

Trading Money For Time: When Saving Money Doesn't (And When It Does)

Trading Money For Time: When Saving Money Doesn't (And When It Does)

Michael Larsen

How you spend your time is more important than how you spend your money. Money mistakes can be corrected, but time is gone forever.

â David Norris.

In any endeavor I participate in, I have choices. In many of life's transactions, I can spend money to have something done, or I can spend the time to do that same thing. For some things, it's worth it to invest the time to make up or offset the amount of money to be spent. At other times, there is a level of involvement and a required expertise that makes not spending the money a barrier to achieving my goals. There is a tradeoff; money for time, or time for money. The problems arise when we don't give both of each of these areas proper consideration. In the world of software development, containing costs is important, but so is delivering a good quality product at the right time. Containing costs when a project is on time is a good thing, but losing time and not delivering a product when scheduled can cost both time and money, often in the lost revenue from dissatisfied customers. At the furthest extreme, the resulting loss of time could result in legal action for not being able to deliver on commitments. This chapter examines the tradeoffs between money and time, and demonstrates examples where time investment has meant financial gain, and where lack of understanding of time has negated or even negatively affected a company's finances.

A New Project Gets Underway

It's time for another software project to begin. Each time I go through this process, I have a similar set of experiences. Each time I need to:

- â ¢ Address physical infrastructure needs
- â ¢ Determine where to set up machines
- â ¢ Gather and install software applications to support the testing needs
- â ¢ Consider the scope and types of testing that will be required

Physical infrastructure needs include buying and setting up server and client system and networking equipment. It may also involve internal or external hosting considerations, plus the adequate securing of these resources. When we consider physical machines, potential lab space for the devices and the need for adequate cooling must be considered. Our software application under development also has infrastructure requirements. SDK's and IDE's are needed for software development. We must configure and maintain file services, database services, web services, etc. We also have to consider deployment of the application(s) for testing and for general use. The specific testing areas (and the software testing tools that we utilize) must also be considered. Unit testing. Functional testing. Load testing. Performance testing. Para functional and human factors/user experience testing. The number of areas to manage and maintain is dizzying!

I have witnessed various companies and their management teams wrestle with the goal to improve

the testing and development process. These goals may be to speed up delivery, or add testing coverage for initiatives that haven't adequately been tested. Many management teams also provided fanfare and cheering about the time savings due to some new enhancement. It might be a new tool to automate all of the regression testing. It could be a new server to run a battery of virtual environments. A robust versioning and back up system to keep all versions of files updated, tagged, secure and available would streamline the application lifecycle.

These are admirable goals, and when implemented, the net result can be more productive testing. Work can be completed more quickly.

All of these initiatives cost us *something*. To set up any of these enhancements, our teams have to spend something to bring them to fruition. With organizations squeezing budgets and economizing where possible, some businesses are reluctant to spend. During economically challenging times, many organizations opt to soldier on, using tools already in place (if any exist) and machines that are already at work in their current state. Money is not spent, so our costs are lower. What this "cost saving" doesn't address is the *time* trade-off made to perform these tasks. The Cost of Testing is not exclusively about money, very often we must include time in that cost analysis as well.

Where Are We Spending Our Money?

As our development project ramps up, the resources available will dictate the total time needed to complete it. The number of people on a project certainly adds to the total price tag of a project, as well as the total amount of time a project may take. With a testing team, two testers working together will very likely complete more testing than one tester will. The combined test coverage is often *much* greater than the sum of individual tester's standalone efforts. Having more powerful computers will certainly shave the time it takes to build applications. Likewise, having more machines makes it possible to deploy more test environments.

Support for multiple Operating Systems provides additional challenges. Microsoft Windows, MacOS, and HP UX are examples of Operating Systems that have an upfront licensing cost for each installation. Linux and FreeBSD are examples of open source Operating Systems that are available for free. Beyond just the cost of the Operating Systems is the time needed to give each system thorough testing. Each supported platform adds a time multiplier as to how long it will take. There are also variations in Database software, web server software, Office Productivity applications, and other components that may be required to evaluate the functionality of the application under test. In addition there are variations in the way the software is written and deployed (pre-compiled code such as C++ vs. dynamic application code like Java). Each variation requires another environment be set up and tested.

Testing these multiple OS's requires choices be made. We can purchase and maintain individual hardware machines. We can also utilize virtualization technology to reduce the need for multiple physical computers. The host server will require many orders of magnitude more capacity and performance than the individual computers we would have virtualization replace. If a project requires six systems of equal capability, virtualizing those machines will require sufficient resources on a single server to be able to house and run those six environments. That host server will require enough disk space, RAM and system resources for the six virtualized environments to operate and respond as though they were standalone machines.

Testing tools range from free, open source options all the way up to very expensive proprietary commercial systems. The application under test often determines which tools, if any, will need to be used. There are many free and open source test tools for testing web applications. By contrast, most of the tools that test a compiled application from a "black box" perspective are commercial products (though there are also a number of open source options available that allow us to create

tests and automate many tasks).

As I have seen over the years, not spending for these items or areas can provide a short-term cost savings. We made do with the servers and machines we had on hand. We also managed to get by with a limit on software licenses (within reason and within the bounds of what is legal; I do not advocate piracy to cut costs at any time). I have set up test beds and test labs using free OS software, using a variety of Linux variations. I have also used both commercial and free virtualization software, where the host was running Linux, and the guest machines likewise represented a variety of free operating system options. These test environments also have been set up to use as many free or inexpensive variations of open source tools for testing purposes. There is no question that using these techniques has saved money.

What's The Tradeoff? The Tradeoff Is Time

There is an adage that says "You can have a system that is cheap, fast, or of good quality. Pick two." Price, time and quality rarely move up and down together. If one area gets incremented, others are decremented. When we spend less to perform a task, the time to accomplish the task often increases. The quality of the resulting output may also decrease. In my experience, companies want to bring costs down, while keeping the quality level as high as possible. To make both goals succeed (lowering costs and improving quality), in most cases we will have to allow time to increase.

Let's use a build server as an example. If we chain a number of computers together to perform builds the overall time to complete the build process will diminish. What could shaving 25% of the time off completing a build do for your development and testing teams? Think of the amount of time that could be applied to testing and fixing issues discovered, rather than waiting for the build to finish. The quicker the turnaround from development to testing, the quicker issues can be discovered, addressed and fixed. Quicker builds allow for better integration of new code. Quicker builds allow for a system that is up and running and more frequently available. With the effort to set up the scripts and tests needed, the quality of the build process can be improved. The productivity gains were produced by spending money to add server horsepower. Paying for the human brainpower to create scripts and processes makes for fewer errors in the build process.

To contrast, let's consider what would happen if a new initiative of ours requires a server, but our budget doesn't allow for it. We could share the workload on the existing build servers. We could pull one of the machines out of the build process and make it a standalone server. What happens? Cost savings are immediate; we didn't have to purchase a new server. The quality of the builds may be unaffected, but the time to complete the builds has increased. If the system performs daily builds, the time we have lost because of the server change will be multiplied by the number of days for as long as the projects are active. The benefits of quicker turnaround for development and testing will not be achieved.

I have used test environments with open source operating system software and open source testing tools. On the surface, the environments appear to be totally free or have a very low physical cost to implement. In my experience, deploying, maintaining and administering these operating systems and applications requires a significant time commitment on the part of the development, test and IT teams. Open source software (and the communities that help with bug fixes, documentation and implementation of new features) is on par with the development of many commercial applications. The ease of installation and maintenance of the installed software has likewise improved dramatically. Still, a high level of knowledge and skill is required to keep multiple open source applications and tools running smoothly. That knowledge development and maintenance requires time.

Several groups I have been a part of, especially at the beginning, had no formal tools in place. Most of the testing steps are performed manually. Many tasks are best suited to active manual testing, but when all our testing is manual, the time required to complete it can be significant. Multiply our manual testing efforts by the number of projects. The time commitment to do all of the needed testing rises exponentially. Few organizations have the luxury of long periods of time to make sure all testing is performed. What tends to happen is that testing time gets cut (or at best held to the required scheduled time). The total amount of tests get cut, which can lead to a lower quality product. When we apply a risk based test approach, we aim to maximizing the potential of covering areas we consider most critical. Even with this method, consider how long it would take to cover all areas in the time limit. Where will the cuts be made? Should we eliminate human factors testing? Will considerations for user experience go untested or receive a lower priority? Will features that are desired but can't be fit into the timeline be moved to later releases? If so, how long will it be until we will be subject to the hands of Kronos once again?

What is Time Really Worth?

Let's say an average developer earns \$50/hour. That includes all benefits. \$50/hour is not unreasonable given wages, vacations, holidays, benefits, and sick days. What other aspects of their work day need to be considered? We do not have the ability to simply wake up, teleport ourselves to work, meet our objectives, and then immediately zip away to do other things. We commute and we spend money to make those commutes. When we work at home, we pay for infrastructure to allow that ability. We purchase clothing and food to support us while we are in the mode of working.

Some commentators have said that our full 24 hour day should be viewed in light of what we earn. To really determine the value of our time, we would need to take the total amount of time in a week, and divide it into an hourly figure. If we were to divide those hours in comparison to our average developer earning \$50/hour, that individual is *really* earning \$11.90/hour.

This is the concept of the "real wage", and it was popularized in the book *"Your Money or your Life"*, written by Vicki Robin, Joe Dominguez, and Monique Tilford. ¹ By using this approach, the real wage is almost 80% less than what our mid-level developer earns while working. We can also look at a less extreme example. When all activities and expenses that immediately impact our work are considered, the difference in the real wage can be anywhere from 30% to 50% lower than our earned hourly rate. When we realize just how much our real wages are, we feel prompted to do significantly different things with our spending and behavior.

The value of time becomes more critical as additional people come into the picture. Time is multiplied, and each non-optimized hour affects all of the individuals involved in a project. Going back to my previous build server example | imagine a daily build currently takes three hours. Adding another server could bring the time of the build process down to two hours. Put into monetary terms, if a team of 10 developers was able to save just two hours each per week, the team would then have 20 hours more per week to add value to the product. That is a saving of \$1000/week or close to \$50,000/year!

The "real wage" for an organization can be greatly increased by its ability to save time. It can also be greatly decreased by the time it loses.

Is Time Always the Enemy?

It is possible to lose money when time is not taken into account. Time can also be used to an organization's advantage as well. Using the build server example once again, we will need to spend money to get the hardware and to achieve the time savings. What if we could get the same

hardware for ½ the cost? Could we get double the hardware for the same cost? For the past 30 years, computer and networking equipment has roughly doubled its performance every 18 months. This doubling capability has also been closely tied to improvements in RAM, disk space, peripherals, networking devices and the cost of internal components of motherboards and expansion cards. We can see a general trend that machines effectively double in power every 18 months to two years, while staying relatively even cost wise.

With this in mind, does the time not saved, and the subsequent lowering of the real wage of the organization, offset the potential of gaining double the power in machine performance in 18 months? If the answer is "no", then waiting could potentially lower costs while still getting a better return on time. Consider what the cost/benefit analysis of buying a piece of hardware now versus later would be (specifically with the idea of the gained time for our example team providing a \$50,000 cost savings each year).

What other steps could be put off until later? Consider what happens when automating tests. I have a natural tendency to want to get in early and start working with the system as soon as the UI or other elements are coded. By automating tests early, the ability to test the system early and often should yield a good return on the time spent to automate the steps. What happens if the UI changes? Additional fields are added (or removed), or just relocated to a different part of the screen. My automation efforts will have to be modified (best case) or completely redone (worst case). In this instance, is it better to wait until the UI is finished? The odds (and risk) of the UI being changed are now greatly lessened, and my automation efforts are less likely to require reworking or redoing.

Balancing Short Term and Long Term Gains/Losses

When I look to balance and consider the costs associated with money and time, it's important to look at both short term and long term views. There is always a return on investment when it comes to the time put into any process vs. the value of the output. Whether the return is positive or negative, and by how much, is up to the team and the organization. This may also be dictated by issues that we have little control over. Using automation as an example, there is a formula that Deon Johnson refers to as the Automation Return on Investment (ROI) [2].

The Automation ROI is determined by comparing Cost Savings, Increased Efficiency, and Reduced Risk. The equation we use for this is:

$$ROI = (Gains - Investment Costs) / Investment Costs$$

The larger challenge is for us to convince management that there is likely to be a *positive* return for our investment. It may cost our organization \$100,000 to automate a series of tests. If the net result of that automation is quicker time to market and an increase in sales, that's a strong incentive to make the changes. If the \$100,000 spent for automation does not generate additional revenue to equal or surpass the initial investment, it can be argued that the return on investment is negative. In this situation, a manager would be unlikely to recommend continued automation efforts.

Figure 1: The cumulative benefits over time comparing a range of tests performed manually vs. with some level of automation (numbers are in months) 3.

What I think needs to be considered is the window of time being used. One financial quarter may be far too short a time period to determine the value of a testing effort. Sadly, quarterly numbers are what drive many organizations. In this instance, a \$100,000 deficit may doom a project. However, if a longer view is taken, that initial deficit will be erased. As seen in figure 1, if an investment is made up front, and that investment can get a modest performance increase over an extended period of time, the return on that investment can be considerably higher than not making the investment. Compared to a longer time period (say, three or four years) the benefits and cost savings will become more apparent [4](#).

As an example of balancing short term expense with long term benefit, Cisco Systems offers an interesting example. They invested time and resources between 1992 and 1995 to create a robust automation framework around the Tcl language. This framework was then used to create extensive automation libraries and script suites to test the Cisco Internetwork Operating System (IOS). Scores of testers were responsible for developing these scripts. These scripts would then communicate on the routers' console ports, and send commands that would set up the router's internal configuration parameters. These Tcl commands, using the Expect extension, were capable of initiating numerous test sequences. The scripts allowed the testers to confirm results, determine the pass/fail criteria and then tear down the tests.

Decades of cumulative man hours were put into creating the libraries needed to allow the test framework to be completed and enhanced over the initial three years. Large lab facilities were created to house all of the equipment required to run the tests. This required a large up front expense, but it proved to be an excellent way to run regression tests and to capture information for reports and for issue tracking. Today, Cisco's investment in Tcl goes beyond testing. Cisco IOS has an extension that allows administrators to use Tcl directly. They can use it on the console port to configure routers and switches in production environments, as well as to create queries for SMTP messages and also for accessing SNMP MIBs [5](#).

I have also had many experiences where the short term benefits of saving money outweighed the long term issues. One of the favorite "war story" examples used by many testers (and yes, I've experienced my share, too) come from experiences with outsourcing. It's considered a boon when outsourcing effort helps you and your team get more done and increase profits. It's seen as a bane when that outsourced team ends up taking over your job. I've been in both situations, and can speak to exactly how both situations feel. The business case I've heard most often used to justify outsourcing testing efforts is that the organization can save money. To be fair, when it is done correctly, with a high level of communication and collaboration, and with a motivated and well-synchronized team, outsourcing *can* be beneficial and *can* be a cost saver.

For outsourcing to save money, the entire business case must be considered. For example, I was part of an outsourcing process when I worked with a Japanese video game company. For our group of testers, we were the outsourcing group working on their behalf, helping to ready titles for release and distribution into the U.S. market. We reaped the benefits of that relationship, and provided a good service with a high degree of quality and satisfaction to our clients.

By contrast, a number of years ago, when I was working with a technology company, there was a large initiative where management decided to have an outsourced group assist in the development and the testing. The reasoning was that there was more than enough work for our group to do and this would allow us to quickly get a large number of tests completed and have the reporting turned around to management quickly. That was the goal, a way to get a lot of testing done quickly, and at minimal cost. At first, it looked like the organization would save a lot of money. Over time it became clear that what management had hoped would be tested, and in the timeframe they wanted, wasn't. Several meetings were required to clarify the situation. Each meeting needed to be conducted with the management team in one location and the development and testing team half a world away. This made the effective time between each daily build and receiving results to be two

days.

Days turned into weeks and weeks into months. It became clear that any short term cost savings was consumed when the product was not ready to be shipped on time. As the team continued to try and get the product ready, more issues appeared. Some issues were related to usability, some with performance, and quite a few issues affected the overall user experience. Ultimately it was determined that the project would not meet the needs of the customers, and both development and testing were halted. Subsequent work was brought back in house. The cost benefits originally envisioned never materialized. In this case, the cost in both money and time was significant, with a negative return on investment.

So, if outsourcing is an option to consider, here are a few questions we should be asking:

1. What is the cost of trying to collaborate across greatly differing time zones?
2. What are ways that the collaboration can be improved?
3. Would tools aid the process of collaboration? If so, how much would they cost?
4. How much travel will be required to keep the two groups in sync?
5. How long will it take to fully ramp up an outsourced team?
6. How will these changes effect the designers of the system?
7. How much does the design team have to document for the outsourced team to be effective in their testing?
8. How Much will it cost to produce that documentation?
9. Will there be any long term value in the documentation produced?

Asking these questions up front, and many more, plus considering all costs associated with them, will give a truer picture of the expense, in terms of both money and time.

How Can We Optimize Costs (Money and Time?)

Finding the magic point where we can save money and keep to a good amount of time while ensuring high quality is a delicate balance. Many might say it can't be done. I say with an eye towards having all three perfectly balanced, it's unlikely. I do believe, however, that there is a way to get to where the monetary cost is as low as possible, the overall time is as long as needed or short as needed and the quality is as good as needed to be effective and release a good product to stakeholders who want it in a timely manner. There is no magical incantation or special formula to let the organization know exactly where that "good enough" is, but there are a number of areas that I feel can be examined and, when taken into context, useful to gauge whether or not a project or application will fulfill the needs of the customer, in a way that is timely and provides for good cost savings to implement it.

1. Determine the Most Critical Areas to the Stakeholders
2. Focus on the Highest Risk Areas First
3. Make Sure to Ask the Right Questions
4. Fix the Show Stoppers, and Understand what Constitutes a Show Stopper
5. The Perfect is the Enemy of the Good

Determine the Most Critical Areas to the Stakeholders

Who is the product for? What is their expectation? Does our solution meet it? If the answer is âyesâ, then it is safe to say we are on the right path. If it does not, no amount of cost cutting or optimization will matter. When I was sitting in a development and testing meeting, discussing a particularly time sensitive software project, our development manager asked the assembled group: âDo we make software here?!â When the developers and testers said yes, he shot back âNo, we do not! We make a *solution* that solves a *problem* for our *customers*. At the end of the day, if we donât get that part right, it doesnât matter what else we *did* get right!â

Focus on the Highest Risk Areas First

The areas that matter most to our stakeholders are ultimately the highest risk areas. It doesnât matter if we agree or not, they are the ones who want the application, and their wish list is whatâs driving the purchase of the application. An example might be to make sure that any UI screens can be viewed in an 800x600 window. It may seem like a trivial issue, but it isnât when the target application is going to run on a kiosk where that is the maximum resolution. Screens that do not look good or require scrolling, especially in an application that uses a touch screen or does not have a method for scrolling, will be a major problem. If this is a key consideration, testing with an 800x600 screen better be one of the first priorities in our test plan.

Make Sure to Ask the Right Questions

Who are the intended users? Does a product already exist that meets this need? If we are first to market, does the application do what our customers want it to do? If we are not first to market, what differentiates us from the competition? Is that difference significant enough to make for an appealing alternative with the established players? What kind of environment will the application be deployed in? Does the product live up to the expectations set for it? Are there any legal requirements we need to be aware of? Each of these questions provides valuable information to help testers set their priorities on the right areas. This information helps make sure that, again, the areas that matter the most get the most coverage. Areas that are less important, or not important at all, get prioritized appropriately.

Fix the Showâ Stoppers, and Understand What Constitutes a Showâ Stopper

Many testers are trained to look at crashes as show stoppers, but they may not understand that a key business rule is being violated, or that a poorly worded paragraph or stray punctuation could prove embarrassing should it get out. If we press a combination of keys, and the application crashes, that looks bad. Many testers will rightfully state that this is a big issue and a âshow stopperâ, but is

it? Is the combination of keys a regular occurrence, such as a combination of regularly used shortcut keys, or is it an unusual combination that is very unlikely to come into everyday use? By comparison, look at the main text that spells out what the users of the software will do as a service for a customer. Imagine that the text is misspelled and the punctuation is wrong. Many testers would consider text issues to be a low level cosmetic bug. That may be true, but in this case, it may be an issue that really sets off an alarm in the customer buying the product. The random key presses that caused the crash was a way lower priority compared to the text on the page that spelled out the agreement between the service provider and their client.

The Perfect is the Enemy of the Good

There may be many small issues that have been found, and there may be many low traffic areas that may not have received thorough testing. The ship window is two weeks away, and we determine that we can cover all of the remaining areas, and many of these little niggling issues can be resolved and retested, but only if we get six weeks of additional testing time. What do we do? When faced with this option, I tend to look at the areas that have not been tested and evaluate the risk if any of those areas were to house a monster bug. What are the odds that area will be hit? What are the chances that the issues I have discovered will be seen as catastrophic? In this case, I rate on a scale of 1 to 5, 1 being really critical and 5 being virtually unlikely to happen, and see what rates a 1, a 5, or in between. I then rank them in order of that gut feeling and I see what I can hit and what I cannot. I may have to leave several 5s on the table, but I better make really sure that I have covered all the 1s in question. After doing that, the project manager, development team and test team, along with any other key stakeholders, will have to make a judgment call. Do we keep digging and testing, to see what else we can uncover, and risk not making our release date? Do we roll with it and decide it's time to let the product go, because we have all decided that, based on the criteria we have received, the risk areas we have covered, the questions we have asked, the answers we have received, the feedback we have received from end users, and the showstoppers we have fixed, have given us the gut feeling that we are ready to go live? Different situations will inform either of those decisions, but there are times when a good enough really is, and waiting for perfection will prove more costly than beneficial.

Conclusion

The cost of testing is real. There is always a price we have to pay if we want to improve the quality of the software. Sometimes that cost is in dollars, sometimes that cost in hours, days, weeks or months. Many times, we have to make tradeoffs to deal with both. When we look to save money due do not taking additional time into the equation, it's entirely possible that the money saved will be outweighed by the opportunity costs of the time we have lost. Take the time to determine what your real wage as a tester, as a manager, or as an organization, actually is. Consider what the value of your time actually is, and make sure to include that when trying to determine how to reduce the cost of testing. With an eye towards looking at both money and time, it's possible to strike a balance where both can provide a positive return on investment.

References

1 â Robin, V., Dominguez, J., & Tilford, M. (2008). *Your Money or Your Life: 9 Steps to Transforming Your Relationship with Money and Achieving Financial Independence: Revised and Updated for the 21st Century*. New York: Penguin.

2 â How Is Automation Returnâ Onâ Investment (ROI) Calculated?, Automated Testing Institute,
http://www.automatedtestinginstitute.com/home/index.php?option=com_content&view=article&id=1097:faq,
retrieved electronically on August 18, 2010.

3 â Scumniotales, J., Why Incremental Development is Better, an ROI Perspective,
<http://agile.scumniotales.com/2009/02/why-incremental-development-is-better-an-roi-perspective.html>,
retrieved electronically on November 15, 2010

4 â Johnson, D, Test Automation ROI, Automated Testing Institute,
http://www.automatedtestinginstitute.com/home/articleFiles/articleAndPapers/Test_Automation_ROI.pdf,
retrieved electronically on August 18, 2010.

5 - Cisco, Cisco IOS Scripting with Tcl,
http://www.cisco.com/en/US/docs/ios/12_3t/12_3t2/feature/guide/gt_tcl.html, retrieved electronically
August 18, 2010.

6 â Bach, J., Do We Really Need all This Testing?,
http://www.quardev.com/content/whitepapers/do_we_really_need_all_this_testing.pdf, retrieved
electronically on August 18, 2010.

Nicely. I fixed some very minor typos while reading through it. On the style, you may want to consider to shorten the longer sentences in the beginning or splitting them up. Half-way through the language got more condensed to read, so there shouldn't be a problem.

On structure, you may want to point out more directly on your key points in the end in a bullet list, before diving into more details.

On "we build a solution, not software", I think James Bach notes in the principles for the context-driven school, that when the problem (of the customer) is not solved, the software is not working. You may want to make this more explicit.

Overall, I love this. Very nice.

contributed by  Markus Gärtner on Aug 30, 2010 11:17am

I think you have some great ideas in your chapter. You show a great deal of thought in it and make some solid observations. That is fantastic!

There are some suggestions I'd make stylistically to help you deliver your message. Some of these

Markus has already commented on, for example, breaking things out in a list before delving into details.

I like diving in quickly and have the same issue when I'm writing tech documents at the day-job (software tester.) This, in my mind, shows your passion for the topic as well as your familiarity and understanding of it. Having said that, I find that when I'm reading something, particularly if it is an idea that is new to me, if there are points to be expanded on or explained in detail, I can grasp the idea better if there is a list of points, followed by a discussion around those points. The tricky bit is remembering that your reader is being introduced to your ideas for the first time. I look at it as if I'm in a conference room full of people. I want to introduce the guest to each person in the room by going around with names, so you can identify them, then discuss each in turn.

OK - My list:

A. Sentence structure. Compound sentences can make a challenge to sort out what is being stated. Being more concise can draw the reader in and encourage them to finish the passage rather than skip to the next paragraph.

B. Currency. "Current Events" are perfectly reasonable in a blog, magazine article or conference presentation, in a book it can be dangerous. Use references to with care.

C. Spaces (after periods). Some of us of a certain age had typing classes where we learned to have two spaces after a period before starting the next sentence. Yeah, sometimes Word or other tools have "style filters" active, other times they don't.

D. But. The word "but" is dangerous. Have you ever heard "I like you but ..."? The "subtle" message here is "I don't like you..." and the real message follows the ellipses.


Last one. You reference "Your Money or your Life" (Robin, Dominguez and Tilford) in the body of the chapter. I would suggest it be listed with the other references at the end of the chapter. Even though you cite the title and authors in context, full bibliographical information at the end provides proper accreditation.

There are some other issues (do you mean gammut where you have "gambit?") and those will wait a bit.

Overall, there are some very good ideas here. I look forward to reading more.

Pete

I can be reached by email (peterwalen@msn.com) and I'm on Skype to chat if you'd like.

contributed by  *Pete Walen* on Aug 31, 2010 7:06pm

Good stuffs for a first-time author.

Would you allow me to offer some revisions on the 1st two paragraphs? If this is helpful, I can offer more on the chapter. Just compare this to your initial two paragraphs and see if you like it.


Each time a software development or testing project has started over the years, I've had a similar experience. Over time, I've noticed certain key factors that simply must be in for a testing initiative to be succeed. First must have our physical infrastructure needs covered: Server and client machines, networking equipment, internal/external hosting considerations, and they must be secured for the duration of the project. In addition to physical machines, we have to look environmental factors: lab space (especially for projects with external devices) and the need for adequate cooling. Our software to be tests will likely have numerous infrastructure requirements, including everything from Software Development Toolkits (SDKs) to Integrated Development Environments (IDE), but also file services, database services, web services, and deployment considerations from build to test group to general use. On top of all of this, there are also the specific testing areas (and the software testing tools that we use) to perform different types of testing: unit, functional, load, performance, and all those squishy things involved in human factors/user experience testing. When you think about it, the number of areas we need to configure and manage is dizzying!

Likewise I have witnessed a whole set of "process improvement" initiative designed to improve transparency, accountability, predictability, speed up delivery, or add testing coverage to areas that have not been adequately tested before. Many of these initiatives came with a great deal of fanfare


and cheering about how much time will be saved because of some new enhancement; a new tool that will automate all of the regression testing issues, a new server that will house a large battery of virtual environments, or a robust versioning and back-up system that will keep all versions of files updated, tagged, secure and available. These are admirable goals, and in many cases, when we as an organization implement these initiatives, the net result is that testing is more productive, and work can be completed more quickly. Yet all of these initiatives cost our organizations something. To set up any of these enhancements, our teams have to spend time, money, and attention to bring them to fruition. In our economic environment of budget squeezing and "belt tightening", I see companies looking for immediate savings, not investment. So more and more I see organizations 'soldier on', using tools already in place (if any exist) and machines that are already at work in their current state. Money does not go out the door, meaning immediate savings. What this â cost savingâ doesnâ t address, however, is the time trade-off made to perform these tasks. The Cost of Testing is not exclusively about money, very often we must include time in that cost analysis as well.

contributed by  [Matthew Heusser](#) on Sep 4, 2010 9:17am

Thank you everybody for the very solid and constructive comments. I'm in the process of revising the sections to make for "less wordy" sentences and to "tighten up" the earlier paragraphs.

_contributed by  [Michael Larsen](#) on Sep 7, 2010 5:15am _

First three sections have been revised. More to come. Thank you for all the feedback and comments, everyone, feel free to keep hacking and slaying :).

_contributed by  [Michael Larsen](#) on Sep 14, 2010 5:41pm _


I have given the first three sections a quick pass. Two major comments:

- simplify. too many words. i am finding the the points are all good, but, the they are buried deep in the text. as an example the first para of "What is time really worth?" could be more simply stated in far fewer words. i think people get the concept that time is money. state that. give a short example. done.
- introduction. a short paragraph with the premise of this essay would be great. get the reader into the mind set of what you are going to tell them. (the ole adage, "tell them what you are going to tell them, tell them, and tell them what you told them".


another point is overall length feels long. essay is about 5500 words at present. this is probably me just restating bullet point one is a different way.

i did not consider grammar yet. too early in the drafts. when the structure start to emerge i will get into grammar, at least as best that i can :)


what works for me is keeping the text real simple and adding words to create clarity. when there are too many words to take away i get lost in them.

contributed by  [Glenn Waters](#) on Sep 16, 2010 5:42pm


OK, I've taken all of the ideas suggested thus far and I've made changes. Please feel free to go through and let me know if more pruning or reshaping is needed :). Thanks again for the comments, they are really helping me.

_contributed by  [Michael Larsen](#) on Oct 1, 2010 10:22pm _

Nice job! Much better.

contributed by  [Pete Walen](#) on Oct 4, 2010 5:31pm

I just did a full review. See attached word document with change control on.

contributed by  [Glenn Waters](#) on Oct 8, 2010 11:59am

Thank you Glenn for your excellent review guidance. I hope I have met the letter and spirit of the recommendations.

Done!

_contributed by  [Michael Larsen](#) on Nov 15, 2010 8:51pm _