Well, that's why I'm here. I won't leave you hanging. Now that we've come this far together, let's get you the final pieces of information you will need to do your job most effectively.

Chapter #11: The 5 Steps to Bug Regression

You can't make a program without broken egos.
~ Internet Humor

After the bugs you have written are fixed, they get assigned back to you in a Resolved state and should be ready for testing. This is called "bug regression". You are going to work through the fixes that the developers made to your bugs and see which ones were successful and which weren't – "regressing a bug".

This is where are off the hard work you have expended in testing, correct bug writing, and commenting all pay off. Because you did you work so cleanly, the developer had no problem understanding your bug. This allowed them to quickly deduce a fix for it and now they have implemented that fix. Your job now is to validate the success of that implementation.

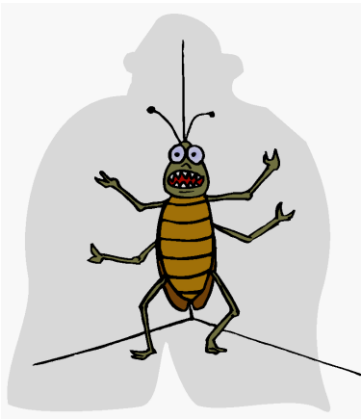
Since you wrote the bug so clearly, you should have no problem following the Setup and Steps to Reproduce. By following the instructions that you wrote, it should be obvious very quickly if the fix was successful.

Undoubtedly, the Reproducibility you wrote is clear and accurate so you know how many attempts you must make to attempt to validate the bug, right? There is no question in your mind as to how many tries will get you the information you need, right?

© Phillip Bailey - All Rights Reserved

Well, just to make sure that you finish the job in stellar fashion, let look at how to perform bug regression properly. You have tested, explained, and commented the bug and now it is time to wrap it all up and close the book on your bug.

In order to put your bug to rest, you need to follow the 5 Steps to Bug Regression.



5 Steps to Bug Regression

1. **Know the Version**
2. **Understand the Fix**
3. **Validate Appropriately**
4. **Comment Thoroughly**
5. **Assign Accurately**

© Phillip Bailey - All Rights Reserved

Step #1: Know the Version

As with much of Software Quality Assurance, this detail is very simple. It is also, unfortunately, very often overlooked by the average tester. Since it is your goal to be a true Quality Assurance Professional, it will be easy for you to catch this detail and make sure you work this step without fail.

In this **first step of Bug Regression**, it is your job to **be sure you know which version of the product contains the fix.** This is to say: you need to know which build to test in order to verify the fix. Then make sure you are using that correct version when you actually regress your bug.

There are two places in your bugbase where the version or build number that contains the fix may be listed. **They are the Fixed In field and/or the Comment field.**

The Fixed In field goes by several names; Fixed For, Fixed In, Resolved In, Version, etc. This field is used for stating which version contains the most current fix for a bug. It may show the version that the bug ***will*** be fixed in or that the bug ***is*** fixed in. This is because a fix may be planned for a version that has not been created yet.

The version or build that a bug is fixed in should also be available in the Comment field. This is where a clear statement should be made when a bug is fixed. It should read something like:
"Fixed in version 1.0.a. Speaker volume is no longer affected when scroll button is selected."

There you have a stated version for the fix and you have an explanation for the bug – perfect. If the Fixed In field matches the version stated in the Comments field, you're in business!

© Phillip Bailey - All Rights Reserved

Another place that the version in which the bug is fixed can be found is in the Release Notes. Release Notes are a document that is delivered with a build. **Release Notes can contain several pieces of information useful for a tester:**

- New features
- Changes to existing features
- Changes to documentation
- A list of bugs that were Deferred
- *A list of bugs that have been fixed in the current version*

That last bullet is what we are looking for here. A comprehensive set of Release Notes will clearly state for you which bugs were fixed in the version of the software that has been delivered to you for testing. With this list of fixed bugs in hand, you can begin regressing your fixed bugs.

This is the **first Step in bug regression: Know the Version that a bug is fixed in.**

The reason that I point out this seemingly obvious Step is that often the average tester overlooks this detail. When they do, they create confusion and show off that they aren't paying attention to what they are doing.

Let me explain what happens:

- A bug is entered – Fine, everyone is happy
- Eventually a developer implements a fix for the bug – Good so far
- The developer changes the bug to a Resolved state then lists and/or comments the build in which the fix will appear. They assign the bug to QA for regression – but QA ***does not*** have this build yet
- The tester, seeing the bug Resolved and assigned to them (because it's in their "My Bugs" bugbase query that they check daily), leaps into action without confirming which build the bug is fixed in – uh, oh…QA still ***does not have*** the build that contains the fix
- The tester regresses the bug in their current build and sees that the bug still occurs – because they are testing a build that ***does not have*** the fix
- The tester comments that the bug still occurs and assigns it back to the developer – even though the bug **should not have been tested yet**
- The developer is confused as to why their fix didn't work

© Phillip Bailey - All Rights Reserved

Because the tester regressed a bug on a build that did not have the fix, the following events have occurred:

- QA wasted time testing something they shouldn't have
- The developer will have to figure out why the fix didn't work – so they waste time attempting to determine whether their fix was correct although it may work just fine
- The project team will notice that QA tested for a fix that they did not have yet – this makes QA look like they are not paying attention, are wasting time, and are creating more work for the rest of the team.

This will only serve to lower the trust the rest of the team has in QA. **This is preventable**. Take the time to be sure you know which build a fix was implemented in ***before*** you begin testing it.

If you are not being provided with the information regarding which build a bug is fixed in, then you will not be able to test accurately. Request the information you need. This will pay off in the long run.

This may make the rest of the project team comment bugs more accurately and keep the bugbase up to date, but that is a good thing. They may not like that it seems like more work, but they will understand in the long run how this will help keep the project on track.

Work like a professional. Attend to the details.

© Phillip Bailey - All Rights Reserved

Step #2: Understand the Fix

Once you know which build contains the fix and you have that build to test on, you can move on to the next step: Understand the Fix. This means that you must **have a clear knowledge of the fix that was implemented.** You don't need to know what each line of code does, but you must know what the expected behavior is now that the fix has been made.

Since you are the one that will see the current behavior, you ought to know what it is supposed to do. Just like any other testing that you execute, make sure you have an accurate expectation of the outcome of your test.

Sometimes this is as simple as: "The help menu should open" – OR – "Program will close but the computer will not crash." If this is the case, it probably synchs up well with the Expected Result you stated originally.

When this happens, it will be easy and straightforward for you to test the bug to verify whether the fix worked or not. For many bugs, this situation is very clear and you will understand the fix without any further input.

If, however, the developer has **not stated the fix clearly** or has not stated the fix at all, you better go **get some clarification**. If the developer's comment is (and I quote): "fixed" then you need to go see your developer about expanding their vocabulary.

As much as it is your job to write up a bug in such a way that anyone can understand it, it is up to the developer to explain what they fixed. If they don't, you have no way of accurately testing it. They will need to be able to explain to you what was fixed if it is not readily obvious in the bug.

So, if the developer that Resolved the bug has not been explicit enough regarding their fix, what should you do? Fortunately, we have covered this previously: **Be nice and go ask**. That's all. Know what you need to ask them and do so politely.

Most developers are very happy to discuss the wonders of their code. All you want is for them to explain what the fix was so that you can test it. They should be happy to detail their fix for you. If you ask nicely, they will welcome your questions in the future.

© Phillip Bailey - All Rights Reserved

Once they have explained the fix, get them to tell you what else their fix may have affected. Did their fix put any other functionality at risk, in their opinion? They are most often very forthcoming about whether they think it did or not. So listen to what they have to say so that you can target your testing most effectively.

Once they have given you all the information you need, say thank you and go test. And then remember that they gave you their educated opinion, and as nice as they are, you can't trust that they are right 100% of the time about the functionality their fixes affect.

If they *never* made a mistake and were *always* 100% right, you wouldn't have a job testing. So use the information they have given you to accurately regress the bug. Then take the information they gave you about potentially affected functionality and check that too. This is the thorough way to regress all bugs.

© Phillip Bailey - All Rights Reserved

Step #3: Validate Appropriately

You have the build and understand the fix. Now all you have to do is see if the bug still occurs. Easy! Test the bug one single time and if you don't see it, all is well, right? Uh, no…

When you regress a bug, you must **keep in mind the Reproducibility** of it. Why is that? Because you are a Quality Assurance Professional who, using numbers as ammunition, reports your results with hard data. This means that as you execute your bug regression, you are building that data to report.

In order to validate appropriately, you must **do** the right things so that you can **report** the right things. What does this mean? What are the right things to do? I thought you'd never ask…

When regressing a bug, you **follow the exact Setup and Steps to Reproduce to see if the bug occurs.** That part is simple and straightforward. Make sure you follow the exact same Setup and Steps that are reported in the bug.

If you see the bug occur – even once, then the test for this fix fails and you can now move on to Step #4: Comment Thoroughly. But, if the bug does not occur, what should you do?

If the bug you are testing has a Reproducibility of Always, then you should test it at least 3-5 times and maybe even as many as 10 times. If you do not see the bug occur, you should feel confident that the fix was successful. Since this bug was occurring 100% of the time, you can feel certain after 10 or so attempts that this bug has indeed been fixed and you can move on.

The key to this Step is in testing until you are comfortable stating with confidence that the bug no longer occurs. That level of confidence takes more than a single try to attain. So test until you feel confident that the bug has been successfully fixed. And, since you are documenting everything, make sure you note how many attempts you made to reproduce the bug without being successful.

If the bug you are testing has a Reproducibility of Intermittent, the number of attempts you will need to make to feel comfortable probably just went up. How many tries would you need to make in order to feel comfortable that an Intermittent bug was no longer occurring? 5? 10? 20? 100?

© Phillip Bailey - All Rights Reserved

It is really up to you – and the Intermittency. Of course, the Reproducibility was reported in a format that was impactful, right? The Reproducibility reads: "Intermittent: Occurs 6 out of 10 times." Right?

If that is the case, then you can probably test 10 times. If the bug doesn't occur at all but you don't yet feel confident, give it another 10 tries. If you don't see that bug in those 20 attempts, you can report that the bug no longer occurs with a high degree of certainty, right? Great!

But what if the Reproducibility as reported reads: "Sometimes. It happens about 10% of the time." Then you need to find the person that wrote the bug and…no, no, never mind…let's not do any of that. Being a true Quality Assurance Professional, you seize the challenge and do your best.

You attempt to reproduce the bug 20 times. If you see the bug at all, fine. The fix failed (when you comment the bug, please update its Reproducibility with information that is actually useful). But what if you attempt to reproduce the bug 30 times or 50 times and don't see it? What if you don't yet feel comfortable stating with certainty that the bug is fixed?

What if the Reproducibility as reported leaves you without adequate confidence to close the bug? In this case, you should talk to your lead and explain the situation to them. It would be a mistake to close a bug that you were not certain was fixed. There is a risk in this scenario that you may not be sure the bug is fixed, so what should you do? I'll tell you.

Talk with your QA Lead or Manager and explain the situation to them: Due to the reported Reproducibility, you are not comfortable closing the bug yet. You have made 30 attempts to reproduce the bug and have not yet seen it, but tell them that it is unclear how often the bug occurred in the first place. Suggest that you comment the bug with your most recent data and leave the bug open and assigned to you for testing in the next build.

If you will be getting another build, then there is less harm in testing the bug again than there is in closing it when it might still occur. So, what you should do is to properly comment the bug with the latest information (we will cover that in a moment in Step #4: Comment Thoroughly) and leave it open to be tested when the next build arrives.

© Phillip Bailey - All Rights Reserved

Then, of course, you must be sure to test it in the next build and add further data to it then. If after another 30 attempts, the bug has not occurred, you may feel comfortable closing it.

One place that I worked, we had a rule of "3 consecutive builds". That is: if a bug were very intermittent or otherwise difficult to reproduce, once a fix was in place we would thoroughly test the bug in 3 consecutive builds. If we didn't see the bug then, and only then, we would close it. It would also be added to our list of "most wanted" that we kept at our workstations to keep an eye out for.

The real key to regressing a bug in a way that Validates Appropriately, is to **test until you can state with a high degree of confidence that the bug no longer occurs.** Or, alternatively, test until you see the bug occur and can state with confidence that it still needs to be fixed.

Either way, **test until you are sure.** Test until you feel confident. Don't trust anyone else to tell you whether or not everything is good and happy. You are a Quality Assurance Professional. You carry the standard and set the bar for excellence. You are the consumer's advocate!

When you say that a bug no longer occurs, others will believe you. They will believe you because you are QA. They will believe you because they *want* the bug to be fixed. They will believe you because you have built a relationship of trust with them. So, be right and be accurate. **When you claim a bug no longer occurs and can be closed – have confidence!**

© Phillip Bailey - All Rights Reserved

Step #4: Comment Thoroughly

This Step is **where you use that data!**

- You tested the fix in the right build
- You understood the fix and so were able to test accurately
- You tested until you were confident of the outcome
- Now – Comment!

Here is where you get to show off all the information you have gathered. Just as important as how you wrote your bug, how you comment is crucial. If you comment in a way that gives the reader useful information, you have done your job as a professional and it is time well spent. If you comment like the average tester…well, let's just hope you understand the difference by now.

Once you have assembled all of your data, you can Comment Thoroughly. This is very simple. **A Thorough Comment contains 3 statements:**

1. **What you did**
2. **What happened**
3. **What you are doing because of what just happened**

Very simple…and yet.

Many testers don't include enough information. Just like a developer that comments a bug they have resolved by writing (again, I quote):"fixed" – if you don't include adequate information, you will cause confusion and more work for others. Sadly enough, there are many testers in the workforce today who don't understand why they should make this effort or what a Thorough Comment should look like.

© Phillip Bailey - All Rights Reserved

To help you have a clear understanding of how to Comment Thoroughly, here are a few likely scenarios and the comments I would use:

Scenario #1:

- Bug with Reproducibility of Always
- I try 10 attempts at reproduction
- The bug does not occur

Thorough Comment I would write #1:

"Tested version 1.0.a. Made 10 attempts to reproduce. Bug no longer occurs. Closing."

Scenario #2:

- Bug with Reproducibility of Intermittent (6 out of 10)
- I try 20 attempts at reproduction
- The bug does not occur

Thorough Comment I would write #2:

"Tested version 1.0.a. Made 20 attempts to reproduce. Bug no longer occurs. Closing."

Scenario #3:

- Bug with Reproducibility of Intermittent (1 out of 25)
- I try 50 attempts at reproduction
- The bug does not occur

Thorough Comment I would write #3:

"Tested version 1.0.a. Attempted to reproduce 50 times. Bug did not occur. Leaving open for further testing in next build."

Scenario #4:

- Bug with any Reproducibility
- I try 3 attempts at reproduction
- On the 3rd attempt, the bug occurs

Thorough Comment I would write #4:

"Tested version 1.0.a. Bug still occurs: Program crashes when help button is selected. Reactivating and assigning to <developer X>."

All that effort! See how easy that is?

© Phillip Bailey - All Rights Reserved

Just state what you tested, what happened, and what action you took. That way, the next time anyone looks at that bug they will have an understanding of why it is in the state it is in. If you encounter a bug that has been closed, you can see what action was taken previously.

If you do anything to a bug, add information or change a field or change the assignment – whatever – **you must Comment Thoroughly**. When I encounter a bug that is Closed, but there is no comment as to why, I reactivate it. If a bug is Resolved, but there is no comment, I assign it to the person that resolved it and ask why.

There is no reason other than laziness to not Comment Thoroughly. Forgetting to or deciding not to thoroughly write what you did and what happened is strictly amateur hour. The QA field already has enough of that. Separate yourself from the crowd and do the job of a professional.

State your information with confidence and Comment it Thoroughly. It is a habit that will make your daily work easier and more organized and will show that you are an aware Quality Assurance Professional!

© Phillip Bailey - All Rights Reserved

Step #5: Assign Accurately

Once you have regressed the bug, gathered and reported all pertinent data, it is time to either Close the bug or Assign it. The important part of this Step is to Assign your bug Accurately.

Depending on the outcome of your testing, you will need to take some action with the bug. If you have Validated Accurately that the bug no longer occurs, the bug needs to be Closed. Depending on how your company is set up, you may close the bug yourself or assign it to your QA Lead to close. If you are to close the bug, the scenarios above should be an adequate model. If you have to assign the bug to someone else to close, then Comment Thoroughly and make sure you state something like:
"Bug no longer occurs. Please close. Assigning to <your QA Lead> for closure."

That takes care of a bug that needs to be closed, but what if the bug you are testing still occurs? What if you Re-Activate the bug? Then what? Then, you Assign Accurately.

What I mean by this is that you need to make sure you assign the bug to the appropriate person. This may be the developer that attempted the fix, it may be the producer so that they can track how many bugs were fixed and how many re-activated per build, or you may need to assign it to your QA Lead so that they can verify your work and then assign it to the correct person.

All companies have their own policies for this, but the most common practice is that you would Re-Activate the bug as assign it back to the developer who attempted the fix most recently. When you do this, make sure you have written a Thorough Comment and then Assign Accurately. Assign the bug to the correct developer. If the developers have to play hot-potato or ping-pong with the bug once you assign it to them until they can figure out who it should be assigned to, you have created more work for them.

© Phillip Bailey - All Rights Reserved

Make sure you know who to assign the bug to and then do so. Now you are free to move on to the next bug on the list.

So there you have it!

Remember the 5 Steps to Bug Regression

1. **Know the Version**
2. **Understand the Fix**
3. **Validate Appropriately**
4. **Comment Thoroughly**
5. **Assign Accurately**

Follow those Steps. Practice them as perfectly as possible until you can do them in your sleep. Once you master these details, you will have increased the value that you bring to every project you participate in. Pay attention to what you are doing. State what you do with authority and confidence. Take the correct action and do the right things.

And when you do the right things, be sure to do them in the right way.

© Phillip Bailey - All Rights Reserved

Chapter #12: The 5 Most Common Bug Regression Mistakes

Failure is the opportunity to begin again more intelligently.
~ Henry Ford

When executing Bug Regression, there are many hazards that a tester may encounter. I will explain each of these so that you will know them and be able to avoid all of them.

To set the stage for this part of our story, let us return to a point in the project just before QA begins performing Bug Regression. Let's go back to a simpler time. Back to when a developer implements a fix.

The developer has looked at their list of bugs and has fixed each bug that has the designation of Priority 1. When they fix the bug, they *should* change its state to Resolved and assign it to QA…but in the real world, this doesn't always happen.

Sometimes the developers are so caught up with so many bugs that they forget to Resolve and Assign one of their bugs, or they Assign it to QA without Resolving it. When this happens, it is up to QA to make sure that the appropriate steps are followed.

When it comes to Bug Regression, there are many places for many people to make a mistake. As an elite-level Software Quality Assurance Professional, it is your job to identify where the process is failing and to help the appropriate people correct the failings. If you don't, you won't be able to do your job effectively. You will have to work around other's mistakes and the integrity of your resulting data will be at risk.

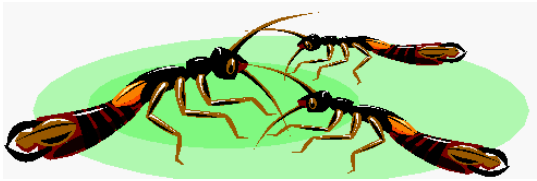
So, when the time comes to perform Bug Regression, you must do your best to ensure that all the appropriate steps leading up to *your* work has been performed correctly. You should hold others accountable for their jobs just as you are accountable for yours. QA is a *part* of the Software Development Lifecycle, not separate from it, neither outside nor above it, but an integral part of it.

© Phillip Bailey - All Rights Reserved

Here is a list of the most common errors made leading up to and during Bug Regression. Please do all you can to avoid or correct them and I assure you your daily work life will be much more pleasant.

The 5 Most Common Bug Regression Mistakes

1. **QA Resolves A Bug**
2. **Non-QA Closes A Bug**
3. **Not Commenting**
4. **Commenting In The Wrong Place**
5. **Not Asking for Help**



© Phillip Bailey - All Rights Reserved

Mistake #1: QA Resolves A Bug

QA can Open bugs. QA can Assign bugs. QA can Reassign bugs. QA can Reactivate bugs, and QA can Close bugs. But QA should NEVER Resolve a bug.

Either a developer or producer should resolve a bug. They are the ones who know if a fix has been implemented, or a change made in documentation, or whether the current behavior is what they desire – they Resolve bugs! They are the parties accountable for changing the Status of a bug to a Resolved State.

QA DOES NOT RESOLVE BUGS! If you are testing and you notice that a bug is no longer occurring, great! Perform thorough testing to accurately validate that the bug no longer occurs and then make a comment in the bug. Contact the person that the bug is assigned to and let them know what you have found – that the bug seems to be fixed. They can check their notes or the code and, if warranted, they can then Resolve the bug. Once Resolved, you are welcome to re-test the bug and then close it if it truly is fixed, but don't assume that it is. Have a developer check. Once they comment the bug and change its Status to Resolved, only then should you go forward.

Some think that it is a conflict of interest for QA to resolve bugs. Others believe that it removes the accountability from other parts of the project team if they are not the ones to resolve a bug. This is their responsibility, not the responsibility of QA.

Trust me; you will have plenty of other work to do. Let those who are accountable for resolving bugs do so, and concentrate on being accountable for you own job. Once you have mastered that, then we can investigate branching out.

With that said, this brings us to our next point…

© Phillip Bailey - All Rights Reserved

Mistake #2: Non-QA Closes A Bug

Although you may need some outside input to regress, test, and verify certain bugs, ***only* QA should close bugs**. This will keep the accountability for all bugs of the project within QA.

When only QA closes bugs, it is much easier for them to track the progress of the project, build metrics, and make forecasts. Additionally, QA has the first line of responsibility and accountability whenever a bug appears in the field.

Let me say that another way:
If ANY bug appears in the field, the first question by the company is, "How did we miss that?"

That question prompts people to look to QA for the answer. *How did QA miss the bug?* Although QA may have reported the bug, there are many reasons why the bug may not have been addressed:

- The fix was too risky
- The fix was too costly
- The fix would delay the ship date

So as you can see, it is possible that QA can identify a bug yet it still may be seen by the public. In this case the bug would have been Deferred (action performed by producer or lead developer). When the bug is tracked down in the bugbase, QA can point to it and show that it was Deferred, not Closed. This means that the company knew it was shipping with this issue.

If, however, the bug had been Closed? Then what?

Then the Question becomes:
"Why did we close a bug that was, and is, obviously still occurring?!? QA, what do you have to say for yourself?"

Because now what has happened is that a bug was reported as fixed and VERIFIED as fixed – thus the company believed that the bug no longer existed. The bug was reported to NO LONGER BE ANY THREAT WHATSOEVER. So the Question now is, "How did QA screw up?"

This is never a Good Question (but you probably figured that by now).

© Phillip Bailey - All Rights Reserved

So, *how* did QA screw up? – OR – *Did* QA screw up?

There are a couple of possibilities, but hopefully QA didn't screw up. Hopefully QA did their due diligence, but a rogue producer or developer decided they would go ahead and close a bug because they hadn't seen it in a while. Pray that this was the case. There is, of course, no way QA would *ever* close a bug that was still occurring, right?

If you have executed all your tests and performed your job as a true Quality Assurance Professional should, the chances are that you ***did not*** close a bug that was still occurring. But how would you like to be accountable for anything going wrong in the field after allowing someone else to validate an implemented bug fix? Does that sound like a good idea to you? Not me.

So make this a rule of thumb: Only QA Should Close a Bug (just make sure you only close bugs that no longer occur).

© Phillip Bailey - All Rights Reserved

Mistake #3: Not Commenting

This cannot be stressed enough. Every time you test a bug, change a Status, add information, Re-Assign, or modify a bug in any way you MUST add a comment to the bug. **You must clearly state what you have done to the bug** so that everyone will have the same understanding of the bug and what needs to happen to it next.

If you don't comment, then none of what you have done can be tracked and is, thus, not helpful. If you make any changes to a bug, write what you did in the bug. That way, the next time someone looks at it they can see the history of the bug and will know what to do with it.

Any changes made to a bug (especially changes to Status, Priority, etc.) will be mysteries or thought to be mistakes if the bug has no comment about who did the change, what the change was, the reason for the change, and the date of the change.

This applies to adding information as well. How else will someone know that the bug has more information now when they look at it than when they looked at it the last time if there is no comment telling them that?

To be fair, I should point out that some bugbases capture some or all of this information automatically. They will post automatically in a Comments or Bug History section who made a change and what change was made. In this way, there is some information visible to future readers of the bug. However, a bugbase cannot capture rationale.

Even automatic bugbase change tracking can't reason WHY a change was made. So, even if your bugbase will automatically comment every time you change the Status of a bug, it is still up to you to provide a reason for the change. Go back and look at 5 Steps to Bug Regression: Step #4: Commenting Thoroughly to get an idea of what to add.

I have worked on multiple projects where team members could not be bothered, or didn't understand why, to add comments when they changed a bug. These project always become more maintenance-heavy (and thus more expensive) because of the effort wasted due to people who don't comment bugs appropriately – like making no comment at all.

© Phillip Bailey - All Rights Reserved

As I am reviewing bugs daily, if I find one that has an action taken that isn't commented, I immediately take corrective action. If the bug has no comment, it gets returned to its most recent previous state. That is to say that if a bug has been closed, but there is no comment about who tested it, what version they tested, what happened, and what action they have taken, I Re-Activate the bug.

That's right. Here is a bug and it's Closed. But there is no reason given in the bug as to why it was changed from Resolved to Closed. I must assume that someone made a mistake, because no one on my QA team would ever do such a thing (and just to be sure, I actually do ask if anyone did this – they have always truthfully assured me that QA had nothing whatsoever to do with an episode such as this).

So, here is a Closed bug. No stated reason for it to be closed, I Re-Activate it. It might have been fixed. Someone might have verified the fix. But since they couldn't be bothered to share that information with anyone else, Re-Activate the bug and test it – or test it again as the case may be.

If a bug is going to be closed on my watch, I make sure that QA knows the bug is fixed.

Make everyone's life (including your own) less complicated: **Always Comment.**

© Phillip Bailey - All Rights Reserved

Mistake #4: Commenting In The Wrong Place

When reviewing the history of a bug, it is most helpful when the most recent changes are at the top of the Comment (or Bug History) field. This way the most recent, pertinent information you need is right there in front of you. You don't have to scroll to the bottom of the bug, through all of the many comments that have been added throughout the life if the bug, just to find the most recent changes.

Save yourself headache and time, and save your company money by organizing your bugs in the most efficient and effective way possible. Put the newest, latest, and greatest comments at the top of the Comment field.

Some bugbases actually control this for you – sort of. They control where the comment goes (chronologically), but in this case you have control of how you view those comments.

You have the choice of the oldest comments at the top of the field, with the age of the comments descending in order until they reach the most recent (meaning you will have to scroll much of the time). Or you can choose to have the newest comments at the top of the field, with the age of the comments getting older and older as they descend down the page.

You should select whatever is most comfortable and efficient for you, but I recommend configuring your page so that when you view a bug, the most recent comments are right in front of you. And if you have a choice as to where to add a comment, **be sure you place the newest comment at the top!**

© Phillip Bailey - All Rights Reserved

Mistake #5: Not Asking For Help

There will be times when you will need assistance to regress a bug. When this happens be sure you do what it takes to receive that help. There are many testers who are afraid of other members of their project team; within and outside of QA. This is not sustainable behavior. QA is a necessary part of the project team, and as a part of the team QA must be able to effectively interact with other team members.

There are times when you, as a tester, will need to motivate other team members to lend a hand. Here are some common scenarios:

If the bug involves a judgment call to verify, you may need help. If a bug reads, "The text is too green." then you will need to track down the person who entered the bug or the person responsible for deciding just *how* green the text *should* be in order to verify the desired "green-ness" for the text. You will need help because, obviously, *you* would never enter a bug this ambiguous!

Once this judgment is made by the appropriate party, then you either Verify, Comment, and Close the bug, or Assign back to the appropriate party to continue to attempt fixing the green text – always commenting on the actions you have taken

If verifying the bug requires Engineering to explain the fix and show you why it is working, then by all means have them explain themselves to you. You are the one that will be responsible for Closing the bug, so make sure that you fully understand what was fixed, what the current behavior is, and why it is an acceptable fix.

While performing this review with Engineering, it may become apparent to both you and the engineer that this fix is not appropriate and that a different fix must be implemented. When this is the case, is it up to you to ask the right questions until you understand the fix and help the engineer understand how their code is interacting with the rest of the program.

Don't walk away until you understand. Don't pretend that you understand just so they think you're smart! The key here is to understand what is happening in the program and to be able to make an accurate call as to whether or not the bug is actually fixed. **If you need help, ask for it!**

© Phillip Bailey - All Rights Reserved

You have just learned all you need to know to begin your mastery of Bug Regression. You have the 5 Steps to follow and you have learned the 5 Most Common Errors to avoid. You have the tools to take the Bulletproof Bugs you have written and finish them up.

The 5 Most Common Bug Regression Mistakes

1. **QA Resolves A Bug**
2. **Non-QA Closes A Bug**
3. **Not Commenting**
4. **Commenting In The Wrong Place**
5. **Not Asking for Help**

Sidestep these common traps and your work will remain professional from start to finish. Your work will be consistent, you will be consistent, and that will build trust. You will be able to bring your greatest strengths to bear each day as your consistency will allow you to focus on the issues that are most pressing. You won't be distracted by having to answer questions about the meaning or relevance of your work.

© Phillip Bailey - All Rights Reserved

You are ready to join the party!

The future has several names. For the weak, it is the impossible. For the fainthearted, it is the unknown. For the thoughtful and valiant, it is the ideal.
~ Victor Hugo

Work while you have the light. You are responsible for the talent that has been entrusted to you.
~ Henri-Frédéric Amiel

With all that you have learned, you are now prepared to fully participate in all aspects of any project and carry your own weight. You will be able to provide value as more than just a tester; you can carry the Standard of a Software Quality Assurance Professional.

You can find and report all the issues in a program and, once fixed, you are able to verify all the fixes and Close the bugs. You can track all of the work that you perform and are able to report it with impact and certainty. You have the tools to play a leading role in bringing high-quality products to market!

You began the project, reported all the bugs, tracked your progress, completed all of the test suites, and regressed all fixed bugs. The program is ready for release to the wild. Will the public embrace it? Did you advocate effectively for the consumer? Did you help to create a product that you would be happy to own?

Whatever the outcome, you have wrapped up your project. You ran it from beginning to end and learned every step of the way. You are ready for the next project – or did they throw that one at you last week while you were trying to finish this project?

You are ready to move on…but wait!

Hold it right there, Buck-o!

Not so fast, Senorita

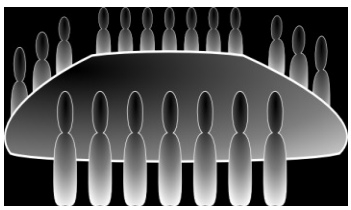
You…forgot…one…thing…The Post Mortem!

© Phillip Bailey - All Rights Reserved

Chapter #13: How To Ace A Post Mortem With 3 Actions

The test is to recognize the mistake, admit it and correct it. To have tried to do something and failed is vastly better than to have tried to do nothing and succeeded.
~ Dale E. Turner

Now that you have wrapped up your project, we turn to what you should do when a project has been completed: The Post Mortem. The Post Mortem is where you will take all you have learned during your project and turn it into replicable efficiencies. The Post Mortem is an opportunity for the team to meet without a looming deadline, share ideas about what went right, brainstorm causes of what went wrong, and agree on what can be implemented so that the next project goes more smoothly.



A Post Mortem is a meeting that is held at the end of a project. At a Post Mortem meeting, the team discusses all of the things that went wrong, everything that went right, and where and how they or the process can improve. Ideally, the entire project team should participate in this meeting. It is a great opportunity to freely discuss the whole project.

Before I get into detailing the joy of a Post Mortem meeting, I should point out that not all companies hold these meetings. I have worked with companies that are always in such a rush to get the next widget out the door that they don't take the time to hold a meeting whose agenda is focused on how to make the company better. If you find yourself at a company that does not support holding a Post Mortem for the whole project team, then you have a choice:

- Either get a job with a company that takes improvement seriously

– OR –

- Be proactive. Speak to your QA Lead or Manager and tell them you want to hold a Post Mortem for the QA team

© Phillip Bailey - All Rights Reserved

With companies that do, indeed, hold Post Mortem meetings, there are many different formats that are used. There are many ways to hold an effective Post Mortem. Most of the differences can be traced to differences in management style. But no matter how the meetings are run, your job remains the same.

Ace a Post Mortem With 3 Actions

1. **Prepare**
2. **Participate**
3. **Produce**

© Phillip Bailey - All Rights Reserved

Action #1: Prepare

Once your project is complete, it is time for you to prepare for your Post Mortem meeting. The most effective way to prepare is to write a report summarizing your project in four sections:

1. **What Went Right** – Make note of all of the hurdles that you overcame, list all of the milestones that were reached on time, detail all of the improvements that you implemented during the project that made it run more smoothly, explain any efficiencies found during your work on the project.
2. **What Went Wrong** – List all of the issues that impeded testing, all of the incidents that affected the schedule, any gaps in testing that required scrambling to make up for. Write down any issue that had an adverse affect on the project. This can be difficulty in communication, lack of clarity around expectations, lack of response to reports; anything that made the project more difficult is fair game. Note that this part of the report is often given a name with a more positive spin than What Went Wrong. It may be called What Can Be Improved or What Didn't Go Right or What We Can Do Differently – whatever it is called; the goal of this section of your report should be focused on the areas of the project that could use improvement.
3. **What You Learned** – In this section you will want to outline all that you learned on this most recent project. What did you learn about communicating with other departments? What did you learn about bug metrics? What did you learn about how to do your job better?
4. **How You Will Improve** – In this section you will detail all of the improvements you believe should be implemented. For each one, you will include specific actions that can be taken to make the improvement a reality. You will explain the benefit(s) each improvement will contribute to future projects. Since you have been subjected to all of the ups and downs of the project, you should have some very clear opinions about what could be done differently so that the processes that make up the your projects would be better.

This report you will use in two ways:

1. As detailed notes that you can reference during the Post Mortem meeting (so that you don't have to remember everything)
2. As the final end-of-project report that you will submit to your QA Lead or Manager to highlight your involvement in the project

© Phillip Bailey - All Rights Reserved

Take the time that you need to write your report so that it has value. When you are done, you should have a report that clearly states your insight about the project.

This report is not for the faint of heart, it is not for those that go with the flow. Your Post Mortem report should be a straightforward telling of the great things the project accomplished, all of the weaknesses that the project exposed, the high points of what you learned during the project (this is *very* different for each person), and actionable suggestions for what improvements should be made.

You will want **your report** to be as clear and direct as possible. This report **should be focused on how to move forward and improve**. You identify what *did* happen and then propose actions that *you* can take to improve projects in the future. This is why your report must be clear and direct.

Once your report is ready, you will…

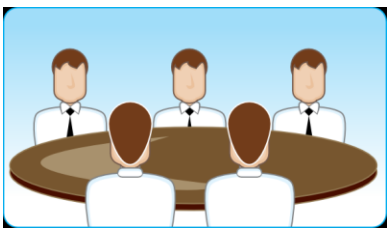
© Phillip Bailey - All Rights Reserved

Action #2: Participate

A Post Mortem meeting is a forum in which everyone should feel free to speak their minds. This is where you can talk about the project and gain consensus about how to move forward more effectively. A Post Mortem meeting needs to be an environment in which all parties can participate equally.

Your job in this meeting is to contribute as much as you can, to listen attentively, *and* to help others contribute. Be open with your facts and opinions, be direct in making your points, and by all means…be nice!

Going around the room or by opening the floor to the whole team, the meeting should follow the same outline as the report you have written. Openly discuss each of the sections of your reports one at a time. As each section is discussed, have one of the meeting participants capture the high points on a white board where everyone can see and discuss them.



Begin with What Went Wrong. Go around the room and have each person contribute at least one item to the list. Everyone in the meeting was involved in the project and unless the project went smoother than any other in history, each team member was affected by at least one less-than-perfect situation. Talk about each one. Give a voice to what happened. This is the first step in your ability to move beyond the difficulties your project presented.

Talk about What Went Wrong. Get everyone's input. This should be an open discussion, not a debate. The goal is for everyone to understand all of the issues even if they were not personally affected by each one. Strive to understand. You don't all have to agree, but you must understand. Keep to the point and don't let this discussion turn into a negativity session.

Once you have covered all that went wrong, move onto…What Went Right.

Again, one-by-one, present your observations about **What Went Right.** What about the project was positive? This can be more difficult for some people. If it is, help them along. Be aware of what your team members contributed to the project. Maybe they are not fully aware of how the way they did their job made your job easier. Focus on all of the things, large or small, that helped the project succeed.

© Phillip Bailey - All Rights Reserved

After you have all patted each other and yourselves on the back talking about all of the wonderful things that you did, wrap up this section and move on. Now it's time to talk about What You Learned.

Every single team member should have at least two items they can talk about. What did they learn during the course of the project? This part of the meeting should be an enlightening discussion for all involved. It can be uplifting and educational when you gain an insight into the learning process of your teammates.

Talk about What You Learned. Explain to the rest of the team why each item was helpful to you. Talk about what you wanted to learn and were able to. Discuss how the learning occurred.

If you can discuss the learning process in this way with your whole team, it will help everyone understand better how they can improve. It will offer other horizons for your teammates to be aware of and other avenues that are available for them to pursue. Talking openly about what each team member and the team as a whole learned during the project is a major step toward creating continual improvement. If you want to improve, you better understand what you have learned.

Once you have wrapped up discussing all that you learned, turn to how best you can now improve. In this portion of the meeting you will really get into the nitty-gritty of process improvement.

© Phillip Bailey - All Rights Reserved

Action #3: Produce

Having discussed all that went right, all that went wrong, and all that you learned during your project, your team is now primed to take the bold step toward improving the work that you perform and the ways in which you perform it. Now you get into **How You Will Improve**!

In this part of the meeting, get all of the ideas on the board first. What I mean by that is that *before* debating the merits or potential of any of the ideas for improvement, get them all out in the open. Have each team member give a suggestion. Do this one-by-one until no one has any more ideas. As each idea is offered, list it on your white board so that everyone can see them all.

Get them all out. This is very much like a brainstorming session – any idea is fair game at this point! Get every idea for improvement out and get it on the board.

Once all of the ideas are listed, begin to discuss the benefit of each. Which improvements would be most impactful? Have the team talk about each improvement and then give each a rating. Number each improvement 1, 2, 3, etc. to designate the level of impact the team believes the improvement would produce.

Once you have designated the impact of each improvement, you must decide *how* each improvement could be implemented. It is important to have specific steps that can be taken immediately to create the desired change. If you have written your report in a well thought out manner, you should have a list of possible actions that would produce the outcomes you have suggested.

From the list you have created, pick the improvements that the team agrees will be implemented. This can be a list of the top 3 or the top 10; it's up to your team. Decide on a list of improvements that you will commit to implement.

Each improvement on the list should have a set of actions attached to it that will be taken to produce the desired change. Each action item should have a specific person assigned to it and a date by when the action will be completed.

© Phillip Bailey - All Rights Reserved

This gives you:

- The agreed list of improvements
- The actions that will create each improvement
- The team member that is responsible and accountable for each action
- The agreed upon completion date for each action

Armed with this plan, you can now begin to affect real change in your processes and in your organization. Your list of action items may include participation that you need from other departments. An action may need more than a single team member to implement. This is fine. Work as a team, that is all the better. Just make sure that each item is owned by a single person. Only one person may be accountable for each action. Follow this guideline and you will see greater improvement sooner and will experience less confusion.

After having made the decisions about what improvements to implement and how to implement them, your next step is execution. You must be able to follow through individually and collectively, as a team, in order for any of what you just did to have any meaning. How can you help this process along? Glad you asked!

When the meeting has been completed, one of the team members will need to create a summary report of the meeting and distribute it to the whole team. This report is simple. **Capture the decisions that were made to affect improvements:**

- What will be improved
- What the improvement will affect / What the benefit will be to the team
- What steps will be taken to implement the improvement
- Who is responsible for each action to be taken
- When each action will be completed

With all of this information detailed, each team member is accountable to the whole team. Everyone will know what they are responsible for and when their tasks need to be finished. This will give the team a solid timeline of when they should expect improvements in the way that projects are handled. This will make them even more aware of areas that are not as efficient or as effective as possible. This begins the continuous-improvement virus.

© Phillip Bailey - All Rights Reserved

Since this report is so simple to compile, since whoever creates and distributes it will look as though they are brimming with initiative, and since the team member that compiles the report will gain even greater insight into how to improve the project processes…who do think I am about to suggest should write this report?

If you have read this far in the book, I am sure you know that I **think you should write this report**. It is another chance to separate yourself. It is an opportunity to learn more. You should be jumping at every opportunity like this that comes your way. If you do, you will be getting paid to learn and positioned to shine.

Write the report and send it out to the team. Show your boss that you are the person that is interested in doing the right things and doing them in the right way. Seize the opportunity to learn and practice and gain a deeper understanding of the work that you do.

That is how you can most effectively achieve in the Post Mortem process.

- **Prepare** – By writing an insightful report that shows you are a Quality Assurance Professional that is committed to learning and improving by defining what happened and then creating a plan to do it better the next time.
- **Participate** – By being prepared, you can fully join in the meeting and help others participate also. You will open avenues of discussion whose goal is understanding, if not agreement.
- **Produce** – By writing and submitting the plan that will be followed to create improvement going forward.

After you have prepared your own report, participated in the meeting, and then produced the report for the whole team you are truly ready to wrap up this project. The testing has been completed and now the reports are achieving completion.

Send the compiled report you produced from the meeting contents to the whole team. Then submit your own, personal Post Mortem report to your QA Lead or Manager (and be sure to keep a copy for yourself). This report that you send to your Lead or Manager is your final “Here is my view of the project” report. This will give them the explicit information that only you, with your unique perspective can provide.

© Phillip Bailey - All Rights Reserved

Ace a Post Mortem With 3 Actions

1. **Prepare**
2. **Participate**
3. **Produce**

Create your reports. Discuss the content of your reports in the meeting. Compile the agreements for distribution. Once you have submitted your reports, get ready and gear-up for the next project. It is surely on its way!

© Phillip Bailey - All Rights Reserved

Chapter #14: The Top 5 Challenges of Being a QA Tester

Better to light one small candle than to curse the darkness.
~ Chinese Proverb

There are several challenges that you will encounter as a Quality Assurance Professional, but the principles, methods, and tools in this book will help you to minimize or overcome all of them. The major challenges you will experience working in QA include, but are by no means limited to:

Top 5 Challenges of a QA Tester

1. **Lack of Respect**
2. **Ignorance About QA**
3. **Position of QA**
4. **Lack of Tools**
5. **Miniscule Margin for Error**

There are completely valid reasons for each of these challenges and I am in no way claiming that QA is the only job that has hurdles to overcome, but the challenges I experienced as a Software Quality Assurance Professional, I found to be unique to QA. Let's look at each of these challenges so that you better understand them before you have the pleasure of their acquaintance.

© Phillip Bailey - All Rights Reserved

Challenge #1: Lack of Respect

*A priest asked the dying Spanish statesman and general **Ramon Maria Narvaez**,*
"Does your Excellency forgive all your enemies?"
"I do not have to forgive my enemies," answered Narvaez, "I have had them all shot."

Many companies I have worked in and worked with view QA with a suspect eye. They view QA as a necessary part of the project process prior to shipping a product, but have great difficulty trusting the work that is performed by the Quality Assurance team. This lack of trust is, unfortunately, not without reason.

This view has evolved because too often companies fill their QA departments with gamers or other computer literate humans that have the ability to launch software and follow basic instructions. They are labeled "QA testers" or "QA associates" or "QA engineers" for no apparent reason other than it must look better on a budget report or an account ledger than "warm body". Though there is nothing wrong with these people, they are by no means Quality Assurance Professionals.

These people are certainly NOT quality assurance engineers. And I don't even know what a "quality assurance associate" is supposed to be. These people are gamers that are able to follow the steps in a test case – thus they perform "testing" and are therefore considered "testers".

This is not their fault. This is the fault of the companies that hire any warm body they can find, tell these people they are testers, and then are surprised when their product ships late, over budget, and not as functional as it should have been. Then the company asks, "How could this have happened?"

The belief that these companies seem to hold is that if anyone uses the product, they are testing it. They will tell the company what is not functioning. If there is a piece of functionality missing that this "tester" doesn't notice, then maybe it wasn't such an important piece of functionality anyway.

You should understand by now that real Software Quality Assurance Testing takes practice and skill to be performed effectively. A tester must be able to do more than simply launch the program. Unfortunately, since many companies do not seem to truly understand the value a true Quality Assurance Professional can

© Phillip Bailey - All Rights Reserved

bring to a project, they may hire “testers” that are not at all qualified to wear that title.

When these “testers” are viewed by professional developers and producers, they appear (accurately) as not up to the task. When the company calls them “QA”, they leave a stigma for real Quality Assurance Professionals to contend with.

This “throw-as-many-warm-bodies-at-it-as-you-can” or “shotgun-testing” style of QA has proven (at least in the short-term) to be very cost-effective for many companies. It does not, however, ensure professional testing of the company’s products. These gamers are happy to get paid to play games for a living – and I am not here to judge that – but they are not trained in proper testing methodology, bug reporting, or bugbase use. They are not real Quality Assurance Professionals! That the company calls them “testers” is where QA begins to lose the respect of other departments. Often these employees do not act professionally but they do represent QA.

These are the main reasons that other departments do not respect QA. This is the environment you have chosen as a challenge. But take heart: History has clearly shown that a small group (2-3 people) of elite testers will outperform a larger group (20-30 people) when working head-to-head on the same project! In fact the results are not even close! This book has given you the tools to be one of these elite Quality Assurance Professionals!

To overcome this lack of trust that may greet you, just **follow the lessons outlined in this book:**

- Consistently deliver hard data results
- Stay up to date on the status of your project and the issues affecting it
- Add value in every endeavor
- Never stop learning

As you master the lessons in this book, you will quickly overcome the view others may have about QA. You will build trust among your peers and will be seen as a professional who will add value to any project you are a part of.

© Phillip Bailey - All Rights Reserved

Shotgun vs. Scalpel – OR – How Two Outperformed Twelve

Many companies waste enormous resources by trying to get by relying on, what they hope are, comprehensive test plans and by throwing as many computer literate people in front of their product as possible. With an enormous suite of tests to be followed, companies will hire as many warm bodies as they can to follow these tests in the hopes that with enough people executing their tests, their project will be thoroughly tested and be shipped to the store shelves on time.

This is often referred to as Shotgun Testing. Fire as many people/pairs of eyes as you can at the product, no matter the skill level, and hope that they find "enough" bugs. "Enough" seems to be defined by:

- Achieving target sales numbers
- Unfixed or unfound bugs being tolerated by the consumer
- Keeping the consumer confident enough in your product that they will purchase the next product

Unfortunately, Shotgun Testing often has several detrimental results, whether or not the company's target sales are achieved. All of these results cost a company money, sooner or later

- Delays in getting the product to market
- Shipping a product that still has WAY too many bugs for easy consumer use
- Having to manage a constant turnover in the staff that has been "trained" to test for the company
- Consumers that tolerate a far lower quality of software than is possible

There are, of course, other ways to do things.

Let me relate a brief anecdote to illustrate this point…

Many years ago I worked for a Quality Assurance company that did only testing. We did not create any products, we tested them. We tested web pages, games, internet radio apps, educational software, server-client database programs, etc. If it was software, we tested it. And we delivered elite testing for every product.

We had one client in particular for whom we had been testing for a few years. As they migrated their products from paper-based games to CD versions, we delivered great results for them time and again. As that company grew, they were acquired by a larger game company. This larger company had originally

© Phillip Bailey - All Rights Reserved

built its success on board games and had recently been trying to become competitive in the software game industry.

When our long-time client was acquired by the gaming giant, our client was told that all QA would be performed "in-house" by the larger company's "interactive" software testing department. Would you be surprised to learn that this larger testing department applied the Shotgun Testing method? Well, if so, you shouldn't be.

As a new product came up for testing, the producer for the product (whom we had worked with quite successfully before) called us to say that she was sorry but she was not going to be able to give us the business. She was, however, working on a proposal to give us a chance to show the cost savings the larger company could realize if they contracted us for testing in the future as opposed to using their own "in-house" team.

A couple of weeks later, that same producer called again to tell us she had been given the go-ahead. She had been given a budget for 2 of our testers for 2 weeks. She proposed that we test the same builds at the same time that the other team was testing and she would show the difference in the results of testing and in the cost (since we were not to be given access to the results of the "in-house" testing team).

We jumped at the chance! Not only did we want the very lucrative contract back in ***our*** *house, this was a chance to show off our superior testing skills (at least, that was our opinion).*

Being a project that had not been taken into account on our schedule, we had to decide who we would assign to the testing. We had one tester that was finishing up a project. For the sake of this story we will call him Jimmy. Jimmy was one, but we needed one more. Since I was spending the bulk of my time managing the QA Leads and the interaction with their clients, it was decided that I would be the other tester. Jimmy and me against a team of, what we discovered later, was twelve testers that the gaming giant thought were some of their best.

I was thrilled to have the chance to test with Jimmy. He was, and is to this day, one of the finest testers I have ever seen. He is not the fastest tester I have worked with, but he never missed a bug. Nothing got by him. He was the most meticulous, detailed tester I have ever known. It was rare that Jimmy ever made any kind of mistake and if he did, he never made it again. Testing with Jimmy raised the level of everyone who tested with him!

© Phillip Bailey - All Rights Reserved

We received our first build late Monday afternoon. Just enough time to install it and read through the release notes. Our testing began in earnest on Tuesday morning. Jimmy and I looked at the program for the first time and met to decide on a plan of attack. We decided to go to our strengths. I would hammer all of the functionality and Jimmy would dissect the datasets. And off we went!

By mid-afternoon Tuesday we had logged 40 bugs, most of which we identified as bugs that were keeping us from being able to test other parts of the program. Every bug we logged was preventing us from accessing further functionality. At that point we had done all we could and we were stuck.

We emailed the producer that we had done what testing could be done and would need another build before we could continue into any more testing. She responded immediately that she had seen our bugs, understood that we were unable to continue, and was wondering what the other team was doing.

As it turned out, the "group-of-twelve" was still testing away. They were, apparently, unaware of all of the issues in the build that prevented further testing (chalk up "1" for the home team!). The producer was not happy at this most fundamental shortcoming, but said she would be sure they got a new build to us ASAP.

True to her word, the developers got some fixes in and overnighted a build to us that we received on Wednesday morning. Our assault continued!

Jimmy and I spent the rest of the week executing our respective tests. I logged tens of issues relating to blocked functionality, incorrect functionality, repeated patterns that precluded proper game play, etc. and Jimmy tore apart the datasets!

Jimmy logged every typo, incorrect copyright, incomplete sentences, inaccuracies, and every other issue that affected the game's datasets. One bug in particular stands out to me to this day.

Jimmy logged a misspelling of one of the game character's names. The bug was assigned back to him later that day with a note that read (I'm paraphrasing here):

> The name is spelled correctly. We have checked the documentation and with <the lead programmer>. This is not a misspelling.

© Phillip Bailey - All Rights Reserved

But Jimmy knew better. This character had a Japanese name and Jimmy could see that it was not possible to spell a name that way in Japanese, so he persisted. He assigned the bug back to the developer with a very diplomatic note explaining that it was not possible to spell a Japanese name in this way. He also included, very diplomatically, three different links to official game web pages where the character's name was spelled correctly. He pointed out that these were official game sites (that the company we were testing for owned) and detailed where on the page you would find an accurate spelling.

The bug was again assigned back to Jimmy with a note:

The name is spelled as it is supposed to be spelled. Please don't tell us how to spell the names of our characters. This is not a bug.

Knowing that this error would be major to a consumer who was an avid player of the game, Jimmy persevered. He again very diplomatically explained why it was not possible to spell a Japanese name is this way (this had to do with the name being spelled using letters/sounds that exist in English, but not in Japanese – and he was, of course, right). Jimmy then included 3 links to websites that showed how to read Hiragana and Katakana ("simplified" Japanese alphabets).

The next time we saw the bug it had only one additional comment: Fixed.
And, sure enough, it was!

The next Monday, a new build arrived. Jimmy and I tested all of the fixes and completed our testing sweep, hoping all the time that our work would show that although they were spending more per tester with us, if the gaming giant contracted with us, they would not need to spend as much overall as we could do more with fewer, higher quality testers. We finished testing and waited to hear back from the producer about how the "experiment" went.

As you might have guessed, the results showed that we provided far superior testing with far fewer testers. Although we did not get to test every game that the company subsequently developed, they definitely kept us busy going forward.

There was, however, almost a major glitch after all of that work we did.

We had two upcoming releases planned for testing with that company when, one day, my boss came flying out of his office and landed on my desk. He had just gotten off the phone with the producer in our story above and he was frantic.

© Phillip Bailey - All Rights Reserved

"They shipped 800,000 units internationally and they have to recall them! How did we miss a bug that will make them recall 800,000 units?" He asked me. I felt a gurgle in the pit of my stomach. Not sure if I just needed to go get lunch or look for a new job, I ventured on.

"What bug? What recall?" I queried.

"The project you and Jimmy worked on! We missed a bug that is making them recall 800,000 units worldwide! How did you guys miss something like that? We just got the contract back and now they are threatening to pull their projects that are scheduled with us! How did you guys miss that?" He calmly explained.

I tried again, "What was the bug?" I thought about it a little more and realized that this didn't *make sense. I think rather highly of my own testing skills and I* knew *Jimmy wouldn't have missed something like that. What could this be? "What bug is making them recall 800,000 units?" I asked.*

Not to be placated by my most calming demeanor, by boss blurted, "Foreign characters! The game doesn't recognize foreign characters. You can type them in, but it won't remember the username. English is fine, but the game can't deal with any non-English character!"

"Wha…?" It seemed to me that, indeed, that could *be an issue since the game had been shipped to several countries worldwide. "We couldn't have missed that!" I hoped. "Give me 30 minutes to look at our documentation. I'll let you know in 30 minutes what happened." I offered.*

"Hurry!" and he was back in his office.

So, Jimmy and I compared notes. We looked through the documentation that had accompanied the builds. We loaded up the latest version that we tested. And then both exhaled a deep sigh of relief…

Fifteen minutes after the unrest had begun; we were ready to put it to rest. We summoned my boss to explain and, if necessary, show him what we found. While we were testing, the part of the program in question had not been enabled. In short, we never got a chance to test it because in the two weeks we tested, that functionality was never implemented. We didn't miss it! It didn't exist in the builds we had.

© Phillip Bailey - All Rights Reserved

My boss, feeling much better now, called the producer back to explain the situation. As it turned out, her final report about the project and a "team-of- 2" vs. a "group-of-12" had one additional item to be added. Not only did we outperform them, they also missed a bug that we never had the chance to test. A bug that resulted in the gaming giant recalling 800,000 units.

Take two things away from this story:

1. A scalpel is an absolutely fine tool when applied correctly
2. Document Everything!! (you may have encountered this advice somewhere before)

© Phillip Bailey - All Rights Reserved

Challenge #2: Ignorance About QA

You live and learn. At any rate, you live.
~ Douglas Adams

Many people do not understand what QA does or how we do it. Testing is thought to be just "common sense" and there is much belief that anyone can do it. Although it is in your best interest to keep your testing as simple as possible, as you have learned throughout this book, "common sense" is only the beginning.

Many QA departments do not deliver their results in a transparent manner, thus keeping other departments (unintentionally) in the dark. This creates and perpetuates the ignorance other departments have about what QA is and does. This too can be changed. How?

When asked about your methodology regarding the testing you perform, explain as simply and clearly as possible the reasons for your testing, the steps you will (and do) take to complete the testing, and what information your testing will yield. Then deliver a report that shows this.

Reports to summarize your testing should show:

- **What you did**
- **What the outcome was**
- **What that outcome means**

In this way you will help others understand what they want and need to know about what you do and will help assure them that you do your job professionally by delivering quality results. Even if others have no desire to understand the details of what you do, by delivering your results with an explanation will help them make the decisions that need to be made when prioritizing the issues affecting the project.

They will be able to make those decisions in an educated and informed manner. This is necessary to build a trust that QA is doing the right things and adding value to the project.

© Phillip Bailey - All Rights Reserved

Another area of value that a true Quality Assurance Professional brings to the table is the ability and desire to ask the right questions. Since I have covered this point in some detail previously, you should know how and why this skill is helpful. Unfortunately, this is an area of expertise that real Quality Assurance Professionals offer that is often overlooked by many companies.

It is very inexpensive to include QA during the planning and design phases of a project. In fact, it would save companies a great deal of expense in the long run if they made a policy of doing so. Why? Because QA would ask the questions that others overlook. This means that potential bugs could be averted in the design phase, before a line of code is ever written. How does QA do this? By mastering the skill of being able to ask the right "what if..?" questions.

Where else can this skill be applied preventatively? In documentation review. No matter the depth or completeness of the documentation; whether it is single-sentence use cases or highly detailed marketing and production requirements, having QA review documentation before the project is being coded can save a company a great deal of money.

Why is that? Because a **trained Quality Assurance Professional can identify bugs in the documentation**. A bug that is found in the documentation is *much* cheaper to fix than a bug in the code.

An elite-level Quality Assurance Professional has developed the skill to be able to prevent bugs from ever being coded. They can point out where the documentation is lacking, in conflict, or just wrong or impossible. When allowed to help in this way, QA increases the odds that a project will stay on-track and get to market on time because proper planning was invested in initially.

Why don't companies do this? Honestly, I'm not certain. It would seem to be a no-brainer. But I have witnessed a few reasons that might dissuade companies from engaging in this preventative behavior.

The company does not trust, or perhaps respect, QA. This goes back to Challenge #1. There are many valid reasons for this lack of trust and respect. If I didn't trust someone, I would not include them during the planning phase of *my* project.

© Phillip Bailey - All Rights Reserved

Also, in their haste to get a product to market, many companies sacrifice spending adequate time on planning. Thinking that they have planned "enough" a company will try to decrease their time to market by hurrying up and getting started. Their belief seems to be that the sooner they start, the sooner they will finish.

If they spent the proper resources planning their project, they would show a decrease in their time to market, a smoother and more measureable project pace, higher quality product, and a more fulfilled workforce. So again, why don't they do this?

To be fair, there are companies that do, indeed, execute their projects this way. They are, however, the exception…not the rule.

Mostly companies don't indulge in this behavior because they don't understand how QA can help. They are ignorant of what QA brings to the table other than the ability to test.

As you have seen, actual testing is only part of what a Software Quality Assurance Professional does. And this book is focused on you, the tester, the person that spends the most time testing. Once you are leading a team of testers, you will have even more duties to attend to…but that is for a later book.

The Challenge that you must overcome in this scenario is that the company won't know all of the valuable skills you bring to the game. You must develop your skills and then find or create opportunities to show how those skills can be most effectively utilized. That is your Challenge. They don't know what you can do, so you will have to educate them.

© Phillip Bailey - All Rights Reserved

Challenge #3: Position of QA

The real reason for the use of pressure and overtime may be to make everyone look better when the project fails.
-DeMarco, The Deadline

Because of where testing falls in the timeline of most projects, any delay to the project will immediately affect QA. Although this affect can be less severe in an Agile Development environment, testing still happens *after* development. This means that any delay in the project will have a ripple effect that can play havoc with your testing schedule.

These scheduling changes do not affect the date that the company *wants* the product to ship, that date remains unchanged. What they do affect is the amount of time allotted and available to complete the as-yet-unchanged effort required for testing.

Here is why:

- The testing required to fully test a product is estimated by QA – and agreed upon by the project team early in the process
- The effort is then translated into an amount of time (total estimated tester hours)
- Based on the number of testers available, a schedule is created detailing what testing will be executed, when, and by whom
- This estimate is integrated into the project schedule

Since QA, when properly done, measures most (if not all) of their work, **these estimates are based on hard data**. They are based on results culled from having executed these types of tests time and again. So, they are not mere guesses but are known, measured, quantified results. QA knows how long it will take to test something because it has been done before and QA has those results!

© Phillip Bailey - All Rights Reserved

At this point it all looks good and everyone is happy and friendly. Then something happens to the project. One or more of the following scenarios may happen:

- **Features are added** to the product that were not originally in the schedule
- A feature is **more difficult to implement** than initially thought – so the development team delivers a feature later than expected
- A developer is out sick and so the **delivery of a build** to QA **is delayed**

Any one of these scenarios mean that QA does not receive the build with the agreed upon features when it was agreed that they would receive it. Again, this does not change the shipping date – only the amount of time available to test.

Can you see why any delay to the project will definitely affect your planned testing? This doesn't even account for any inaccurate estimation given by QA about a feature that will take longer to test than originally thought.

Because the testing of a product is positioned between the development of features and the ship date, and since once a feature is coded it is often assumed it will be a shippable feature, once QA gets a build, the clock is running. The company does not want to change the date they promised a product would ship, so any delay must be made up during the testing phase.

Here are **a few simple actions that you can take to help minimize the impact** of these scenarios on QA:

- **Review all specifications and documentation** for a project as early and thoroughly as possible
- When reviewing the specs and docs, **point out any and all errors, omissions, conflicting statements, unclear behavior, etc.**
    - For the company, this is the least expensive place to find a bug
    - For you, it alerts others to issues that need to be addressed *before* the testing clock begins to tick
    - This also shows that you are paying attention to every detail of the project from the outset and will help you be viewed as a real professional
- **Plan ahead for any testing** that appears it may be time-intensive
    - What efficiencies can you implement to reduce the time this testing will take?
    - Are there tests that you can run in parallel to complete your testing sooner?
- **Be prepared to work overtime** when the testing schedule gets compressed

© Phillip Bailey - All Rights Reserved

None of this is to say that these effects are unavoidable! On the contrary, they certainly are if the proper steps are taken prior to beginning a project. If the project is planned with accurate forethought, these affects will either not occur or can be vastly minimized.

I once worked for a QA Manager who believed (rightly, in my opinion) that the need for QA to work overtime was almost always due to some other part of the project team making an error. Somewhere along the line, someone did not accurately estimate the complexity of implementing a feature, or allowed additional features to be added late in the project. So **we took steps every day to minimize the impact of these potential scenarios:**

- Always looking for how to improve our process and procedures
- Always searching for a way to test more efficiently without any loss in thorough test coverage
- Always alerting the project team if there was a potential scheduling issue looming

And some companies do things a different way.

I once worked for a company that had a project that continued to experience development delays. What they decided to do to account for these delays was to have QA work overtime. This is not a rare occurrence, but this particular situation amazed me.

They had that QA team of 20 testers work 15 hour days, 6 days a week, for 3 months! Think about that. Have you ever worked a schedule like that? I've seen it and it is NOT pretty.

I have never seen so many people look like the walking dead at a workplace ever! To their credit, the product did eventually ship…I think it was on time. Of course, there were a few bugs when it shipped – are you surprised?

In order to be a truly effective tester, you must be attentive to every detail! If you work the hours that those testers did, there is no way you will be as aware of what you are doing as you should be.

© Phillip Bailey - All Rights Reserved

This has many detrimental effects to the project (not to mention on the budget):

- The most obvious outcome is that you will miss bugs
- When you find a bug, you have a greater chance of entering it as a duplicate
- When you enter a bug, you have a greater chance of omitting crucial information
- When regressing bug fixes, you will not be as thorough
- You will enter a greater number of issues that are not truly bugs
- And the bugs that you do enter will not be as thoroughly vetted out as they should be

I am *not* making this up. This is my experience of working with testers who cannot pay the proper attention to what they are doing.

Companies ship their product, but not without a high cost both to their budget and to the quality of their products. The consumer receives lower-quality product, the company's brand and reputation are negatively affected, and the project team is highly motivated to leave for another job at their earliest opportunity.

Take proactive measures to build in as many testing efficiencies as possible. You will need the time savings eventually. And if overtime is not required, you can show off how quickly you can zip through a test suite without sacrificing thorough test coverage.

© Phillip Bailey - All Rights Reserved

Challenge #4: Lack of Tools

If written directions alone would suffice, libraries wouldn't need to have the rest of the universities attached.
~ Judith Martin, "Miss Manners" columnist and author

The test plans, test suites, and test cases that are created to ensure thorough project coverage are supposed to be based on specifications and other documentation created by marketing, product development, and engineering. These documents are the foundation for QA to determine what to test and how to test it, as well as what the desired outcome of each test should be. Without this documentation, it is impossible for testing to guarantee complete, comprehensive coverage.

Unfortunately, it is common that the documentation delivered to QA is either incomplete or non-existent. When this is the case, QA has no way of tracing the product requirements to their test cases. This heightens the probability that testing which should occur will not or that when the product does not function as desired, QA is not aware of it. It will also result in issues being logged that will later be claimed to not be bugs – if the desired behavior is unclear, QA is left to make their best guess as to what the behavior should be.

In this all-too-common scenario, there are still **actions that you can take to minimize the gaps in your test coverage:**

- **Have the entire project team review all test plans, test suites, and test cases.** This will not guarantee complete coverage, but it will increase the chances that inaccurate tests become identified and modified *before* you spend time executing them. This also helps spread the accountability for test coverage across the whole team because they have invested time and effort into vetting out your tests
- **Spend time with the developers** (engineers, product managers, etc.) to understand the desired functionality of the product
- **When in doubt, LOG IT!**
    - As unfortunate as it is to log an issue that is determined to not be a bug, it is better to have a logged issue not be a bug than a bug not be logged

© Phillip Bailey - All Rights Reserved

Find and review all of the specs and docs that you can that relate to your project, as peripheral as they may be. You will be surprised what nuggets of information you can uncover that no one thought to tell you about. This will give you more in-depth coverage in your test cases and your testing.

Another **common tool missing from the QA arsenal is a real bugbase**. Many companies consider the expense of a state-of-the-art defect tracking system to be a luxury. What this means to you is that you may have to work on a free, or a cheap, or an old-slow-featureless bugbase that will cost the company more in wasted time and effort in a single year or two than a real system would cost them in the first place.

Not bitter, just aware of how much, otherwise preventable, wasted effort is expended in the high-tech sector.

For you this means that the sooner you can master your bugbase, no matter how unwieldy it is, the sooner you can minimize the adverse affect this shortsightedness will have on your daily work life. **Master your bugbase and you will be a formidable opponent for anyone and a great ally for all.**

© Phillip Bailey - All Rights Reserved

Challenge #5: Miniscule Margin for Error

Be wiser than other people, if you can, but do not tell them so.
~ Lord Chesterfield

If at first you don't succeed, you must be a programmer.
~ Internet Humor

This last Challenge is one of the biggest and ever-present that you will have the privilege of overcoming as a Quality Assurance Professional. The reason that I have spent so much effort in this book making the point that you must attend to each and every detail, is that you have so little margin for error.

Each of the principles that I have outlined in this book has been explained to you for a reason. Each skill I have told you to master was given to you so that you could excel. Each list of Steps to take, Keys to learn, and Mistakes to avoid were presented in their most simplistic form so that they would become second nature to you as soon as possible in your real-world everyday work environment.

The reason for this need for speed? Because your work needs to be flawless (or as close to it as possible) as soon as possible.

Why must your work be flawless? Because that is what is expected of a true Quality Assurance Professional. I have found the following to be true:

- **When a developer makes a mistake, it is a bug**
- **When QA makes a mistake, it is a recall**, a stock rebalance, a halt of the manufacturing line, etc. A QA mistake can cost tens of thousands, hundreds of thousands, or even millions of dollars to correct.

This means that the margin for error in QA is less that for many other teams. QA is the last line of defense. QA ensures the quality of the work of other teams. There is no other department (during the development process) that exists to verify the work of QA. This is something that QA must do itself.

This is why it is imperative that you are attentive to every detail, why it is crucial to document all of your work. You must know that you are doing the right things and doing them in the right way. Document all of it, review what you have documented, and measure your results. This is how you will ensure that the consumer will get the product that has been created for them and that it really will meet their needs.

© Phillip Bailey - All Rights Reserved

You must be the consumer's most ardent advocate, their most passionate advocate, and you must know that the work you have done ensures they receive the best efforts of your project team! This you can do by planning your work, documenting the work you perform, reviewing your effort and results to ensure you are progressing as planned, and squawk as loudly as possible when the product is not meeting expectations. This is where you will shine as a true Quality Professional!

Knowing that you have so little margin for error can be intimidating. Alternatively, you can **accept the challenge and strive for excellence**! Working in ignorance, you would not be as effective. Knowing the challenges you will face allows you to plan your work most effectively.

Armed with the knowledge that you are the one to bear the Standard of Quality for the consumer, the one to champion the vibrancy of your company's brand, and the one to always seek improvement in your process, you can challenge the level of excellence accepted in your company. You can be the one to lead the charge to help your company raise the level of quality of their products.

By documenting all the work that you do, you will have a measurable trail of what effort is necessary to complete a project. You will then be able to compare that measurement against the effort involved in fixing the product you have shipped. By showing the effort involved and forecasting what the effort would have been to complete the project the right way the first time, you will be able to help influence a change in project planning and execution.

And there they are…

The Top 5 Challenges of Being a QA Tester

1. **Lack of Respect**
2. **Ignorance of QA**
3. **Position of QA**
4. **Lack of Tools**
5. **Miniscule Margin for Error**

Now that you know what your biggest challenges will be, you can plan accordingly and begin to manage your own expectations. You will be able to begin (or continue) your career armed with the knowledge of what you will face on a daily basis and how to navigate those challenging waters.

© Phillip Bailey - All Rights Reserved

If you find that you have the additional Challenge of a boss that doesn't understand your issues or a company that doesn't care about the quality of the products it ships, then you should consider moving on. There are employers that place a premium on quality. There are many bosses in the workforce that will help you create change for the better, mentor you so that you can improve, and teach you the things you want to learn.

Quality Assurance has many obstacles that can impede the ease with which it can be executed. But they are all effectible to a greater or lesser extent. It is up to you to create the necessary change!

You have been forewarned. But you have also been given the tools to thrive when confronted by the daily Challenges that will confront you.

© Phillip Bailey - All Rights Reserved

Chapter #15: Getting a Raise

The trouble with doing something right the first time is that nobody appreciates how difficult it was.
~ Steven Wright

We have already covered most of what you will need to know in order to get a raise at most companies. By attending to the following details, you will have the ammunition you will need to make a convincing case for yourself.

The key actions you will take to gather your ammunition:

- **Document** your work
- **Report** your work
- **Save** your reports
- **Master** your bugbase
- Turn your bugbase queries into **charts and graphs that show your work**

With all of that information at your disposal, you should have more than enough information to make a convincing case for yourself. So what else do you need to know?

If you are an FTE (Full-Time Employee), the first thing that you should find out is when you are eligible for a review and a raise. Some companies follow a policy of conducting reviews once each year. Some companies do it more often. What you need to know is when *your* company holds its reviews.

If you are a temporary employee or a contract worker, find out what needs to happen for you to be eligible for a raise. Every company has a policy. Either you will be eligible at a certain time or you will be eligible at the discretion of your QA Lead or Manager. Find out what the criteria is.

Once you have determined *when* and *how* you are eligible to get a raise, you need to have a meeting with your QA Lead or Manager. Whoever has the authority to make the call regarding how much money the company will pay you – that is the person with whom you need to meet.

© Phillip Bailey - All Rights Reserved

It is up to you to ensure that the company will reward you for the work that you do. If given the choice, most companies will pay you as little as possible while you produce as great a benefit as they can dream up. Not all companies, but most. So it is up to you to meet with the person who can has control over how much you get paid and find out what it will take to get a raise.

When you meet with them, you need to establish the criteria by which you will be judged. You must find out what it will take for you to get a raise. What goals do you need to meet? What competencies will you have to master? Work with your Lead or Manager to set goals for yourself that will produce tangible benefits for the company and your team as well as show that you should be given a raise and a new set of goals should be established.

These goals should be **S.M.A.R.T. goals.** S.M.A.R.T. stands for:

- **S**pecific
- **M**easurable
- **A**ttainable
- **R**ealistic
- **T**imely

In your meeting, you will come to an agreement about what S.M.A.R.T. goals need to be reached for you to move to the next set of goals. If your Lead or Manager is not experienced in doing this, then you will be helping them learn how to accurately evaluate you and your work. Have suggestions for goals that would both benefit the company and allow you to continue progressing in your career.

With **defined S.M.A.R.T. goals** in hand, your daily work will have specific focus that you will be able to measure. You will document and report your work and you will be able to measure your progress toward the achievement of the goals that you established with your boss.

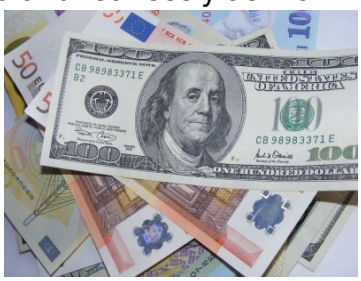
Once the time has come for your review – either due to company policy or because you have met the criteria defined in your S.M.A.R.T. goals – you should have all the information you need to make a case for yourself.

Prepare yourself before you have your review. Proof your reports. Check your data to ensure its accuracy. Attack your proposal as though you were

© Phillip Bailey - All Rights Reserved

someone trying to NOT give you a raise. What holes in your data does this uncover?

If you have done all of your preparation as detailed in this book, you should be ready and able to have your review and get yourself a raise. You should be fully prepared to answer any questions that are presented to you. You have the data to prove your case, and you have done all of the work to gather and present that data. Be proud of what you have accomplished. Go and fearlessly ask for what you deserve!



And remember: If they don't want to pay you…someone else out there will!

You have all the information you need to present your case. You can answer every question point-by-point. You have the knowledge. You have the ability. You have the tools to make of your life whatever you desire.

© Phillip Bailey - All Rights Reserved

Chapter #16: The Job Hunt

Live out of your imagination, not your history.
~ Stephen Covey

Ok, so now you have all of this information. You are filled with the knowledge you need to excel as a Software Quality Assurance Professional. You know all of the actions you will need to take to perform as an elite-level Quality Assurance Tester. But what if you don't have a place to perform?

Are you all revved up with no place to go? Then let's get you positioned to make a difference in this high technology world. Let's find you a place to strut your stuff, a team to join, a company to impress.

Before you begin your search, there are two questions that you will need to consider in order to target your search most effectively. They are questions that only you can answer. The answers to these questions may change over time, so come back to them periodically and assess the answers you give.

The two questions that you will need to answer are:

1. *Where* do you want to make a difference?
2. *How* do you want to make a difference?

Let me break each of those questions down so that you can consider them accurately.

Twenty years from now you will be more disappointed by the things you didn't do than by the ones you did. So throw off the bowlines, sail away from the safe harbor. Catch the trade winds in your sails. Explore. Dream. Discover.
~ Mark Twain

© Phillip Bailey - All Rights Reserved

Where **do you want to make a difference?**

Your answer to this question will include many different factors. Below is a list of questions to ask yourself. I hope that they help you create an understanding of the type of job you should be looking for.

- **In what geographic location do you want to work?**
  - What country do you want to work in?
  - Do you speak more than one language? Does this provide another option for a country you can work in?
  - Do you want to work in Europe or in Japan?
  - Do you want to work in the United States? East Coast or West Coast?
  - In the San Francisco Bay Area or in Silicon Valley? Seattle, New York, or Florida?

  - Would you prefer to be located in Toronto, Canada?
  - What about Australia? Or do you want to try Ireland?
- **Where is the business world do you want to work?**
  - Do you want to work for a Fortune 500 company?
  - Do you want to test games?
  - Do you want to work for an established company like GE, or would you prefer to spend your time working at a start-up?
  - Do you want to work for a Non-Profit company?
  - Do you want to work for a Non-Governmental Organization (NGO)?
  - Do you want to work in the Financial or Medical sector?
  - Do you want to join Microsoft or work with the next upcoming Google?
  - Would you prefer to be part of a 100 employee QA department or are you looking for a smaller team?

Only you can answer the questions about where in the world and the business world you want to be. Each person has different criteria and ambitions, so take inventory of YOU. Discover where YOU want to make your mark.

Remember that there are all kinds of ways to get where you want to go. There are opportunities and resources everywhere.

Do you need to learn a language so that you can work in the country of your choice? If so, do you know anyone that can help teach you the language?

© Phillip Bailey - All Rights Reserved

Do you know anyone that can introduce you to someone that can help you learn the language?

If you want to work in the United States, do you need H1 Visa help? There are always technology companies hiring qualified personnel to fill their many vacant positions.

If you need H1 Visa help, check out this program. It should be able to help you. They claim (and do appear) to be "one-stop" shopping.

This is from their website:
"H1B Jobs and Visa Sponsorship - to get an H1B visa
Professional services and assistance to speed up the process and make it easier.

- A decade of expertise in the H1B visa employment and sponsorship market.
- A decade of helping people find and secure H1B jobs and visa sponsorship.
- A decade of helping thousands of people start a new and exciting life in the USA.
- H1 Base services and solutions are made available to people of all national origin and citizenship."

© Phillip Bailey - All Rights Reserved

How **do you want to make a difference?**

We all have different strengths and weaknesses, unique motivations, and desires we want to fulfill. We all have our own way to contribute to the world – how do you want to make a difference?

- **In what arena do you want to contend?**
  - Do you want to master all of the automated tools used in Software Quality Assurance testing?
  - Is there a specific programming language or technology that you would like to dedicate your career to testing?
  - Do you want to be a contract-for-hire consultant?
  - Do you want to be a tester only? Or do you want to lead a team?
  - Do you want to focus your career on the managerial path?
  - Do you want to be the white-box engineering expert?
  - Do you want to become a leader?
  - Do you want to become a trainer? Do you want to eventually teach Quality Assurance?
  - Would you prefer to test cutting-edge database breakthroughs or the latest games?
  - Would you prefer to test consumer electronics or position yourself to test Web 3.0?

Take the answers to the above questions into consideration as you target your job search. Look for opportunities that will move you in the direction you want to go.

As you gain experience, **return to these questions** and review them. Have your answers changed? Based on what you have learned so far, are there other opportunities that you want to pursue? Do you want to change industries that you test in? Is there a new type of technology that you want to pursue?

Armed with answers to the question above, you are ready to find a position that will suit your needs. There are different methods that you can take to find employment in the high-tech field. No matter what technique suits you the best, there are some specific actions you can take that will help you find a position where you will succeed.

© Phillip Bailey - All Rights Reserved

Prepare Your Resume

The first thing you need in order to apply for the many job opportunities in your future is a resume. Without a resume, most companies probably won't even expend any thought to consider you.

You will need a resume. No matter how much work experience you have; a lot, a little, or even none at all – you need a resume. Do you know how to write one?

There are all kinds of free advice on the internet about this as well as plenty of professionals that will charge you upwards of $300 just to get you a working resume draft. A great resume can really help you, so you may consider doing the research and finding out how to create one.

There are two resources that I use for resume information. The first is from **a great website on how to write anything.** Read what a professional copywriter with decades of experience has to say.

If you want a simple "here's-how-to-put-your-resume-together" system, then **take a look at this!** Jimmy Sweeney offers a money back guarantee on all his products.

His **resume creator** is a great resource if you're not sure what to write. If you don't have a lot of experience or just want a simple step-by-step plan and helpful template to follow, take a look and see what you think. All the help you need is there…just choose the best fit for you.

Take care that you craft the best resume that you can. A solid resume will serve you well. It is one of two tools that you will use to apply for your testing positions. Take this step seriously if you want to be considered and paid as a professional.

The better your resume, the better your chances of landing the job interviews that lead to the jobs you want – not the jobs to have to take because you have no choice. Take control of your career – build a resume that is worthy of you!

Work to understand how to build your resume. Learn what a quality resume that induces prospective employers to call you looks like. Developing the ability to put together an eye-catching resume time after time is a skill that will pay off tenfold throughout your professional life!

© Phillip Bailey - All Rights Reserved

If you have an engineering background and want to get into Software Quality Assurance, you should make sure your resume is configured appropriately. Take a tour over at **Best Engineering Resumes** and see if what they offer is a fit for you. I have a non-engineering background, so they are not able to really help me. But if you have a background in engineering, they should be able to give you the expertise you need.

© Phillip Bailey - All Rights Reserved

Your Cover Letter

If you apply to jobs with your resume and a cover letter, keep something in mind:

Most applicants write BORING cover letters

The person in a position to hire you will have to weed through hundreds of applications. They will read countless cover letters that begin with "I am a motivated self-starter looking to secure a position with your company…" or "I am the best candidate for your open position because…" or "Hello, my name is <name>. I have a degree in <whatever> and want to get a job in <industry>…"

These people mean well. They are following the instructions and templates in the books they have read, the websites they have studied, and/or the advice of their school placement counselor. What does not seem to be taken into consideration is the level of patience that the reader of their cover letter will have.

I have read through hundreds and thousands of these letters over the years. It is a part of my job that I least enjoy. These letters may as well read "blah…blah…blah". They are incredibly boring. The applicant usually spends the majority of their correspondence writing about themselves – this is a mistake.

Just as you wouldn't buy a product from a company whose commercials focused solely on how great the company is, why would an employer want to hire such a self-involved, egomaniacal, narcissist?

Company X Commercial:
"We are such an amazing company! Everyone wants to be us. Look at just how wonderful we are and bask in our glow. Send us money!"

Uh, no…

Applicant X Cover Letter:
"I am so wonderful. Attached is a list of all that I have achieved. Look at just how special I am. Ask me nicely to join your company and I might consider it. Hire me!"

Uh, again…no…

© Phillip Bailey - All Rights Reserved

Your resume is already a list of education and experience that will help the company decide whether or not your skills may meet their needs. You don't need to send a bland, boring, cover letter that reads just like everyone else's.

In your cover letter should explain how your skills would be appropriate for the position that you are applying for. Not just: "I have <skill X> and you are asking for people experienced with <skill X>." You need to show how your skills would fit. **What benefit do you bring?** There are probably several people applying for the position that have experience in <skill X>. Many may even have more experience with it than you do.

So how do you separate yourself? Explain, describe, and detail *how* you would make a difference. What impact will you produce? Tell the reader how your unique background, skill set, and temperament would be helpful to *them*. That's what is important to the company. How will you provide value to the company – to *them?*

It's always about them. Differentiate yourself by showing how your uniqueness will provide value and impact to the company. If they decide they don't want that value, you have at least given them the chance to make the right decision.

One other note about your cover letter: In my opinion, it is ok to use humor. Don't be crude, but it is alright to be funny. If you can tailor your cover letter to the company with a humorous story that is applicable to the industry or product or service that the company is in, you will stand out. They may not choose you, but you will stand out.

The funniest cover letter that I ever read was from a guy trying to get into the software testing industry. He had no experience. Zero…none…zilch. His only real work experience was working retail at a comic book store. Not the best background since I needed to fill the opening as quickly as possible with a functional tester.

He had no experience, but his cover letter was entertaining. After reading hundreds of the previously described, torturous, pointless compositions, when I came across his letter, it was like an oasis. He explained that although he did not have experience testing, he had been playing video games since he was very young.

© Phillip Bailey - All Rights Reserved

He told a story of how his family had chipped in and bought a game console as a birthday present for his 89-year-old grandmother who could no longer get out on her own. She had been a golf enthusiast for most of her adult life and due to multiple hip surgeries, was no longer able to go out and play golf herself anymore. So the family decided to buy her a state-of-the-art gaming console and a golf game.

Well, it turned out that Grandma didn't think as highly of video golf as she had of real golf. In no time, she had given up on the gaming console and given it to her "favorite" grandson, my applicant. Since that day, he explained, he had been an avid gamer in addition to being a masterful video golf player.

After reading through his cover letter and resume initially, I didn't put him in the "for sure" pile, but he did stay on my mind. After I completed reviewing all of the applications, I decided that his was one of the five I would call to investigate further.

What I found out in the interview with him was that he was very bright, learned quickly, had fantastic communication skills, and still had absolutely NO EXPERIENCE. I hired him. He had the right attitude and enthusiasm. Skills I can teach. Attitude? That's on you.

I hired this previously inexperienced tester and never regretted it. He learned how to test and report quickly. He was a great communicator and was fun to work with.

With his lack of experience in testing, I would never have called him if his cover letter had not stood out. His cover letter differentiated him from the rest of the candidates. Make yours differentiate you.

Oh yeah…one last thing I learned about him: The story about his grandmother and the gaming console? He made the whole thing up.

If you want more specific help in writing a cover, then you should definitely look at the **Amazing Cover Letters system**. He guarantees his program. I have used his program to write cover letters myself and all I can say is that I wish I had this system 20 years ago. I would have had different opportunities, to be sure…

Also, don't forget to read what **Dr. Ugur Akinci writes about cover letter creation.** A clear perspective and a great deal of professional experience. Don't waste the opportunity to harness someone else's expertise.

© Phillip Bailey - All Rights Reserved

So, once you have your resume together and understand how to write a cover letter – are you ready to put the tools and info you have to use? Are you ready for the next step?

Almost…let's just go back and double check one more thing. Far too often, people put together a solid resume and start sending it out and posting it all over the internet.

This all seems well and good until they realize (usually after months of no phone calls)…their resume is FULL of typos!

© Phillip Bailey - All Rights Reserved

Proofread Your Resume!

This to me has always seemed very obvious, but I have been amazed at how many resumes I have received over the years that were full of misspellings, bad grammar, and error-laden syntax. And these people were applying for a position in Quality Assurance! Not only were they not hired, I never even called them.

The position is in Quality Assurance. If you cannot put forth the effort to create a resume that shows you have "dotted the i's and crossed the t's", I don't need to waste my time. You are the one who benefits from your resume. If you won't pay attention to your own work, why should I trust you with any work that would reflect on me?

So, **after you have written your resume, proofread it. Then spell check it.** Then proofread it again.

When you read your resume, you should be able to answer "yes" to the following questions:

- Are all of the words spelled correctly?
- Does each sentence a make sense?
- Do you communicate your points clearly?
- Have you communicated your points with brevity?

If you can answer each of the above questions with a "yes" then you have passed the first hurdle. Remember, in QA communication is paramount! Your resume is your first chance to impress a prospective employer with your communication skills. View this as an opportunity to use your resume to show off your communication skills. Provide value. Be succinct. Show attention to detail. Be a professional.

© Phillip Bailey - All Rights Reserved

Post Your Resume Online – EVERYWHERE!

You probably know by now that you should post your resume on the internet. Once you have proofread your resume and you are certain it is solid, post it. Post it everywhere. This means everywhere! The more places you have it posted, the greater chance it will be seen and the more opportunities will be presented to you.

Here are some sites to get you started on the right foot:

- **Yahoo Hotjobs**
- **Monster**
- **Career Builder**
- **Resume Bucket**
- **Linked In**
- Etc.

Make a list of everywhere you have it posted so that you can keep track of all the sites that are hosting it. Take note of sites that will need you to resubmit your resume every 30 or 90 days. With those sites, if you don't resubmit it, it will disappear from their site.

Another tip: Edit your online resumes every single day. Why? Because when a recruiting manager is searching for candidates, it is often easier for them to search for "new" resumes uploaded in the last day. If you edit your resume and re-upload it, you will fall into this search category. This will increase your odds of being seen. The edits you make don't have to be major overhauls. You can add a comma, delete a dash, change something minor – just be sure to do this every day. It will keep you more visible.

© Phillip Bailey - All Rights Reserved

Research Companies You Want To Work For

When you find a company that looks promising to you and you want to apply for a position they have open, great! But before you submit your resume and cover letter, **do some research**. Read all you can find about the company to gain an understanding about who they are, what they do, what product or service they provide. Armed with that information, tailor your resume so that it targets that one company. Include your observations about the company in your cover letter. The key is to do some research.

Companies like it when you have spent the effort to find out something about them. This shows initiative on your part, so dig in and find out what you can about the companies you apply to.

Even though this is a fairly standard activity, if you really want to work at a particular company, you will increase your chances if you spend the time to learn about them. As you uncover information about the company, you may find an opening where they could benefit from your skills – whether or not they have a job opening posted.

If you really want to work for a specific company, write to them and tell them! Do some research so that you can show you know what they are about and then write to the HR department. If you *really* want to work for the company, keep writing (you are communicating clearly and spell checking, right?). Ask what you can do to get an interview. Tell them what you have to offer and *show* them how much you want to work for them.

This doesn't guarantee that you will get a job with your dream company, but it will certainly increase your odds. Companies need employees with enthusiasm! They spend time and effort trying to get the employees they have excited about the company. Wouldn't they want to add an employee that is already excited? They should – as long as you can do the job.

© Phillip Bailey - All Rights Reserved

Prepare For Your Interview

Whether over the phone or in person, a company will want to speak with you before they make a final decision on whether or not to hire you. I am not going to attempt to summarize the thousands of recent books written on the subject of how to be a good interviewee – but I will give you a couple of tips that should increase the odds of you landing with a company that is a good fit for you.

But before we get to those tips, I do want to recommend two specific books that will prepare you better than any others I have ever read. Once you land a job interview, either on the phone or in person, you must be able to answer the questions better than your competition.

When I was writing the 2nd edition of this book, I sent out many inquiries looking for job interview resources that would be valuable to *you*. There were two responses in particular that truly stood out; two books written by real professionals that had exactly what I was looking for ***and more…***

The **Job Interview Success System by Bonnie Lowe** is a fantastic piece of work. A complete program that gets you prepared for what to do before an interview, during an interview, and after. ***Just a truly quality system that I can't recommend highly enough.***

Not only do you get a complete system, but **she offers unlimited email consultation for free** when you purchase her system! I have never seen that offered anywhere else. She is a real expert and a true professional.

If you want to control your future, give her risk-free offer a chance. I am willing to bet that you will be as impressed as I am! This system is worth its virtual weight in gold!

The other book is the most in-depth that I have ever read about answering job interview questions. **The Ultimate Guide to Job Interview Answers** is not some amateurish rehash of books written before it. This book is truly one of a kind. It stands alone as the most revealing job interview answer book I have ever read or even heard of.

© Phillip Bailey - All Rights Reserved

Bob Firestone gives you not only the questions and answers to ace your interviews, but he *explains the reasoning* behind the questions. This means that even if you are asked questions other than those covered in his book (as I have been) you will still be equipped with the tools and information to answer them like a pro.

Each of these books alone is incredibly valuable. But if you want to be prepared for ***any interview*** at ***any time*** and be able to ***answer any question***…***do yourself a favor and get them both***. If you are not sure they are worth it, please consider two key points:

1. They both offer money back guarantees (I wouldn't recommend them otherwise – even as amazing as both books are)
2. One question: How much is a career in the high-tech field worth to you?

That should get you started. You probably won't even understand or remember everything the first time through – I know I didn't. There is so much valuable information, that it took me a few read-throughs to catch everything…and there may yet be more.

Invest in yourself and your future. Get the books, read them, and do what they say. You will be interviewing like a pro in no time. And remember that interviewing, just like anything else, takes practice. The more of it you do, the better and more comfortable you will get.

And if you walk into an interview with an ace up your sleeve that none of your competition will have…get…**Wow! You're Hired!**

Take a look…you won't be sorry

Onto *my* tips and two cents…

You need to take preparing for an interview seriously. This entails researching the company you will interview with, deciding beforehand what you will wear, making copies of your resume to take with you to the interview, and preparing notes that you can reference during your interview.

We have already covered doing your research – in short, use the internet (at home or at a library) or go to a library and get a reference librarian to help you look up information on your chosen company – and making copies of your resume should be a straightforward activity.

© Phillip Bailey - All Rights Reserved

Let's look at the clothing you will wear at your interview. I promise this will be brief. Decide *before* you have to go to your interview. Some people leave this to the last minute and then discover that their shirt, pants, skirt, or dress has an unknown or forgotten stain on it. Uh, oh! Time for Plan B. So, lay out the clothing you will wear beforehand. Look it over and make sure it, too, is ready to interview.

When selecting your clothing, choose something appropriate. **You should look professional, but you should also look like YOU.** If your normal garb is a tank top and Bermuda shorts or a thong and halter top, then you should probably tone down your style a bit for your interviews. Dress professionally, but don't overdo it. *You* should be comfortable in *your* clothing. If the suit you wear to an interview is too large and you never wear it anywhere else, you will be uncomfortable in your interview and that will be obvious. Select your clothing appropriately.

Another action you can take to **prepare for your interview** is to **make some notes**. Now, I'm not suggesting that you write down just any old notes. You should make notes for yourself to reference when you are asked questions during the interview. Jot your notes down on the notepad that you will have with you in your interview (inside your portfolio, if at all possible). You will have this notepad so that you can take notes *during* the interview and to keep a list of questions that you will ask.

When the interviewer asks you to describe a situation or a scenario or asks you to explain how you overcame a particular type of hurdle, if you have notes you will be prepared. **Make notes that are a quick reference for you so that you don't have to remember everything.** You may be nervous during your interview, so if you have notes you can just reference them. This way you won't stumble and stammer when searching for the right answer. You will have notes you can look at and then tell your story.

© Phillip Bailey - All Rights Reserved

Prepare notes that will help you answer questions/scenarios like the ones below:

- Describe a challenge that you overcame
- Explain how you handle high-stress situations
- Describe how you interact with other people / Do you work better solo or on a team
- Tell me about a work-related conflict that you had and how you handled it
- (the ever-famous) Where do you see yourself in five years

The questions you will be asked during an interview cover a wide range of possibilities, but if you prepare yourself for the basics you should be able to handle any variations.

© Phillip Bailey - All Rights Reserved

When You Interview

When you sit down for an actual interview, there are many details to try and keep in mind and you have probably heard them all before:

- Have a firm handshake
- Sit up straight
- Maintain eye contact – but not too much
- Don't fidget
- Don't say "um…"
- Don't yawn
- Don't interrupt

Wow, this sounds a lot like grade school. Ok, so there are a million books on the market about how to be a good interviewee. Find them and read them. I am certain you will soon ace all of your interviews.

Just a couple of things that I would like to throw in for you. Try these out and I am sure you will enjoy your interviews a lot more.

Think Of Something Happy

Although you will probably be nervous going into your interview, there is something that you can do about it. Before you go into the office building or wherever your interview is going to be, take a few moments for yourself.

Sit down in as quiet a place as you can find. Settle and quiet yourself. Now, think of a time when you were the most happy. Think about when you first fell in love, or when you got a puppy, or when you went to the beach with your family, or whatever it is for you. Bring that thought to the front of your mind and focus on the feeling you had when it occurred. Feel the joy of the event. Experience it again. Hold that thought and hold that feeling. Sit quietly and feel the joy that the event brought up in you. Hold onto that joy. In the grand scheme of things, this is what is truly important. Feel that joy. Hold that joy. Feel grateful for that joy. Stay with that joy. Hold it and feel it.

When you are ready, get up and take that joy with you into your interview. It's just an interview. Your joy is what is important. Maybe by carrying that joy with you, you will make your interviewer's day a little brighter. Stay in the joy. **Go have fun in your interview.**

© Phillip Bailey - All Rights Reserved

Smile
Yeah, that simple. Smile. It will be much more difficult for you to be nervous or feel intimidated if you smile. People will be more drawn to you. I'm not making this up. Go and do some communication and neural programming research and see what you find. Smile.

You will feel better and the person interviewing you will be more at ease with you – this will make you appear to be a more favored candidate. It's the same principle as how listening to others talk about themselves has the effect of the talker viewing you as a good conversationalist. Smile pleasantly. Your interviewer will most likely feel much for comfortable speaking with you.

Be Enthusiastic
Don't sit there like a bump on a log. Don't treat this like some painful surgery that you have to endure without anesthesia. Be enthusiastic! **Bring some energy to the interview.** You have done some background work about the company and want to work there, right? So show your excitement. Show them what you will be like to work with. Be positive and solution-oriented – and do it with enthusiasm.

I have had the great displeasure to interview many, many applicants that have shown up and expect simply to answer my questions with their practiced responses. They want a job; otherwise they wouldn't have shown up. But other than being physically present, they don't bring anything to the interview. This is another way that you can differentiate yourself.

An enthusiastic applicant is more the exception than the rule. **Be that exception!** Bring some energy into the interview and be enthusiastic!

Be Interested
Pay attention to the questions that you are being asked. When your interviewer speaks about the company, really listen to what they are saying. Not just the words, but the meaning. Be interested in your interviewer. Listen to what they have to say. They will think you a better candidate if you listen with interest as they talk about their job and their company.

Show interest in the reason they have a job opening. What need are they really trying to fill? Be interested in why you are interviewing with the person sitting across from you. What role are they playing in this hiring process? Be interested in their role. Be interested about whether they want to be talking with you or not.

© Phillip Bailey - All Rights Reserved

All of this will help you to be more aware during your interview, will help you appear to be a better fit, and demonstrate your ability to attend to details (you are interviewing for a QA position, right?). And please remember to be genuinely interested. If you are not sincere, your interviewer will notice and you will look like a fraud. Be genuinely interested.

Ask Questions

At some point during your interview(s), you will be asked if you have any questions. Today's tip: **ALWAYS HAVE QUESTIONS.** This is your opportunity to learn about the company from a unique perspective. You have a company employee right in front of you – ask them about the company and they will give you their perspective.

If they are comfortable with you (because you have been smiling) and think you really want an answer to the question you have asked (because you have asked with enthusiasm) then they will tell you what they think. When they do, listen (be interested). You will be surprised how much an interviewer will open up to you once they feel comfortable.

This is a great opportunity to find out if you really want to work at that company. They may share their frustrations and challenges with you – if you make them comfortable and are interested.

This will give you valuable insight as you continue your interviews. It can also be a very quick leg-up when you get the job. You will already know what they are frustrated about and can attempt to alleviate their frustrations.

© Phillip Bailey - All Rights Reserved

There are many different questions that you can ask. Here are a few suggestions:

- Ask about the products or services – ask about details that are unclear to you after the research you have done
- Ask about the company's participation in community events
- Ask how your interviewer got involved with the company
- Ask what your interviewer likes about the company
- Ask what your interviewer finds most challenging about the company
- Ask about the biggest challenges your interviewer expects to encounter in the next six months or in the next year
- Ask how they plan to meet those challenges
- Ask about the plan for the department you are applying to – in what direction are they heading

Those are some ideas. What is most important here is that you do, indeed, have some questions. An applicant with no questions is an applicant that really isn't that interested. Simple…NEXT, PLEASE!

Even if you don't have any testing experience, you can make an impression with your **resume**, **cover letter**, and in the **interview process**. If you have the attitude, the drive and the enthusiasm, you can differentiate yourself from the other candidates. You must be yourself.

A Software Quality Assurance department is always in need of smart, enthusiastic, solution-oriented team members. People come to QA from all different previous experience; IT, developers, non-technology-related fields.

Even if you have no previous testing experience**, if you can learn, you can find a place in QA. Software Quality Assurance departments often have to handle 90% or more of their own IT.**

Since we spend the day breaking hardware and software, we have to be able to "reset" the lab into a usable state for the next day's testing. This means that there is usually a great deal of non-testing-specific tasks that QA must perform each day and each week.

If you have computer skills, you don't need to have paid testing experience to get into QA. The more operating systems, applications, and hardware you are familiar with, the better your chances of landing a job quickly. If you know how to use Windows, MacOS, Linux, Unix, or any other widely used

© Phillip Bailey - All Rights Reserved

operating system well, you should be able to get an entry-level job without any other experience.

In this book, I have taught you what you need to know to impress once you get the job. But even in your interview you can speak about what information you have already learned. If you are applying for an entry-level position, the lessons you have mastered by studying this book will have placed you far ahead of your competition. Tell your interviewer what you already know.

You have read the book and done the work – be proud of that. Have confidence in what you know. Let your interviewer know that you are armed with the fundamentals and are looking for a place to contribute with them. Tell them how you will make an impact with what you know.

Whatever you can do, or dream you can do, begin it. Boldness has genius, power and magic in it.
~ Johann Wolfgang von Goethe

© Phillip Bailey - All Rights Reserved

Job Hunt Resources

Job Interview Success System
Use This Simple, Step-by-Step System and You'll Ace Your Interview and Get the Job, Too! Use Powerful, Proven Strategies To Convince The Interviewers That You Are the Best Person for the Job! ***(It's Easier Than You Think!)***

The Ultimate Guide to Job Interview Answers
Read these Sample Answers, in less than 15 minutes you will learn how to:

- "Package & Spin" your work experience so it is a perfect fit for the job
- Increase your confidence, easily calm nerves or fear
- Be more likeable
- Use professional words and phrases that will impress them
- Position yourself as the BEST candidate for the job
- Tell them exactly what they want to hear — so you get hired!

Amazing Resume Creator
From the website: "Jimmy Sweeney has helped 19,946 job-seekers (at last count) in over 25 countries, land more quality job interviews and job offers in dozens of highly-competitive fields using his breakthrough, step-by-step resume formula."

Amazing Cover Letter Creator
From the website: "Stop writing cover letters the hard way! Here's how to get your phone ringing off the hook with more quality job interviews and job offers faster than you could ever imagine..."

Wow! You're Hired!
From the website: "Unleash My Amazing, Two-Page 'Secret Career Document' At The Beginning Of Your Very Next Job Interview And I Guarantee You Will Shoot Straight To The Top Of The 'Must-Hire' List AUTOMATICALLY..."

Visa Help
H1B Jobs and Visa Sponsorship - to get an H1B visa Professional services and assistance to speed up the process and make it easier.

Language Resources
Affordable and effective language software for all learning levels and styles

© Phillip Bailey - All Rights Reserved

Chapter 17: Want To Be A Game Tester?

You think you want to become a video game tester, huh? Nothing wrong with getting paid to play games, right? Are you sure? As rewarding and enjoyable as testing games *can be*, becoming a valuable video game tester entails more than you would probably think.

Before you go running off to become the world's highest paid Halo player, let's take a look at the reality of the job…

Can You Get Paid to Play Your *Favorite* Game?

Well, yes and no. The reality is that companies **DO** need gamers to play test their games before they are released. It ***is*** possible (if you know what you are doing) to land one of these positions. But they don't need to pay you to play a game that has already been released.

Once a game is already live, they really don't need you to test it anymore. Perhaps the next version of the game…the next iteration, next year's model…but not the one you have already played. So, although you probably can't play your favorite version of your favorite game, you may have the chance to play the next version – *the one none of your friends has seen yet!*

Before you go running out in search of a game testing career, **there are some things you should know**. The most important of which is:

Testing is NOT the same as gaming!!

There is just no way around this one. Many gamers think that they can just play a game, use their gaming skills to find some obvious defects, and that makes them a tester. Sorry, nope. Not even close.

Getting a job playing games is one thing, actually ***testing*** them is another.

© Phillip Bailey - All Rights Reserved

Can You Become a Game Tester?

Finding bugs and reporting them, pouring over every single possible path through the game, tracking every combination available to the user, and contributing to the overall quality of a shipped product is a much more disciplined endeavor than just sitting down to enjoy endless hours playing your favorite game.

You also may have to test a game or genre that you don't really like. While your testing brethren are fleshing out the latest nuances of the newest networked racing game, you may be verifying that Spongebob is wearing the correct outfit in each game scene you have been assigned the responsibility for.

You should also know beforehand that after playing development versions (read: unstable, incomplete, and crash-prone) of your favorite game for 4-6 months…or longer… you may never really enjoy playing that game again the way that you used to.

Does that help clear things up a little bit? Do you still want to become a game tester? If you do, let me prepare you even more.

Is It Your Dream?

If you really love playing video games, want to be a part of the industry, and just can't think of your life without video games…you have some options.

The first thing to do is to try and get paid playing games right now, right? Why not get the cash machine started and get yourself some experience?

As I detail on my site, **getting a job as a video game tester *is work*.** They don't just hand out these positions (unless you really know just the right people, and even then…). There is **A LOT** of competition for video game tester positions. You didn't actually think you were the only one that wanted to get paid to play games, did you?

The advantage that you have is the book you are reading right now. This book was not written specifically for game testers; it was written to help create and improve Software Quality Assurance Professionals. So that means you have in front of you detailed information about how to become a great tester!

© Phillip Bailey - All Rights Reserved

This means you have more information at your fingertips right now than most any other game tester (or game tester wannabe) in the world! Why? Because I know what it takes to be successful as a tester in the software industry – starting at day one! I have been one and I have trained them.

I have trained hundreds of QA Professionals! Although I can't guarantee that you will land your dream job tomorrow, if you master the information in this book, you will be able to craft the career you want in the software testing field.

In this book, I detail what you need to know and I tell you why so that you can judge for yourself; today, tomorrow, or five years from now. My record of work and the numerous QA training success stories should speak for themselves. But you don't just have to take my word for it. I am a real person – **you can view my professional profile at LinkedIn** and look at the real-world recommendations written by former colleagues of mine and see my body of work.

I have nothing to hide.

If being a game tester and creating the next generation of video games is you absolute dream, then harness that passion. Learn what is in this book, and take advantage of any resource that will help you advance your career goals (with integrity, of course).

If you persevere, you will succeed.

There are no traffic jams along the extra mile.
~ Roger Staubach

What does it take to become a game tester?

Persistence…

To start with, it takes drive. There is so much competition for these positions that you may very well need to be patient and persistent to land you first job. It may seem daunting, but if you *do* persist, you will find a position that is right for you.

You will have competition not only from other gamers and game testers, but from those what want to become game designers and programmers. With no experience themselves, they find it difficult to be considered for their dream position.

It is, therefore, very common for them to try to land a job with a game company in their QA department. Their belief is that once they have their foot in the door, they will be much more able to slide from "tester" to "developer". This does **NOT** make them good testers, just increases your competition for landing a job.

Fortunately for you, you are reading Software Tester: Elite – you will have the advantage because you will know more about testing, but writing, the Software Development Lifecycle, bug regression, bugbase usage, etc. They may know a computer language, but you will have a solid leg up on them when it comes to QA knowledge (and experience, if you practice).

QA Skills…

Contrary to popular belief (and so-called, "common knowledge") you do not have to posses great gaming skills to become a great tester. Let me say that another way: "Gaming skills do not make you a great, or even good, tester!"

There is nothing wrong with being a masterful video game player, but don't think that alone will make you a great tester. There is certainly a niche for those who are highly skilled gamers, but during video game creation and testing, it is not a high-demand niche.

What you really need are QA skills!

© Phillip Bailey - All Rights Reserved

What are QA skills? Know how to find a bug, how to enter a bug, how to master a bugbase – ANY bugbase, understanding the life of a bug, how to perform bug regression, how to comment in a bugbase, how to execute a post mortem, how to create process improvement, how to build trust with developers, how to communicating in an impactful manner, how to test, measure, and report. Those QA skills!

If you are a great gamer, good for you…just don't let that get in the way of becoming a great tester. Those that are not highly skilled game players will find bugs that the great gamer won't. Why? Because they will find parts of games more difficult and so will attempt solutions the great gamer won't.

If you have fantastic gaming skills, be sure to keep a perspective of the game you are playing. Remember that not everyone will be able to take out every enemy with a single kill shot; some (for fun) may opt for the suicide run, or drive a jeep over their enemies, or jump out of the plane just to see how long it takes to fall to the ground.

Games and how players have fun in them are approached very differently by different users. So, whether you are a great gamer, or just someone that has fun playing but isn't particularly accomplished…remember to ***test*** the game, play it from different perspectives not just by trying to "win" every time.

The other reason that great gaming skills are highly overrated in a tester is that when a game is in development, there are almost always cheats available to the tester. Whether it is a command line option to load a select level or character or feature set or a simple answer to the current puzzle…no gaming company is going to risk relying ONLY on their testers' gaming skills.

Adaptability…

Along with persistence and having the skills to do the job, you must be adaptable. Many testers begin their testing career on a per-project basis. This means that as soon as the project is over, so is your employment.

This is often true even if you are an incredibly impressive tester. The project is done and so they may "not have a position" for you until another project comes up. There is also a legal risk regarding temp employees that many companies deal with by not keeping any temp around for more than six months.

They may hire you again after some time has passed (usually 3-6 months), but they believe it is their best interest to end your contract first no matter how

valuable a tester you are. This allows them to avoid an otherwise costly liability. A liability that wouldn't be an issue if they actually ran their QA the "right way", but that's a subject for a different book.

Does that mean you can't have a QA career? Nope! I began my career as a per-project tester in the late 1990's and I'm still going strong today. The key was that the work I did so impressed the project manager (who I didn't even meet or communicate with while working on her project), that she requested me as a tester on her next project.

So my career as a QA tester began with a six week testing stint and then they sent me home. I was, I admit, a bit disappointed. But then a couple days later the phone rang. They wanted me back, but it would be another couple of weeks – would I be available. You betcha!

As soon as that project ended, I was given a little more time off. But by that time, the company I had been working for saw that I was a valuable testing commodity. So when their next project came along, I was back and in the testing saddle – on a project that was in NO WAY similar to the first two on which I had learned.

This per-project work went on for about a year. Each time they called, it was for a different type of project. I tested educational titles, create-your-own-adventure titles, websites, hardware compatibility, browser compatibility, action games, etc. Then when they changed offices, I helped run all the CAT-5 cable to set it up.

If you can adapt to the uncertain schedule and be able to effectively attack any project, you will quickly become a sought after asset.

Be persistent…develop your skills…learn to adapt

If you can do that, you can make it as a game tester, I have no doubt. It may not happen overnight, but if you want to write your own ticket and control your career, then put in the necessary work…and keep going. A dream is never lost until it is compromised…

© Phillip Bailey - All Rights Reserved

Where to Start?

One place you can begin is on my website in the **Jobs section**. I give you the basics of where to look to find gaming jobs.

I also went looking for resources that could give you better information. Now, not wanting to become a game tester any longer myself, I am not in the best position to evaluate whether these programs are the most effective or not. What they will do is give you information that I don't have.

I have not used these any of these resources myself to obtain a game testing job (as I said, I have no desire to return to game testing – and besides, with well over a decade of QA behind me, it would not be a fair assessment for me to say that if I can get a job with these resources that you can…we probably have slightly different resumes). But after searching the internet for resources to provide you additional help, here is what I have found.

Here are the top three "get a job gaming" programs that I have found. I searched the internet long and hard to find you some help, and these three resources appear to me to be the most legitimate and effective. After reading reviews, looking at their basic offerings, and freebies, these are the only three that look to me to be worth the money.

They cannot, of course, guarantee you a job. You have to learn from their programs and do the work yourself, but each of them give you information about where to look and also claim to keep an up-to-date list of testing opportunities in the gaming industry.

Remember there is a LOT of competition, but with the skills you can learn from this book, along with the help these resources can provide…you should be able to create an opportunity for yourself to join the video game testing world.

Video Game Job Resources…

Take a look at these resources and make your choice. Keep in mind that they each have a ***money back guarantee!!*** I wouldn't list them if they didn't. Give them a shot and if you don't think they provide the value that you were looking for or if they don't deliver on their claims – get your money back. No fuss, no muss…but give ***yourself*** the chance to find out and launch your own career in the gaming industry.

© Phillip Bailey - All Rights Reserved

Only4Gamers

The best reviewed of the programs I could find and the most insightful information about what they offer. Just a few of their selling points (from their website):

- An exclusive, daily-updated list of the absolute best testing jobs (adding 10-15 new jobs per week)
- A list of over 120 gaming companies who are constantly on the lookout for new testing talent
- 8 eCourses, with each course containing as much content as our competitors' entire offerings
- Their guarantee: *"If after 8 weeks you can't score a job playing video games, simply send us an e-mail and we will refund your money on the spot. It's s simple as that. No harm, no hard feelings. And we still part as friends."*

BetaGamingTesters

Begin with Beta testing and work to get hired in-house. This program is offered by a guy that really sounds like he's been there. Look at all the information that he offers and see if this is a good fit for you.

He also offers a 100% money back guarantee so if he doesn't deliver, you get a refund. Read through what he offers and think about whether his program is the one that is right for you.

GamerTestingGround

Specializing in helping you make money playing games as a Beta tester. Read the FAQ, the testimonials, and see for yourself.

This, of course, also offers a 100% money back guarantee. Look for yourself and see if this is the course for you.

© Phillip Bailey - All Rights Reserved

What If You Land a Play Testing Position?

If you are not able to get a job working "in-house" immediately, what are your other options? Well, you might be able to get a gig as a play tester. This is pretty much what most people are talking about when they say beta testing (for video games).

Play testing can be really fun. Early in my testing career, I got the opportunity to do quite a bit of it. it was always a welcome change from testing mouse hardware/software localization, another eCommerce website, or the next big database widget.

We had several companies that would contact us for play testing because we had developed a reputation for delivering outstanding feedback. Too many testers just play the game and give vague, unquantifiable opinions – we delivered hard data that companies valued and motivated them to return for more.

If you are able to land a play testing position, here are some things to know:

- It probably will be a short contract
    - Since the game is almost done, most companies just want a gamer/game tester evaluation for any final tweaks that they might make. This usually means that you will only play-test one version of the game or maybe two. This means that you won't get rich from one contract, but you can open doors if you seize the opportunity!
- To impress, you must deliver valuable information
    - If you want to make a good impression as a play tester you must show the company that you are worth the money. That's what it is all about for them. They are seeking valuable information from real gamers so that they know their game will be profitable when it ships (I have play tested more than one title that was never released – it is cheaper to eat failed development costs than to lose both development *and* marketing costs)

© Phillip Bailey - All Rights Reserved

What Should a Play Testing Report Contain?

First and foremost you should make sure that you address any specific questions or items that the game company asks about. They will tell you what specifically they want to know before you begin testing.

Make sure you are clear about what information they expect from you before you start (see Elite Skill #3: Managing Expectations). Once you know what to look for so that you can address their concerns, begin testing.

As you test you will want to keep some standard play testing report items in the front of your mind. These standards may or may not be requested by the company that hires you. But whether they ask for these specifically or not, here are the keys that I **always** reported and brought me return business (some situations call for additional information, but these are the standards):

- What works
- What's "cool"
- What's broken
- What's missing
- Appropriateness of difficulty
- All must include examples and "WHY"

What Works…

In this item, all you are doing is identifying the major features and options that are up and working as a player would expect. Since you may not get a manual, this is the way to give the company feedback about what you are seeing and if it is working in a way that makes sense to the player.

Are all of the features and options easily intuitive?

It also lets you list which all of the options and features that you have tested. A comprehensive list will help to *show* the company that you are being thorough ***and*** organized. This is a great way to open your report to the company (after your opening statement). This is your way of saying "here is what I saw that is working".

© Phillip Bailey - All Rights Reserved

What's "Cool"…
This is a straight "gamer's opinion" item. Here you want to detail what you think is fun about the game (if anything…we always hope there is something – unless it is a complete dog, you should be able to pick out *at least* one thing).

When playing the game, what feature or option or level or method of removing/eliminating/neutralizing your opponent do you find unique? What about the game do you find special, unique, new, better than before, better than other games, etc.?

This is your opportunity to *show* that you are an informed, experienced gaming *professional*. If you can detail your opinion of how the game you are testing is different or better than other games, you highlight your expertise and knowledge.

What's Broken…
This should be very straightforward: What's not working like you expect it to?

Broken can happen in either of two ways:

- Broken according to gaming "norms" – a feature not working the way a player would expect it to (and is also difficult or near impossible to adjust to)
- Broken according to game docs or technical specs (on the off chance that any are provided)

You want to detail very clearly what feature, option, variable, etc. is not working as it should. When you include this as a section in your report, you should create a bullet list for each item. Each bullet-listed issue should simply be a clear summary with a reference bug number.

You can include the bugs as an appendix at the end of your report. This is a format that I used successfully on several occasions. It looks great and shows how professional you are. I recommend this format unless you are given specific instructions to deliver your report in another manner or format.

© Phillip Bailey - All Rights Reserved

Here is a rough example of what each line item would look like:

Organized by Game (or Feature/Option) Section:

- The most egregious and important issues listed first – See Bug# 1
- Each item summarized into a clear one sentence description(when possible) – See Bug# 2

If you need help making sure that the bugs you write are as professional as possible, **you can submit them to me here. You can include a screen shot if that helps too. You can submit your bug and I'll send you feedback. If you bug is an appropriate example, it will even become its own webpage on my site.**

One word of caution about submitting bugs, however. Be careful what information you submit if you have signed an NDA – that's a Non-Disclosure Agreement. Often software companies have testers sign these so they can protect their intellectual property.

I you have signed an NDA, you can still submit your bugs. All you have to do is change any confidential information and make it generic. Chang the name of the game/program to "Game name" or "Program name" – that kind of thing.

Also, check my **Quality Assurance Forms and Templates page**. It might just have a template you can use for your testing reports.

What's Missing…

What features or options are not in the game that you would expect to be? Or what would make the game more interesting and fun? That's the kind of information you should be including in this part of your report.

Is there an option that is obviously not implemented? Is there a feature or enhancement that would really make the game much more enjoyable?

This can be anything…the more subtle and simple the better. Those things are usually easier and less risky to implement – and nothing major is going to be added at this stage of the game. If you want to show your gaming expertise and testing professionalism, figure out what can be added to easily enhance the game; what has been overlooked?

© Phillip Bailey - All Rights Reserved

I once reviewed a post-apocalyptic racing game that was a great deal of fun. It was almost ready for prime time, but there were just a couple things that didn't quite add up.

We reported that the soundtrack music that played on 8 of the 10 tracks was really quite annoying. Turned out we weren't the only ones who thought so. When the game was released it had 8 new soundtracks…

The other obvious pieces that were missing were two fundamentals of racing games:

1. Which car is where – in order…so the user would have some frame of reference
2. A rear-view mirror

Both items had been left out due to a producer that thought they made the game "too busy". The programmers had developed and included them but then were told to remove them.

After a thorough round of play testing, we submitted our reports. Both items were re-implemented in the finished game. Again it also turned out that we weren't the only ones to report they should be included.

Be an expert. Give your expert opinion. Know your games. If you deliver, you will be a tester that is in demand!

Appropriateness of Difficulty…

Since you are being asked to evaluate the game as a real-world gamer and consumer, you should be sure to offer your opinion of the difficulty of the game. Is it too hard, too easy, too fast, or too slow? Are there parts of the game that just bore you to death?

When you evaluate the difficulty and playability of a game, you must remember who the target market is. This is a key to being a valuable play tester and beta tester. If you know who the target market is, then you can use that perspective to expertly evaluate any program.

In play testing, the target market is very often – You! That's why they picked you out of the hat (or however they make their executive decisions) to test for them. There are times when you will not be the ideal consumer for the game you are testing. But no matter what the program is, remember to keep the target market

© Phillip Bailey - All Rights Reserved

in mind so that you test with the proper perspective. If you can do that, the feedback you provide will be far more valuable.

You Must Give Examples and Explain "WHY"…

As you compile your report and list all of the above items, remember that you are very often just stating your opinion. No matter how expert that opinion is, **it is not fact**, it is your opinion.

Yes, you have been asked to evaluate and to give your opinion, but it is still an opinion. That means that if you want to be understood, if you want your report to carry the weight of a professional opinion, you need to include examples and explain the "why" wherever possible.

Use specific in-game examples when citing any of the items above. Cite your example and then explain WHY. Why is something cool, or broken, or missing? What is the effect to the user? How does the example you are using affect playability?

This is the critical, valuable information that makes your opinion that of an expert. Any idiot can say a game is "fun" or "lame" but most are not able to describe why. If you can explain in a clear, concise and accurate manner *why* you have the opinion of the game that you do, you will be far more valuable. Your report will carry more weight because you are adding reasoning to your evaluation – this allows an educated evaluation of your report.

© Phillip Bailey - All Rights Reserved

Don't Wait to Gain Experience!

If you really are serious about becoming a professional game testing expert, don't wait for an invitation to the party. Crash the party or start one of your own!

You need to acquire as much experience as you can – and you might as well get started. You probably have games already at home that you can take a look at, but in a whole new light. Fire one up and see if you can play it from a different perspective.

Although any game you have at home has most probably been considered complete and ready for purchase (let's just assume that's how you got them), I'll bet they still have bugs. Most games ship with plenty – they're just not the most obvious or egregious ones…usually.

When you're ready for a new challenge, a game you haven't played before or a new kind of game, check out the resources below and I'm sure you'll find plenty of selection. Don't go back to the store and spend $50 or $60 dollars on the newest and latest just for practice. Find an affordable option to get experience with.

Here is a list of affordable options. Choose the ones that interest you the most and work from there:

- **Aces High WWII Interactive Game**
- **Bigpoint**
- **GameFly - Online Video Game Rentals**
- **GameHouse**
- **Gamemine**
- **GameStop**
- **Lord of the Rings**
- **Metaboli**
- **Microsoft Store**
- **Nick Arcade**
- **Nick Jr. Arcade**
- **PopCap Games**
- **Second Life**
- **Shockwave**
- **Yahoo Games**

© Phillip Bailey - All Rights Reserved

A couple more words about video game testing:
As you practice testing, remember to do your best to think about the game you are playing as though you have never seen it before (whether or not you have). As you practice to become a skilled game tester, you will need to develop this ability to see a game with "new eyes".

If you know of existing bugs or workarounds or cheats – block them out of your mind. Become the target audience and play the game anew.

When you are practicing, look for bugs that affect game play. **That is the sort of bug that you want to unearth and submit to show your testing acumen.** Although valid, if the only bugs you can find are "the graphic is a single pixel too far left", it's just not very impressive to a video game hiring manager.

It does show that you have a great eye for detail, but that is not the kind of bug that will get you hired usually. You will need to be able to find and identify bugs that allow the user to "cheat" or otherwise adversely affect the look and feel or playability of the game.

Look for bugs that allow you to heat (and don't submit bugs that are listed in a cheat book). If you are playing a game with a camera that "follows" the player, can you get the camera to "see" through walls? Does this allow you to gain information that would not otherwise be available?

Two other issues with cameras are the "impossible location" and the "exploding pixel". These are both issues that can greatly affect game play.

As you play a game that has a following camera, try to get the camera into a location where it shows nothing useful. Can you get the camera into a corner behind the character's head – so close that all you see is the hair follicles on the back of his/her scalp? Or is the developer's solution to keep the camera a minimum distance from the character – allowing you to put the camera into a room that you should not be able to access otherwise?

The "exploding pixel" is one of my personal favorites. If you're not familiar with this bug, let me explain…

What happens is that a pixel (or a few of them together) appears much larger than they should be on the screen. They cover all or most of the screen, making the UI useless and the game not playable.

© Phillip Bailey - All Rights Reserved

A few years ago I ran into this issue with a PS2 game that had been on the market for over a year. To create the bug required a cooperative, multi-player setup where one player was the Barbarian and the other player the Cleric. When the Cleric cast a specific spell (Spiritual Hammer, or something like that) the hammer would appear and begin dancing around. Within a couple of seconds, one (or several, it was very difficult to tell) pixel of the hammer would explode and take over the entire screen.

The result was one dead Barbarian and one dead Cleric. Unable to see, and thus strike, any of the enemy – dead quickly ensued. Only happened in multiplayer, coop and only with that specific character combination.

If you can get an exploding pixel to occur, take a screenshot! Then you've got the beginnings of a bug. Work your Reproducibility, etc. and **you will have a quality issue to submit.**

Use image capture software to get a pic of your bug. Here are a couple of free options:

- **CamStudio**
- **FRAPS**

When you are able to create, capture, and submit bugs like this, you will have no trouble showing a video game hiring manager just why they should hire you. You can build an online portfolio of bugs you have found in existing games – and I will help you make sure they are written like a professional!

If you really want to be a professional game tester and want to get into the business sooner than later, develop your skills and go get experience on your own. It shows dedication, initiative, and passion – traits that most employers will value.

Develop your craft and become a professional…

© Phillip Bailey - All Rights Reserved

Game Testing Resources (Employment Resources):

Only4Gamers
The best reviewed of the programs I could find and the most insightful information about what they offer. Just a few of their selling points (from their website):

- An exclusive, daily-updated list of the absolute best testing jobs (adding 10-15 new jobs per week)
- A list of over 120 gaming companies who are constantly on the lookout for new testing talent
- 8 eCourses, with each course containing as much content as our competitors' entire offerings
- Their guarantee: *"If after 8 weeks you can't score a job playing video games, simply send us an e-mail and we will refund your money on the spot. It's s simple as that. No harm, no hard feelings. And we still part as friends."*

BetaGamingTesters
Begin with Beta testing and work to get hired in-house. This program is offered by a guy that really sounds like he's been there. Look at all the information that he offers and see if this is a good fit for you.

He also offers a 100% money back guarantee so if he doesn't deliver, you get a refund. Read through what he offers and think about whether his program is the one that is right for you.

GamerTestingGround
Specializing in helping you make money playing games as a Beta tester. Read the FAQ, the testimonials, and see for yourself.

This, of course, also offers a 100% money back guarantee. Look for yourself and see if this is the course for you.

© Phillip Bailey - All Rights Reserved

Game Testing Resources (Gaming resources):

Gaming Options:

- Aces High WWII Interactive Game
- Bigpoint
- GameFly - Online Video Game Rentals
- GameHouse
- Gamemine
- GameStop
- Lord of the Rings
- Metaboli
- Microsoft Store
- Nick Arcade
- Nick Jr. Arcade
- PopCap Games
- Second Life
- Shockwave
- Yahoo Games

Clean Your Computer:

Keep your test computer(s) stable and ready for anything!

- Acronis
- Disk Doctors
- Elephant Drive
- Error Killer
- iolo System Mechanic
- Live Drive
- PC Drivers HeadQuarters
- PC Repair Doctor
- Registry Defender Platinum 2010
- RegZooka

Screen Capture Software:

- CamStudio
- FRAPS

© Phillip Bailey - All Rights Reserved

Chapter 18: You've Been Laid Off – Now What?

Laid off?!? Are you kidding me?

That's right sports fans. This is a far too common occurrence it the world of software quality assurance. So many companies simply have to concept of the true value of in-depth product knowledge, how it reduces cost long-term, or how to use it to create better products.

This is one of the main reasons that so many SQA positions are created as contract only. The assumption is that they can get someone to test their product, help finalize a data migration, upgrade the infrastructure…and as soon as that is done, it is most cost effective to get rid of the QA.

This is very common. It is the way that much of the industry handles software development. That doesn't mean you can't find a long-term position that is stable. I am just letting you know that the software industry – especially the QA end of it – is rife with contract positions.

This creates a great deal of flexibility for you, but also gives you an industry-specific challenge if you want a "permanent" position. I like a stable QA position, myself. It gives me the opportunity to create longer lasting change. And in this field, I am one of a minority of professionals I know that has been at companies for multiple years.

But whether contract or full-time employee, there is always a chance you will be laid off. It is the nature of the industry and is currently the trend in more and

© Phillip Bailey - All Rights Reserved

more companies. The very bright upside to all of this is that (to paraphrase James Canton in The Extreme Future) in the future (the near future), companies will be bidding to acquire the services of technology professionals.

There is a predicted shortfall of qualified technology professionals when compared to the predicted number of technical positions that will need to be filled. Software Quality Assurance falls smack-dab in the middle of that prediction.

Not only will more and more software testing and ancillary positions (as we continue to create more and "better" software), but the lack of ***real*** quality QA Professionals will mean there is a greater need and desire for you and your skills. If you master the principles in this book, develop your skills, and position yourself properly, you can be one of the highly sought after technology professionals.

So, know that being laid off is not rare. I have had one company go out of business, another company relocate, and a division in which I worked with the best QA team I have ever seen eliminated – all while I was working there. That doesn't even count the contracts that I have worked. It's just your chance to upgrade…

© Phillip Bailey - All Rights Reserved

It's No Big Deal

As I said above, getting laid off is not rare. Whether your contract has come to an end or you get downsized, know that QA departments are almost constantly expanding and contracting. It is rare to have a QA department remain intact with the same employees and the same number of employees from year to year.

This is normal. The way that most companies (mis)handle their QA departments, methods, and personnel, most every QA professional has been laid off at one time or another. If you haven't yet, congratulations! Probably time you got prepared for "just-in-case-you-do".

If and/or when you do get laid off, the first thing to do is DON'T PANIC! You will make it through this if you keep your wits about it. Although your parents (or elders) may react with fear and worry, this is not the world they grew up in. This is your time and your industry.

Getting laid off should be an opportunity you seize to land a better position! If you follow the principles I detail, you shouldn't get laid off for any performance-based issues (right?). Getting laid off should just mean that it's time to move on to brighter days and greener pastures.

Don't take it personally, take it professionally. Understand that the developers who loved your work, the QA personnel who you helped improve and grow, and the consumers who reaped the reward of your expert efforts – none of them had anything to do with you getting laid off.

Most times the decision is made by a QA Manager under pressure to keep an unrealistic budget in line. Or from company middle managers or higher who are more concerned about their bonuses than about quality or long-term company good. Or they are simply following instructions from *their* bosses.

It has nothing to do with you…

…just make sure I'm right.

© Phillip Bailey - All Rights Reserved

Seize the Opportunity!

You may not be able to control whether or not you get laid off, but that doesn't mean you don't have control. You still have control over your response. What you decide to do once you have been laid off is up to you.

How will you take the news? What will you do from now until your last day? Do you worry about the burden of finding a new job or do you see this as an opportunity to progress, advance, and upgrade?

When you are laid off, it's usually because the company can't afford to keep you (darn shame), won't afford to keep you (different perspective and priorities), or *thinks you are easily replaced whenever they need more QA* (you're better off getting away from them now).

When you get laid off, you should see it as a golden opportunity to upgrade your employment situation. One company has decided to stop paying you, but since you followed my advice and took every chance to learn and improve while they *were* paying you, you should now be an even better prospect to hire.

Seize the opportunity you have been given! Celebrate the chance for change!

Take yourself (and loved ones) out to one nice dinner. Celebrate the job that was and toast the future that will be. Know that you have in-demand skills. Understand that you were a good hire before, but are an even better hire now!

Don't waste time lamenting the company that didn't have the insight to harness your expertise even further. Take advantage of the development they helped (or allowed) you to achieve and hook up with a company that wants to pay you even more!

As I have said before, software continues to be developed all over the world. This industry is still expanding. Consumer facing software testing as we know it today has only been around since the early 1980's. This is a YOUNG industry! Take advantage of that. Make a difference!!

How…?

© Phillip Bailey - All Rights Reserved

Keep Developing Your Skills

Every position you hold gives you the chance to increase and improve your skills. Testing skills, **software skills**, **computer skills**, leadership skills, process improvement, it's up to you.

Every company that pays you to work gives you the opportunity to get paid to learn and improve. You should never stop developing your skills. You should always be moving forward.

When you get laid off, you have even that much more time to invest in YOU. Take a couple of days and pick up another skill. Learn some new software, take a look at what new opportunities are popping up, and take a moment to evaluate exactly what you want to do next.

This doesn't need to be a time-consuming exercise – you *will* need to get on with a job search if you don't have a position waiting for you already. But ***do*** take the time.

It is important to consider what it is you really want and accurately evaluate yourself and how to achieve you goals. This can be difficult when you are working a full-time job, especially if you get as involved in each position as I do. But take the time – you are worth it.

Update your plan. List the skills you want to develop next.

Update your resume. Make sure you get your latest position and accomplishments on it.

Pull out your cover letter templates. Get ready to do some research so **your cover letters show your value to the companies you target**.

Make sure you have a **Job Interview Success System** in place and ready for tracking each position you attack.

Be ready to **answer any job interview question like a professional** and sound like a natural.

Put together your new **"Wow!" plan** and know that, for you, the sky is the limit.

© Phillip Bailey - All Rights Reserved

You have been given the opportunity to find a new position, develop even greater skills, extend your network, and become an ever more valuable commodity. Take advantage of these disguised blessings and position yourself for the future.

You can be one of the most sought after professionals for years to come if you take advantage of the opportunities presented to you. The future is yours if you want it to be.

You can't control everything, but your career and your future are in your hands. If and when you get laid off, don't take it personally – take it professionally. Make everyone that lays you off regret their decision by becoming the Quality Assurance Professional that I know you can be!

Remember: Luck = Preparation + Opportunity

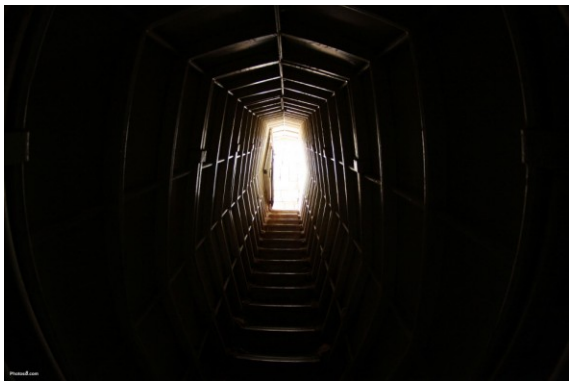
Invest in yourself…go and make your own luck!

Enjoy the Journey

Space we can recover, time never.
~ Napoleon Bonaparte

© Phillip Bailey - All Rights Reserved

Epilogue



You have reached the end of this book. You should see this as the beginning of new opportunities spreading out in front of you. I have given you the tools with which to arm yourself for your journey. You have the information you need to search for and find a niche for yourself in the high-tech industry.

The pace at which technology is begin developed is only increasing. The amount of technological complexity is heightening. All of this technology will need testing. You are now equipped with the tools to join in the game and make a real contribution.

If you have mastered the lessons in this book, you are ready to find a company and hit the ground running. You know how to get started quickly; how to become a valuable employee, how to test, measure, report, and improve. You are prepared to make an impact in any company you join.

You have the knowledge to succeed. It is up to you to put that knowledge to work. If you want to improve, you are the one that will have to put in the effort. No one can improve for you. You have the ability, you have the information, you have the principles, and since you have finished this book, you obviously have the drive.

As you travel through the high-tech world and encounter wonders both great and small, please keep in mind a critical piece of information: **Your ability to continually learn and adapt will be crucial in your quest to succeed**. With the changes that occur in the high-tech industry and with the speed at which those changes occur, you must develop the ability to adapt as needed.

© Phillip Bailey - All Rights Reserved

If you have mastered the principles in this book and taken their lessons to heart, you are already well on your way to developing the skills you will need to succeed in any circumstance. You are ready to raise the level of quality of the products you test. You are schooled in the best practices that have been successfully used by hundreds before you.

Learn the principles…Master the skills…Never stop learning

I hope your journey is fulfilling, your "failures" rewarding, and your testing uncompromising. May you have the favor to work with people of the highest character, companies that embrace the highest quality, and products that challenge your assumptions.

Hold high the Standard of Quality! Be proud to do the right things and do them in the right way. *Be one who makes the difference.*

Go boldly and succeed!

You will never do anything in this world without courage. It is the greatest quality of the mind next to honor.
~ Aristotle

Exploratory testing can be described as a martial art of the mind. It's how you deal with a product that jumps out from the bushes and challenges you to a duel of testing. Well, you don't become a black belt by reading books. You have to work on it. Happy practicing.
~ James Bach

© Phillip Bailey - All Rights Reserved

Quality in a service or product is not what you put into it. It is what the customer gets out of it.
~ Peter Drucker

© Phillip Bailey - All Rights Reserved

Links to Resources

Throughout this book I have listed valuable resources to help you overcome any obstacle that might hinder your ability to create and control your career as a world class Quality Assurance Professional.

I have done my best to categorize each resource and include it in the section where it would be most applicable. But for ease of use, I have also listed each resource in the book right here – so that you have simple one-stop shopping at your fingertips.

Just click on the title or description of the resource you want to investigate and have your browser take you to the appropriate product/service page.

If there are issues preventing you from a career as an Elite Quality Assurance Professional, please let me know. **Tell me what is hurdle is tripping you up and I will do my best to provide a resource to help you overcome the issue once and for all.**

Self-Improvement Resources:

- **Dr. Wayne Dyer**

Computer Maintenance / Repair / Restore Software:

- **Acronis**
- **Disk Doctors**
- **Elephant Drive**
- **Error Killer**
- **iolo System Mechanic**
- **Live Drive**
- **PC Drivers HeadQuarters**
- **PC Repair Doctor**
- **Registry Defender Platinum 2010**
- **RegZooka**

Computer Training:

- **Computer Training Unleashed**
- **Training Center**
- **Total Training**

© Phillip Bailey - All Rights Reserved

Gaming Resources:

- Aces High WWII Interactive Game
- Bigpoint
- GameFly - Online Video Game Rentals
- GameHouse
- Gamemine
- GameStop
- Lord of the Rings
- Metaboli
- Microsoft Store
- Nick Arcade
- Nick Jr. Arcade
- PopCap Games
- Second Life
- Shockwave
- Yahoo Games

Job Hunt Resources:

- Job Interview Success System
- Job Interview Answers
- Amazing Cover Letters
- Amazing Resume Creator
- Best Engineering Resumes
- Wow! You're Hired

Language Resources:

- Rapid English Now
- Transparent Language

Resume Posting Resources:

- Resume Bucket
- Yahoo Hotjobs
- Monster
- Career Builder
- LinkedIn

Visa Resources:

- H1Base

© Phillip Bailey - All Rights Reserved

Writing Help Resources:

- **101 Ways To Power-Up Your Writing**
- **Confident Grammar**
- **White Smoke**

© Phillip Bailey - All Rights Reserved

Glossary

- **Acceptance Testing –** Testing conducted to determine whether a software build is of acceptable quality to test further. Normally performed to validate the software meets a set of agreed acceptance criteria.
- **Ad Hoc Testing –** A testing in which the tester tries to break the system by randomly trying the system's functionality.
- **Agile Testing –** Testing practice for projects using agile methodologies, treating development as the customer of testing and emphasizing a test-first design paradigm.
- **Alpha –** Alpha is the test phase in which the product is usable in a test environment but not necessarily bug-free
- **Automated Testing –** Testing employing software tools which execute tests without manual intervention
- **Benchmark Testing –** Tests that use representative sets of programs and data designed to evaluate the performance of computer hardware and software in a given configuration
- **Beta –** Beta is the test phase in which the product is feature complete. This version may be distributed for limited public testing and feedback
- **Beta Testing –** Testing a release of a product conducted by the public
- **Black Box Testing –** Testing based on an analysis of the specification of a piece of software without reference to its internal workings
- **Boundary Testing –** Testing of the program's own rules, from within
- **Bug –** A fault in a program which causes the program to perform in an unintended or unanticipated manner. An anomaly, defect, error, exception, or fault
- **Bugbase –** The database in which all defects (bugs) are logged, prioritized, and tracked. (see Defect Tracking System)
- **Compatibility Testing –** Testing how well software performs in a particular hardware/software/operating system/network/etc. environment.
- **Defect –** See "Bug"
- **Defect Tracking System –** A tool used by a project team to organize, prioritize, and track all defects (bugs) related to the project. The engine most often used is a database, although projects may also use a spreadsheet
- **Endurance Testing –** Checks for memory leaks or other problems that may occur with prolonged execution.

© Phillip Bailey - All Rights Reserved

- **End-to-End Testing –** Testing a complete application environment in a situation that mimics real-world use, such as interacting with a database, using network communications, or interacting with other hardware, applications, or systems if appropriate.
- **Final Candidate –** A Release Candidate that meets the criteria to have the final tests performed before Gold Master is declared
- **Functional Specification –** A document that describes in detail the characteristics of the product with regard to its intended features.
- **Functional Testing –** Testing the features and operational behavior of a product to ensure they correspond to its specifications. (See Black Box Testing)
- **Gold Master –** A shippable version of the product
- **Gray Box Testing –** A combination of Black Box and White Box testing methodologies to test software against its specification but using some knowledge of its internal workings
- **GUI –** Graphic User Interface
- **Integration Testing –** Testing of combined parts of an application to determine if they function together correctly. Usually performed after unit and functional testing. This type of testing is especially relevant to client/server and distributed systems
- **ISO 9000 –** A family of standards used for quality management systems. ISO stands for the International Organization for Standardization
- **Load Testing –** Testing that creates demand on a system to define its limits by measuring the system's response
- **Localization Testing –** Testing software specifically designed for a specific locality – this testing focuses on language
- **Metric –** A standard of measurement. Software metrics are the statistics describing the structure or content of a program. A metric should be a real objective measurement of something such as number of bugs per lines of code
- **Negative Testing –** Testing aimed at showing software does not work and checking for failure states
- **Performance Testing –** Testing conducted to evaluate the compliance of a system or component with specified performance requirements. Often this is performed using an automated test tool to simulate large number of users
- **Post Mortem –** An end-of-project wrap up meeting to review the project and set a path for improvement
- **Pre-Alpha –** A phase of testing during which builds that are only semi-functional. Testing focuses on specific features, not holistic testing

© Phillip Bailey - All Rights Reserved

- **QA –** Quality Assurance
- **Quality Assurance –** All the planned or systematic actions necessary to provide adequate confidence that a product or service is of the type and quality needed and expected by the customer
- **Regression Testing –** Testing a previously tested program following modification to ensure that faults have not been introduced as a result of the changes made
- **Release Candidate –** A pre-release version, which contains the desired functionality of the final version, but which needs to be tested to verify it is ready for release
- **Sanity Testing –** Brief test of major functional elements of a piece of software to determine if it's basically operational (See also Smoke Testing)
- **SCM –** Software Configuration Management
- **SDLC –** Software Development Lifecycle or Software Design Lifecycle
- **Smoke Testing –** A quick-and-dirty test that the major functions of a piece of software work. The name of this test originated in the hardware testing practice of turning on a new piece of hardware for the first time and considering it a success if it does not catch on fire
- **Soak Testing –** Running a system at high load for a prolonged period of time. For example, running several times more transactions in an entire day (or night) than would be expected in a busy day, to identify and performance problems that appear after a large number of transactions have been executed
- **Software Development Lifecycle –** The processes and parameters that govern the creation of software in a defined project or projects
- **SQA –** Software Quality Assurance
- **Stress Testing –** Testing the limits of a system by altering the external variables
- **System Testing –** Testing that attempts to discover defects that are properties of the entire system rather than of its individual components
- **Test Bed –** An execution environment configured for testing. May consist of specific hardware, OS, network topology, configuration of the product under test, other application or system software, etc. The Test Plan for a project should enumerated the test beds(s) to be used
- **Test Case –** A commonly used term for a specific test
- **Test Environment –** The hardware and software environment in which tests will be run, and any other software with which the software under test interacts when under test including stubs and test drivers

© Phillip Bailey - All Rights Reserved

- **Test Plan –** A document describing the scope, approach, resources, and schedule of intended testing activities. It identifies test items, the features to be tested, the testing tasks, who will do each task, and any risks requiring contingency planning. Ref IEEE Std 829
- **Test Suite –** A collection of tests used to validate the behavior of a product
- **UI –** User Interface
- **Unit Testing –** Testing of individual software components. Often this testing is performed at the code level
- **Usability Testing –** Testing the ease with which users can learn and use a product
- **Use Case –** The specification of tests that are conducted from the end-user perspective. Use cases tend to focus on operating software as an end-user would conduct their day-to-day activities
- **Waterfall Testing –** A model that is sequential, flowing through several predefined phases
- **White Box Testing –** Testing based on an analysis of internal workings and structure of a piece of software

© Phillip Bailey - All Rights Reserved

About the Author

Phillip Bailey has been a Quality Assurance professional for over a decade. He has successfully lead projects as diverse as ***The Flip*** video camera to ***Magic: The Gathering Online***; from ***The Fly Pentop Computer*** to ***Pokemon***.

Phillip has managed the testing of computer games, educational software, IT infrastructure, LAN/WAN, network security, web sites, databases, computer hardware, and consumer electronics. Since the late 1990s, he has trained over 200 quality assurance testers, leads, and managers.

Beginning as a tester, Phillip has worked as a lead, manager, trainer, and consultant. He has benefitted from working with and for some of the finest Quality Assurance Professionals in the San Francisco Bay Area and Silicon Valley.

Through many years and countless projects it became clear to him that a non-technical avenue of entry was necessary to enable high-quality professionals to enter the QA field. It is his greatest hope that this book helps the next generation of testers become effective in the software development industry.

Phillip continues to write about QA, software testing, process improvement, and project management at: **www.Successful-Quality-Assurance.com**. You can read more there, subscribe to his blog and eZine, and get answers to your software testing questions.

© Phillip Bailey - All Rights Reserved

Sample Status Reports

Status Report #1

Tests Completed:
WebRadio 2.0 v1.1

- Functionality testing completed Wednesday 1/4

Priority Issues:
N/A

Current Projects:
WebRadio 2.0 v1.1

- Testing on schedule
- Compatibility tests in progress
    - Estimate completion by Friday 1/13

Unscheduled Projects:
N/A

General Comments:
Team members have been very helpful and supportive.

© Phillip Bailey - All Rights Reserved

Status Report #2

Tests Completed:
N/A

Priority Issues:
WebRadio 2.0 v1.1

- **Compatibility target completion date slipped** due to unscheduled customer-related support for Ops department – **impact to testing = approximately 2 days**
    - **Update target completion date: Wednesday 1/18**
    - Old estimated completion date: Friday 1/13

Current Projects:
WebRadio 2.0 v1.1

- Compatibility testing slipped 2 days
- Compatibility tests in progress
    - Estimate completion by Friday 1/18

Unscheduled Projects:
WebRadio 1.0

- Customer-related support for Operations department was given highest priority
    - Impact to WebRadio 2.0 testing = Compatibility testing delayed on a 1-for-1 basis
        - Total delay was 2 days

General Comments:
Customer support for Ops consumed more time an effort than anticipated. Will work to find a way to catch up in testing to get back on schedule. Do not yet know how this will be accomplished.

© Phillip Bailey - All Rights Reserved

Status Report #3

Tests Completed:
WebRadio 2.0

- Compatibility testing completed Tuesday 1/17
- Stress testing completed Thursday 1/19

Priority Issues:
WebNews&SportsTalk

- Marketing Release Document **not delivered** to QA on Friday 1/20
  - **Updated target delivery date = Friday 1/27**
  - **Impact to project: TBD**

Current Projects:
WebRadio 2.0

- Testing continuing on schedule
  - Compatibility testing dovetailed into Stress testing to increase efficiency
- Load testing in progress
  - Target completion date = Wednesday 1/25
- New build expected on Friday 1/27
  - Bug Regression will begin once new build is verified testable

Unscheduled Projects:
N/A

General Comments:
QA team worked to double up testing to get back on schedule. Compatibility testing was performed at same tame as Stress testing. This efficiency enabled team to maintain schedule. This efficiency should be usable in future testing. Will report on progress as appropriate.

****Note:** I will be out of the office on Thursday 1/26 due to a doctor's appointment.

© Phillip Bailey - All Rights Reserved

Status Report #4

Tests Completed:
WebRadio 2.0 v1.1

- Load testing completed Wednesday 1/25

Priority Issues:
WebNews&SportsTalk

- Marketing Release Document **not yet delivered** to QA
    - **Test creation schedule is at high risk**

Current Projects:
WebRadio 2.0 v1.2

- Build verified testable
- Bug Regression in progress
    - Target completion = Tuesday 1/31

Unscheduled Projects:
N/A

General Comments:
N/A

© Phillip Bailey - All Rights Reserved

Sample Bugs

Sample Bug #1

Brief Description:
Application crashes when <Help> button is selected

Expanded Description:
Application crashes when a user selects any <Help> button on any page in the program

Reproducibility:
Always

Steps to Reproduce:
1. Launch program
2. Select any <Help> button on any page

Result:
The program crashes. The application closes immediately. Operating system is not affected. Program can be re-launched safely and without issue.

Expected Result:
The Help menu should open. The program should not crash

© Phillip Bailey - All Rights Reserved

Sample Bug #2

Brief Description:
Program hangs when 3 Fearless Spider cards are played

Expanded Description:
When 3 Fearless Spider cards are played, all targeting one another, the game functionality creates an endless loop. The program does not respond to user input, but closes itself after approximately 6 minutes the endless loop has begun.

Reproducibility:
Always

Steps to Reproduce:
Using a deck that has at least 3 Fearless Spider monster cards in it:

1. Launch a 2-player game
2. Continue through the game until you are able to put 3 Fearless Spider cards into play
3. With no other monster cards in play, play the 1st Fearless Spider card
4. Play the 2nd Fearless Spider card, and target the 1st so that the 1st Fearless Spider card is removed from play
5. Play the 3rd Fearless Spider card, and target the 2nd so that the 2nd Fearless Spider card is removed from play – bringing the 1st Fearless Spider card back into play
6. As the 1st Fearless Spider card comes back into play, use it to target the 3rd Fearless Spider card so that the 3rd Fearless Spider card is removed from play – bringing the 2nd Fearless Spider card back into play

Result:
At the point the game and program are in an uninterruptible endless loop. User has no way to break this endless loop. After approximately 6 minutes, the program will close itself. Program can be re-launched without issue.
See attached file for debug output

© Phillip Bailey - All Rights Reserved

Expected Result:
The game should give the user some mechanism with which to interrupt the endless loop. Either pause after each sequence to allow user to play a card to end the loop – OR – give the players an option to call a draw.

© Phillip Bailey - All Rights Reserved

Sample Bug #3

Brief Description:
Exploding pixel obscures entire screen when Holy War Hammer is cast

Expanded Description:
In multi-player combat, when Cleric casts Holy War Hammer spell, the hammer graphic obscures screen making game un-playable.

Reproducibility:
Intermittent.
Bug has occurred 37 out of 50 times

Steps to Reproduce:
1. Launch a multi-player game
2. Select <Cleric> for one of the characters
3. Navigate through the game until combat occurs
4. During combat, have the Cleric character cast Holy War Hammer spell

Result:
The whole screen is obscured by the blue War Hammer graphic as it rotates on its axis attacking enemies. When this occurs, nothing but the Hammer graphic is visible until the spell dissipates. This makes the game not playable while the spell is active.
****see attached picture****
****Note:** This bug occurs in multi-player **only**
This bug is Intermittent. Reproducibility is 37 out of 50 times**

Expected Result:
War Hammer graphic should not obscure the screen.
The Hammer graphic should be approximately 1" in size.
All allies, enemies, map features, etc. should continue to be visible during gameplay regardless of the spell cast.

© Phillip Bailey - All Rights Reserved

Sample Bug #4

Brief Description:
Video capture ends too soon

Expanded Description:
When in video capture mode, the KiddieCam will only record 4 seconds of video

Reproducibility:
Always

Steps to Reproduce:
Make sure the KiddieCam has no pictures or video on it:

1. Set KiddieCam to video capture mode
2. Begin recording video
3. Note how long the KiddieCam will record before it stops
4. Replay the video to verify the amount of video you were able to record

Result:
KiddieCam will record only 4 seconds of video

Expected Result:
Video capture should last for more than 4 seconds. The amount of video captured before the recording stops should be at least 8 seconds (per Functional Spec 2.6 v8, page 18, paragraph 3 "When in video capture mode, KiddieCam will capture 8-10 seconds of video…")

© Phillip Bailey - All Rights Reserved

Bug Template

Brief Description:

Expanded Description:

Reproducibility:

Steps to Reproduce:

1.
2.
3.
4.
5.

Result:

Expected Result:

© Phillip Bailey - All Rights Reserved