A Context Driven Approach to Releasing Quality Software

Authors:
Adam White – Manager, Test Engineering, PlateSpin
Steve Rabin – CTO, Insight Venture Partners

Software testing is part art and part science. It's a combination of looking at the components of a system holistically as well as in detail. Mainly – testing is about critical thinking and asking the right questions in order to get an answer that provides value and insight to stakeholders. Testing organizations and the people whose responsibility it is to validate software must balance the above and mix creativity with defined practices. Given the complexities and time constraints usually involved in software development cycles it is easy to lose sight of what is truly needed to validate a software product.

The objective of this document is to serve as a guideline for testers, who ship commercial software, to help ensure everyone involved in the software testing process understand what is required to be successful. The guidelines described throughout are not comprehensive and do not constitute a complete list. It is intentionally incomplete as it would be nearly impossible to create a complete list of every item that must be taken into account, from the R&D/Testing perspective. Since everyone's project environment (customers, dev teams, test teams, schedules, products, deliverables etc) is different, some of the following guidelines may work and some may not. Every team should decide what makes sense and modify the list as needed.

In some contexts the role of software testing is to manage confidence and report on risk. This means the role of testing is to ask the right questions in order to move the product in the right direction at the right time. By doing this, testers indirectly help improve the quality of the product. As will be seen, testing is very much a context dependent activity!

Over time, some of the basics of software testing may be taken for granted. Some might even have been forgotten. Because of this, these guidelines are also intended to act as a refresher.

What is testing?

There are many possible answers to this question. James Bach, co-author of Lessons Learned in Software Testing and owner of statisfice.com, a testing consulting and training company, defines testing as "Questioning the product in order to evaluate it."

Cem Kaner, author of Testing Computer Software, and co-author of Lessons Learned in Software Testing says in his Black Box Testing course "Testing is a technical investigation of the product under test conducted to provide stakeholders with quality-related information"[1]

In his paper titled The Ongoing Revolution in Software Testing Cem says that "Testing is investigation. As investigators, we must make the best use of limited time and resources to sample wisely from a huge population of potential tasks. Much of our investigation is exploratory – we learn as we go, and we continually design tests to reflect our increasing knowledge. Some, and not all, of these tests will be profitably reusable."[2]

[1] http://www.testingeducation.org/BBST

[2] http://www.kaner.com/pdfs/testingRevolution.pdf

What testing is depends on your context

It is entirely proper for different test groups to have different missions. A core practice in the service of one mission might be irrelevant or counter-productive in the service of another.[3]

Consider the two following project examples from www.context-driven-testing.com

One is developing the control software for an airplane. What "correct behavior" means is a highly technical and mathematical subject. FAA regulations must be followed. Anything you do -- or don't do -- would be evidence in a lawsuit 20 years from now. The development staff share an engineering culture that values caution, precision, repeatability, and double-checking everyone's work.

Another project is developing a word processor that is to be used over the web. "Correct behavior" is whatever woos a vast and inarticulate audience of Microsoft Word users over to your software. There are no regulatory requirements that matter (other than those governing public stock offerings). Time to market matters -- 20 months from now, it will all be over, for good or ill. The development staff decidedly do not come from an engineering culture, and attempts to talk in a way normal for the first culture will cause them to refer to you as "damage to be routed around".

The value of any practice depends on its context[4]

- Testing is done on behalf of project owners in the service of developing, qualifying, debugging, investigating, or selling a product. Entirely different testing strategies could be appropriate for these different objectives[5]. What testing activities can help you confidently say, "The product is ready to ship to customers?"

  Here are a few items that may help

  - Figure out what kind of coverage you intend to achieve (where coverage means "the models we have for the product, and the extent to which we have tested against them")?

  - Figure out what your oracles (where oracles means "principles or mechanisms by which we recognize problems")
    - I.E – where do you look, who do you ask/debate/question when you are

  - Asking the right questions can help project owners see the outcome of the project. If you find something that doesn't feel right then voice concerns early by reporting factual problems.
  - Articulate risks and impact of decisions on testing
  - Project owners are the ones who hold the key to quality.

    - If development slips and release date don't move provide a list of features that won't be tested to the project owners. Their views on what can slip and its impact may help you test more efficiently.

[3] www.context-driven-testing.com
[4] www.context-driven-testing.com
[5] www.context-driven-testing.com

There are other test activities that are important, too: modeling the product, determining oracles, determining coverage, choosing techniques, configuring the product, operating the product, observing the result, and evaluating the result

- Tests should become more challenging or should focus on different risks as the program becomes more stable[6]. Figure out what testing techniques and activities are important in your context
    - The following is a sample list of test techniques that can be applied during the project and release cycle.

        - Unit Test
            - No regressions caused, thread safety, code coverage, …
        - BVT (Build Verification Test)
            - A quick test or smoke test to make sure the product can be tested further
        - Full Functional Test
            - Do all functions in the product work as intended?
        - UI testing
        - Acceptance Testing
            - Does it meet user's acceptance criteria?
        - Regression Test
            - Is there consistent functionality across builds?
        - Load Test
            - Make sure that the product is not overwhelmed under stressful numbers of users or amounts of data
        - Benchmark - by build and/or platform
            - Other items may be network speeds, hardware, machine utilization.
        - Installation Test – by build and/or platform

        Note: where possible, consider automating repetitive testing tasks

        The techniques list above, although incomplete, can help give you a starting point when need to report on risk and help give project owners useful information. It's up to you to develop expectations for each of these items for your context. If anything you find something during testing that doesn't fit your expectations then you should figure out why. You may want to report anything you find as a risk to the project owners.

- Projects unfold over time in ways that are often not predictable[7].
    - Realize that requirements are always going to change.
    - Test plans are never truly complete.
    - Try not spending your time updating test plans. Spend more time designing, executing and running tests.
    - Test the product you have.
    - Different types of defects will be revealed by different types of tests-- Find different ways to exercise its functionality in a way that is meaningful to project stakeholders.

[6] www.context-driven-testing.com

Value added testing activities to help further confidence and report risk.

- Make your testing an intellectual activity. Good software testing is a challenging intellectual process[8]. Two ways you might do this is to get involved at the appropriate point of the development phase and push to get the software as soon as possible. Throughout the whole product development cycle be aware of the process you are working in and what your role is.

- Automated testing is not automatic manual testing: it's nonsensical to talk about automated tests as if they were automated human testing[9]
  - Testing tools and automation are important but not a replacement for knowing what to test and how to test it.

- Test artifacts are worthwhile to the degree that they satisfy their stakeholders' relevant requirements.[10]
  - Here are some ways you might provide value
    - Make test results traceable to test plans.
    - Make test results traceable to requirements docs.
    - Try to model how your software is actually used by customers. Techniques that may help are use cases or real-world scenarios or stories.

- Acceptance testing is another testing activity that can be wildly misinterpreted without clarification for the particular context. Michael Bolton, testing consultant and testing specialist, outlined at least 29 groups of people that acceptance testing could be targeted at.

  Michael brings up some interesting questions with regards to acceptance testing.
  - Who are the people producing the item being tested?
  - Who are the people accepting it?
  - Who are the people who have mandated the testing?
  - Who is doing the testing?

  According to Michael - “Acceptance testing is any testing done by one party for the purpose of accepting another party's work. It's whatever the acceptor says it is. The key to understanding acceptance testing is to understand the dimensions of the context.”

Knowing when to ship the software

Ultimately in a market driven company the decision to ship is a business one. So how does one provide the right information to help manage the project community’s confidence to ship software? There are several ways one can use to figure this out. The items below may or may not apply to your context, market driven or not.

- All critical and high priority bugs are fixed
- All areas of the product have been tested to some degree.
- Metrics
- Automation (if any) is reporting proper results.
- Ask your team!

---

[8] www.context-driven-testing.com
[9] www.context-driven-testing.com
[10] www.context-driven-testing.com

Here is the information for each of the parts

- No High severity issues outstanding
    - All high severity issues, that the development team is able to get done, requested by support, product management or other project owners are in the product and tested
    - The process of keeping track of these issues is managed in combination by support, product management, project management, development and test engineering teams.
- All areas of the product have been tested to some degree.
    - A testing dashboard or risk matrix is a great way to provide this information. It could be as simple as a spreadsheet that contains each feature listed in the left column and each build number across the top. Each cell has 3 (or more) status codes - Green, Red and Grey. Green means no issues have been discovered with that feature. Red means there is a problem and includes the bug number in the cell. Grey means that there is more testing or investigation to be done before a reliable status can be done. For a detailed presentation on this see footnote.[11]
    - The usefulness and priority of the risk matrix varies during the project life cycle. The information is always available but we don't want information overload the project community. Near the beginning of the project the interest in the ability to ship the software is generally low so we don't send the document out to broad audience. The week or two before release is a different story. There is a very high interest in this information in the last stages of the product so we provide the document on a daily basis.
    - Providing this information can help the test team achieve visibility and can help achieve buy-in at all levels of project involvement. This helps the test team stay away from becoming the "Quality gatekeepers".
- Automation reports proper results
    - Is your automation focused on the right areas?
    - Can you use our own product to test your own product?
- Requirements verified
    - Walk through the requirements and make sure we have test case coverage for each item
- Metrics
    - Number of customer issues successfully fixed is a good one to start with.
    - Bug rates during the project may provide insight - if all variables stay constant throughout the entire cycle (which rarely happens)
        - The key is that someone has to provide context to the stats or else the numbers/charts/graphs can be wildly misinterpreted.
    - Test case coverage
        - This is a very dangerous metric even when not used in context of the other heuristics. It is also very dangerous to continue to aim for % coverage when the product is growing, testing resources change. Reporting this metric can make those who don't understand testing feel really good about bad software
- Ask your team!
    - A very good way to figure out if you are done testing is to ask them "Can we ship?" or “Would you feel comfortable giving this to customers?” or “Does this product meet the goals for the release.” You will be able to tell their confidence with the product and how it ties back into their own work. The answers you get

[11] www.satisfice.com/presentations/**dashboard**.pdf

from your team can tell you a lot about how well communication went during the project.

If you report on each of the items by themselves then they do not hold much value. When this information is combined it can offer an insight that can be very useful for the all teams involved in the product. The whole is more than a sum of its parts.

Why do these items help in some market driven contexts?

- You can get your product to market quickly – maybe even first with new products. It will help you maintain market leadership with current products if you can respond quickly to customer concerns and deal with any missed bugs/features in a timely fashion

- To keep pace with the market, a lightweight approach to test documentation is needed because product changes are very dynamic. Documenting every step required to run a test will most likely be a waste because tomorrow - some of those steps won't be around.

- In a market driven company information is key. Keeping an eye on these areas, or possibly more or less areas, helps keep information available and emphasizes information flow

What are the risks of shipping software this way?

- All signs can point to go but you might have overlooked something important. The test team might not have found a major bug but your most prized customer does after you ship.

- You can ship too soon. Shipping software is a tradeoff between breadth of vs. depth of product coverage. You have to think about what your software if doing. If you are not comfortable with your ability to find the risks then this ideas might be a good starting point for you.

- You can ship too late. If you are measure the wrong things with your metrics or testing the wrong pieces of functionality, manually or automated, you may discover that you don't make much progress and your release may get delayed unnecessarily.

Software testing is an essential part of the development process. It should be integrated into every aspect of the software development cycle – from initial design/requirements planning, through coding, system validation and on-going support/maintenance. While specific tests may be more or less important for a given application, the principles of software testing remain unchanged.

The guidelines described throughout this document are not meant to be domain specific. The goal was to give you something that could work well across all types of applications including on-line, network centric, business infrastructure and thin client, thick client, web services heavy architectures. What gets tested, who does the testing, how the tests get performed and the communication mechanisms employed to manage the validation process must be taken into consideration regardless of the purpose of the system or the deployment environment.

Testing a piece of software is not a static event if the software is successfully is to be deployed and used. New releases (major and minor), enhancements and patches are a regular part of the software's lifecycle so testing is an on-going event. Features are added and removed from software based on need so there is no such thing as a steady state. Given the many moving

parts and associated complexities of software it's recommended that every test team develop their own set of testing ideas to validate their specific software product.

Resources

Books

Title: Testing computer Software
Authors: Kaner, Falk, Nguyen

Title: Lessons learned in software Testing.
Authors: Kaner, Bach, Pettichord

Online

www.bettersoftware.com
www.satisifice.com
www.developsense.com
www.kaner.com

Training

Black Box Software Testing
http://www.testingeducation.org/BBST/index.html

Rapid Software Testing
http://www.satisfice.com/seminars.shtml