SQA for New Technology Projects

by James Bach
james@satisfice.com
www.satisfice.com

When you're developing a product that involves elements with which you have little experience, such as new technology, you can expect many problems to arise that you did not anticipate. Many of these problems may require new tools, new skills, or substantial time to solve.

A development process that is optimized to handle problems you understand may be wrong for problems you don't yet understand. That's why companies that routinely work with speculative projects tend to use a more exploratory, risk-oriented, incremental form of development, as opposed to one that is emphasizes pre-planned, predictable tasks.

The key to working in unfamiliar territory is to change your project management focus from *task fulfillment* to *risk management*. The project team should continue to devote attention to planning and fulfilling project tasks, but that should not be the *focus* of the project manager or test team. Instead, the project manager must assume that any given prediction, at any given time, may be wrong.

This document consists of two sections: Project Strategies and End Game.

Project Strategies

A risk managed project implies specific strategies to deal with that assumption. Here's my list:

- **Catalog areas of potential uncertainty and other risk.**

  Produce a risk list for the project. If not a printed list, any leader on the team should be able to tell you what the major risks are. All project leaders should be in general agreement about the list.

- **Foster a sense of caution and urgency concerning risk areas.**

  The team should practice talking about risk areas, rather than discouraging them as "negative thinking." The risk that you don't discuss is the one that you also won't guard against. However, you should also stress that a new technology project requires you to be bold and to accept a certain amount of risk. The real problem is when there is an inconsistent attitude about risk, across the team.

- **Track the status of risk areas throughout the project.**

  There should be some process whereby anyone on the project can learn about the current status of risk areas. There should be frequent project reviews (every few weeks) that involve a level of management one step higher (at least) than day-to-day management.

- **Establish controls to manage the introduction of new risks.**

  There should be a change control process that prevents the careless introduction of new risk drivers after the scope of the project has been set. A clear charter for the project can help in limiting scope creep. Too much change control, too early, can also hurt the project. Consider establishing a progressive change control strategy, whereby the closer the product is to shipping, the stricter changes are controlled. I like to call this the highway/city strategy: when we're on the highway, we drive fast. When we transition to city streets, we go slower. When we are parking, we drive very slow. As we get closer to obstacles, we must give ourselves more time to think and react. The same goes for changes in technology products.

- **Limit the scope and extent of project commitments until the risks are mitigated.**

  Avoid specific promises to upper management about revenue dates, or anything else, unless you are in a position to fulfill those promises without compromising the project. Until the risks of the project have been explored and mitigated, such promises are premature. Also, look for an exit strategy that allows the project to be cancelled if it becomes apparent that the risks are too large relative to the rewards.

- **Minimize coupling between risky components and the rest of the product.**

  For each risk driver, look for a strategy that will allow that element to be dropped from the project if it turns out to be too difficult to solve the problems that come up. Consider moving especially risky components to a follow-on release, or making them optional to install. I call this an "ejection seat" strategy. If risky components cannot be ejected, they can drag the whole project down.

- **Develop and test in potential risk areas sooner than other areas.**

  Adopt a prototyping strategy, or some other way that risk drivers and risks can be explored while there's still time to do something about them, or it's still early enough to cancel the project without huge losses.

- **Shift test effort to allow closer examination of risk areas.**

  Adopt a test management process that dynamically reallocates test effort to focus on risk areas, as risk areas emerge. It helps to have an explicitly risk-based test plan. Look for strategies that minimize the amount of time between defect creation and resolution.

- **Staff well to meet the challenges.**

  The project should have a larger test team than normal, in order to react quickly to new builds. The development team does not have to be larger, in fact many say it should be smaller, but it must have the ideas, motivation, and skills needed to do the job. Special training or outside support should be considered.

- **Do the project in evolutionary stages.**

  Craft a project plan that includes iteration and synchronization. Think of it like rock climbing or crossing a fast river by jumping from rock to rock: plan ahead, but revise the plan as new challenges arise and new information comes to light. Don't let the bug list spiral out of control, but rather fix bugs at each stage. You might have to scrap some component of your technology and re-design it, once you have some experience with it in field testing. Don't be surprised by that. It's perfectly normal, and it's the people who refuse to rewrite bad code who suffer more, in the end.

If you're using explicit iterations to plan the project, be sure that your iterations are complete. That means you go through a test-triage-bug fix-release end game cycle for each iteration.

- **Maximize testability at design time.**

  Pay attention to how the product will be tested. Assure that the designers make every reasonable effort to maximize the controllability and visibility of each element of the product. These can include such features as log files, hidden test menus, internal assertion statements, alternative command-line interfaces, etc. Sometimes just a few minutes of consideration for the test process can lead to profound improvements in testing.

- **Apply a diversified testing strategy.**

  Don't put all your eggs into the risk-based basket, because you may inadvertently overlook an area that was risky, just because you didn't anticipate that risk. So, devote a portion of your test strategy, maybe a third, to a variety of test approaches that are not explicitly risk-based. Also, establish a process for encouraging new approaches to be suggested and tried.

- **Let the test team be a hub of for all product quality information.**

  Look for ways of making contact with the field about the emerging product. In speculative projects, problems often emerge that are invisible to the developing organization, yet obvious to customers. Also, assure that technical support, product management, and technical writers are well connected to the test team, so they can feed information to the testers without too undue effort.

- **Allow extra time to do the work.**

  Establish a responsible schedule padding strategy. It's impossible to predict how long things will take when you're doing something new, so more time should be allocated than can be specifically justified based solely on experience. Public padding on the schedule is better than secret padding, as long as you also are public about using the pad, and take reasonable steps not to use it lightly.

  One way to formalize a padding strategy is to use a "straw, tin, and steel" progression of project plans. We instituted this at Borland as a way to float real plans without over-committing to particular schedules. The systems works like this: the first written version of the project plan (or the test plan) is called the straw plan, because it's a "straw man" plan. The protocol of the straw plan is to state a revenue date to the best of your ability to justify, then add 75% to that to get a "no later than" revenue date. After the team is fully staffed and committed, produce a tin plan, and add 50% to your best estimate for a revenue date to obtain the committed date. The steel plan, produced when all functions are complete and the developers are wholly devoted to fixing problems, uses a 25% pad. The pad percentages are based on projects with a predicted duration of about 6 months.

- **Build consensus about acceptable quality.**

  Establish a systematic discussion, near the end of the project, about the status of quality and the acceptable quality level. This should be a consensus building process among the people who control whether and when the product will be released. It should include a comprehensive triage of known problems. Criteria for release should be discussed and amended to the satisfaction of project stakeholders. By the time the product is released, the goal is to assure that the project team is agreed about the status of the product and the pros and cons of shipping it versus continuing to improve it. Failing to achieve consensus creates the risk that the team will lose its commitment and cohesion, which leads directly to poorer quality and other problems in future projects. Failing to attempt to

achieve consensus practically guarantees that critical information about problems will not be freely shared with management.

- **Gather data about the process for future use.**

  Look for records of schedules, schedule slips, meeting notes, and decisions made. Any metrics or general information that can be collected unobtrusively will help you understand the dynamics of the new technology and the team. That, in turn, helps plan the next project.

End Game

The end game begins in the last third of the project, or whenever the development team stops creating new functions and turns its attention to bug fixing, performance enhancement, and fit and finish tasks. A big part of QA for risk-managed projects is handling the end game. That includes the following elements:

- **Mission Status Monitoring**

  Establish a dashboard or some other way of summarizing and updating the status of the project against the mission. In the final third of the project, keep that status updated no less frequently than every other day. The status that matters most is status against the mission that motivates the project. Often, in new technology projects, the mission must be interpreted and reinterpreted many times over the course of the project. Pre-defined ship criteria can be helpful, but it's rare that a team can know what those criteria should be, months before shipping.

  I suggest using the following dashboards: schedule, critical defect metrics, test status, issue list.

- **Re-Scheduling**

  In new technology projects, you need more schedule slack. It's still a good idea to think carefully about the schedule and the project plan, but even so, you will need the slack. That means more schedule padding built into the schedule, or more willingness to slip the schedule outright. There is no technology or technique that will allow you to accurately predict the schedule a complex project that includes unfamiliar elements like new technology. The implication for the end game is that you continually monitor the possibility that won't be able to achieve a good enough product in the allotted time. It helps to prepare upper management for this eventuality.

  There is alternative to accurate schedule prediction, however— schedule control. You can adopt a "ballast" strategy to control the schedule. That means you plan ahead to drop or postpone certain features of the product that can be cleanly separated from the main product. Dropping them lessens the amount of work to develop and test. The key to this strategy is assuring that the developers don't accidentally couple the ballast features so strongly to the core product that there is no easy way to drop them.

  Another strategy is "quick point-release", whereby you release 1.0 with a lot of problems, followed a month or six weeks later by a point release, possibly slip-streamed, that fixes those problems. Sometimes that can be the only way to meet a revenue date while minimizing the overall impact of a not-quite-ready product on the market. The theory behind quick point-release is that the people who buy products first are early adopters who are a little more tolerant of problems than people who buy later.

- **Team Coordination and Consensus Management**

  Obviously it helps if the team is focused on the same, clear set of goals, and is working toward them in a harmonious fashion. If there are an especially large number of new elements to the project, however, the team needs to establish an especially elaborate method of staying coordinated. One method I practice is a daily morning meeting, no more than about 15-minutes long, to go over the status and assure that everyone is on the same page about what to do and how important it is. At that meeting, we read and confirm the contents of the various project dashboards. Any issues are taken offline, rather than discussing them there. I make the meeting compulsory during the end game, and make sure that it does not exceed its allotted time.

- **Problem Triage and Troubleshooting**

  Problems happen. You can't control that. So, you need a process to identify problems, decide how to react, and dispatch them to be solved. On the project level, this is a straightforward project management function. On the product level, the standard for challenging projects is to establish a triage meeting whereby the only defects allowed to be fixed are those that have been approved by triage.

  This sounds easy, and it is pretty easy unless you have a lot of defects to review or a large project team. On a project with 50 or 100 new problem reports every day, a fairly elaborate process is required for triage. On a project divided into 10 sub-teams, there will have to be a lot of people involved in the triage meetings. But, the basic triage concept is just this: fixing takes time, ties up resources, and may create new problems, so let's be sure that the major stakeholders in the project are in agreement on how we should proceed.

  To set up a triage meeting, determine who are the key people who decide if the product is ready to release. That will be the triage team. One person, usually the test manager, runs the meeting. The agenda of the meeting is to visit each new problem that qualifies for triage (usually the higher priority defects), discuss the importance of the problem and the difficulty of fixing it, make a decision, and move on.

  Another part of the triage meeting to is revisit those defects that have been approved for fix but have not yet been fixed. Sometimes it's a good idea to defer a defect if it's proving too difficult to fix.

  Bear in mind that the decisions of the triage meeting, unless otherwise stated, should be strictly followed with regard to *not* fixing a defect, but they should be taken as *advisory only* when it comes to fixing. That's because only the developer can judge if it is truly feasible to safely fix any particular defect.

- **Change Control**

  The triage process is a form of change control. Additionally, I've found it useful to establish a separate Change Control Board meeting. This group consists of all the managers on the project. They meet as needed to consider specific proposals for modifying the feature set of the product.

  There's an important tradeoff between controlling change and allowing improvement. One way to manage that tradeoff is by using a so-called "funnel" strategy of progressive change control. Let change happen, early on. Encourage it. Minimize that bureaucracy. But as you get closer to release, slowly increase the control. You can do this in a subtle way by getting more conservative about what defects you allow to be fixed. You can do it overtly by choosing a progressive more conservative set

of “freeze” rules. An example of one such progression is: UI freeze, partial function freeze, full function freeze, partial code freeze, full code freeze, and showstoppers only.

- **Field Testing**

  Field testing (AKA beta testing) is a good test technique in any project, but in a new technology project it’s especially important. If there are a lot of unknowns about new features, performance, and compatibility, you need to know about that soon enough to make adjustments. Field testing is a major part of connecting to customers.

- **“Level 3” Testing**

  One popular and straightforward way to classify tests is a system of three levels: level 1 tests establish that the major functions of the product can work, and level 2 tests establish that major and minor functions of the product work together, in a variety of situations. Level 3 refers to those tests that target performance, usability, error handling and recovery, installability, reliability under high loads, long term reliability and compatibility. Level three seeks to test something close to the final system under something close to field circumstances, but also examines what happens when the system is pushed to it limits. By its nature, level 3 testing is difficult to do early the project. That makes it a standard challenge of end game test management.

- **Product Fit and Finish**

  In the end game, attention must widen from major features and architectural issues to the overall presentation and experience of the product. Lots of little problems must be ironed out. The further you get into the end game, the less you want to spend your time on little problems, so it helps to invoke “fit and finish” to refer to all little problems, all at once. It may not be important to fix one little cosmetic bug, but it may well be important to fix a dozen of them.

  Another part of fit and finish are the utility functions and features that go along with the product. These include example files, readme, installation, compatibility, performance, help files, etc. Make a checklist of all items like these to assure they aren’t forgotten.

- **Project Closure**

  Project closure is the end game of the end game. Closure includes the following elements: recall plan, file and license checking, signoff, hand-off to manufacturing, retrospective. The recall plan is your plan for what to do if a huge problem is discovered in the product after signoff and before the product reaches store shelves. File and license checking means assuring that the right files are on the disk and that you have the right to distribute each of those files. Signoff is the formal process of getting the stakeholders agreement to release the product. Hand-off to manufacturing formally transfers control of the project to the people who will package and sell it. A retrospective is a meeting of the project team to discuss what happened and how well things worked.