August 11, 2010

Seven Pragmatic Practices To Improve Software Quality

by Margo Visitacion and Mike Gualtieri
for Application Development & Delivery Professionals

FORRESTER® Making Leaders Successful Every Day

August 11, 2010

Seven Pragmatic Practices To Improve Software Quality

Incremental Changes Can Have A Big Impact On Quality

by Margo Visitacion and Mike Gualtieri
with Phil Murphy and Adam Knoll

EXECUTIVE SUMMARY

All software has bugs. Application development teams do the best they can to avoid and fix them. But developing quality software is hard. Why? Virtually all application development teams are squeezed by the need to develop and change applications faster, but few have the time to implement a full-blown, guru-based quality program. What they need are shortcuts — pragmatic changes that application development teams can make to improve software quality without getting the brass and rank and file in a tizzy. Forrester has identified seven pragmatic changes that can have a big impact on improving the quality of the software you deliver to the business: 1) define quality to match your needs; 2) broadcast simple quality metrics; 3) fine-tune team and individual goals to include quality; 4) get the requirements right; 5) test smarter to test less; 6) design applications to lessen bug risk; and 7) optimize the use of testing tools.

TABLE OF CONTENTS

NOTES & RESOURCES

This research is based on client inquiries, existing Forrester research, and interviews with dozens of application development professionals and technology vendors.

Related Research Documents

"The Testing Tools Landscape: 2010"
April 27, 2010

"The Top 5 Changes For Application Development In 2010"
January 4, 2010

"The Dawn Of Dynamic Software Quality Assurance"
February 2, 2009

YOU ALREADY KNOW THE IMPORTANCE OF QUALITY

But how do you improve quality without asking for more "resources," extending already tight development schedules, and fully implementing comprehensive testing tool suites? Quality gurus often promote a Don-Quixote-like quest: Make sweeping changes to implement "real" quality with an approach gleaned from manufacturing processes. However, this approach has two problems: 1) These changes are generally impossible to implement, let alone sustain, and 2) software development is not like building a car on an assembly line.

Forrester defines quality software as:

> *Software that meets business requirements, provides a satisfying user experience, and has fewer defects.*[1]

Many enterprise application development teams invest in tools, processes, and people, yet many still struggle to improve quality. What you need is pragmatic advice to improve *software quality* that won't:

- **Break the bank.** It's wonderful if you already have effective quality assurance (QA) professionals and testing tools in place. If you don't and the break-fix cycle is costing you too much time and money, you need to improve quality. However, in today's new normal of lean and mean, going to your boss to ask for headcount (if you dare) or budget to buy testing tools is a daunting proposition.[2]

- **Cause a revolt within your team.** Sweeping process changes that add new tasks to already overburdened staff may turn your team into a pack of passive-aggressive developers. Despite their popularity, the best-selling quality gurus don't get that you can't just raise your magic wand to implement a new process.

- **Slow down your process.** Telling the business that it will take longer to get things done is a good way to get fired or sent to your firm's version of Siberia.[3] Business partners may accept a slight delay to prevent high-risk issues, but application development professionals have to be fast — and effective.

SEVEN PRAGMATIC PRACTICES TO IMPROVE SOFTWARE QUALITY

Quality cannot be sprinkled onto an application right before it gets exposed to your clients.[4] Rather, it must be a part of the entire software development life cycle (SDLC) from inception through implementation. As such, responsibility for quality falls squarely on the shoulders of the application development manager — it is *not* solely the responsibility of QA professionals.

Inject seven pragmatic practices into your existing SDLC to improve the quality of the software your team develops and delivers (see Figure 1).

Figure 1 The Practice, Roles, And Impact

Pragmatic practice	Impact on quality	Benefit	Relevant roles
1. Define quality to match your needs.	• Meet business requirements • Achieve a satisfying user experience	Your ability to achieve quality is improved because the application development team is not charged with unrealistically perfect expectations. Rather, it is chartered with a definition of quality that fits the given time, resource, and budget constraints.	• Business stakeholders • Entire application development team
2. Broadcast simple quality metrics.	Reduce defects	Highly visible metrics keep quality top of mind for the entire team and expose when efforts fall short.	Entire application development team
3. Fine-tune team and individual goals to include quality.	• Meet business requirements • Achieve a satisfying user experience • Reduce defects	Team members perform according to their incentives; therefore, making quality improvement part of their goals reinforces desirable behavior.	Management
4. Get the requirements right.	• Meet business requirements • Achieve a satisfying user experience	Less rework means less retesting and fewer cycles, which greatly reduces overall effort.	• Managers • Business analysts • User experience designers • Architects
5. Test smarter to test less.	Reduce defects	Focusing testing on the most-important and riskiest areas ensures that they receive the lion's share of test resources and that any bugs that slip through are likely to be confined to the least-important features.	• Quality assurance • Managers
6. Design applications to lessen bug risk.	Reduce defects	Simpler, cleaner designs result in code that is simpler, cleaner, and easier to test and rework — which means that the code will have fewer bugs and that those bugs will be easier to diagnose and repair.	• Architects • Developers
7. Optimize the use of testing tools.	Reduce defects	Automation frees resources from mundane testing to focus on the highest-priority tests and increases test cycles' repeatability.	• Quality assurance • Developers

56939 Source: Forrester Research, Inc.

Improvement No. 1: Define Quality To Match Your Needs

Everyone wants perfect software, but budget constraints, business priorities, and resource capacity often make "perfect" an impossible goal. But if perfection isn't your goal, what is? Recognize that the goal of testing is to mitigate risk, not necessarily eliminate it. Your applications don't need to be perfect — but they do need to support your business processes in time to leverage new opportunities without exposing companies to *unnecessary* or *untenable* risk. Therefore, your definition of quality may vary by application. As you initiate a project, get the right roles involved to ask the right questions: What constitutes perfect versus good enough versus unacceptable?

- **Benefit:** Your ability to achieve quality is improved because the application development team is not charged with unrealistically perfect expectations. Rather, it is chartered with a definition of quality that fits the given time, resource, and budget constraints.
- **Impact on quality:** This improvement will help you meet business requirements and achieve a satisfying user experience.
- **Relevant roles:** Business stakeholders and the entire application development team will need to implement this practice.

Improvement No. 2: Broadcast Simple Quality Metrics

Teams want to win at quality, but to win they have to be able to see the score — the measures that indicate that quality is worth pursuing. Additionally, they must see the impacts to that score — what makes it go up and what makes it go down. You don't need overly complicated metrics models to demonstrate your team's improvement, but it's imperative to make sure that the metrics you do provide are clear and straightforward. If they are too complicated, make them simpler. If you don't have metrics, then start with some simple ones such as total number of defects, defects by phase or origin, and severity of defects by phase; all of these metrics are measurable by the different roles on your team.

If quality measures already exist, don't hide them; publish them. Ensure that everyone can see them at any time and make sure staff members understand how they are measured. For example, HP has a monitor installed in its coffee room that displays a defect dashboard for all to see.

- **Benefit:** Highly visible metrics keep quality top of mind for the entire team and expose when efforts fall short.
- **Impact on quality:** This improvement will help you reduce defects.
- **Relevant roles:** The entire application development team will need to implement this practice.

Improvement No. 3: Fine-Tune Team And Individual Goals To Include Quality

Whether the incentive comes in the form of a carrot or a stick, teams can change behavior when their leaders challenge them to excel. If quality is not in your staff's current goals, introduce it incrementally until you reach the desired level of achievement. It's critical to measure improvements that are simple and transparent to the entire team; all team members have to know when the ball goes into the net and when it doesn't.

- **Benefit:** Team members perform according to their incentives; therefore, making quality improvement part of their goals reinforces desirable behavior.
- **Impact on quality:** This improvement will help you meet business requirements, achieve a satisfying user experience, and reduce defects.
- **Relevant roles:** Management will need to implement this practice.

Improvement No. 4: Get The Requirements Right

Changes in requirements later in the project can lead to rework and poorly conceived architectural changes that result in unstable code. Iterative code delivery isn't exempt from the problem; Agile approaches may simply deliver poor code even faster and greatly increase testing volume. Keep your business analysts and QA professionals actively involved with change-and-build processes to limit the impact of requirements changes.[5]

- **Benefit:** Less rework means less retesting and fewer cycles, which greatly reduces overall effort.
- **Impact on quality:** This improvement will help you meet business requirements and achieve a satisfying user experience.
- **Relevant roles:** Managers, business analysts, user experience designers, and architects will need to implement this practice.

Improvement No. 5: Test Smarter To Test Less

This practice improvement may sound like a challenge directed toward quality assurance; however many teams burn valuable time testing too much. For example, developers may spend time writing JUnit tests for code that is not very prone to risk or performing "code coverage" testing for very stable and mature code. QA professionals may test every conceivable feature rather than the ones that have the most impact on updated code. Risk-averse managers may also insist that everything be tested — regardless of the cost. In these scenarios, a testing extravaganza ensues that saps resources away from fully testing the really important features. Identify test cases based on the functionality's newness or risk level. Mission-critical software where defects can cause loss of life should be tested extensively; however, if you are writing your umpteenth CRUD (create, read, update, delete)-style application, adjust your testing to match the exposure to risk.

- **Benefit:** Focusing testing on the most-important and riskiest areas ensures that they receive the lion's share of test resources and that any bugs that slip through are likely to be confined to the least-important features.

- **Impact on quality:** This improvement will help you reduce defects.

- **Relevant roles:** QA and managers will need to implement this practice.

Improvement No. 6: Design Applications To Lessen Bug Risk

Architectural complexity, spaghetti coding techniques, and poor design all increase the likelihood that your application will contain bugs. Mitigate that likelihood with better design principles such as separation of concerns, frameworks, and design patterns to reduce design complexity and the likelihood of bugs in your code.

- **Benefit:** Simpler, cleaner designs result in code that is simpler, cleaner, and easier to test and rework — which means that the code will have fewer bugs and that those bugs will be easier to diagnose and repair.

- **Impact on quality:** This improvement will help you reduce defects.

- **Relevant roles:** Architects and developers will need to implement this practice.

Improvement No. 7: Optimize The Use Of Testing Tools

Automation can cut unnecessary time and therefore cost out of the SDLC.[6] Automating test scripts can increase repeatability and accuracy, freeing resources to focus on higher-risk testing and exploratory testing that is performed manually or with strong manual involvement. Automation helps QA and development professionals plan test cycles more effectively by allowing them to prioritize and measure the risk of testing requirements (in test management) and determine what can and should be automated for repeated test execution. Over time, teams should build libraries of test cases based on core business processes or models of business processes that can be modified without having to rewrite the entire script. Automating testing is particularly useful for teams using Agile processes, because they are apt to test more often, across many iterations.

- **Benefit:** Automation frees resources from mundane testing to focus on the highest-priority tests and increases test cycles' repeatability.

- **Impact on quality:** This improvement will help you reduce defects.

- **Relevant roles:** QA and developers will need to implement this practice.

WHAT IT MEANS

PRAGMATIC PRACTICES CAN IMPROVE OVERALL QUALITY

Archimedes said, "Give me a lever long enough and a fulcrum on which to place it, and I shall move the world."[7] Your fulcrum is your SDLC, and Forrester's pragmatic practices offer the levers that applications professionals need to make wholesale quality improvements. As you implement these pragmatic practices, remember that:

- **Quality is a team sport; you need to get everyone playing it.** Quality must move beyond the purview of just QA professionals to become an integrated part of the entire software development life cycle (SDLC) to reduce schedule-killing rework, improve user satisfaction, and reduce the risk of untested nonfunctional requirements such as security and performance. Managers must make quality measurable and incent all roles on the team to improve it.
- **Teams need to do the right things — and do them better.** The fish stinks from the head, and so does quality. The fish of software development is a combination of requirements and design: Getting these right can have a big impact on the future quality of the application. A thoughtful design helps teams avoid unnecessary complexity that may lead to more bugs. Poor requirements lead to the all-too-well-known problem of delivering the wrong software and not meeting user expectations or true requirements. Additionally, teams should be sure to remember to apply this thoughtfulness to nonfunctional requirements that can affect performance, scalability, fault tolerance, and security.
- **Testing tools can make everyone on your team more productive.** Testing tools are a critical element of what app dev teams need to be effective, especially when multiple roles are involved in the testing effort. There are suites that offer test management, functional test automation, and even some nonfunctional testing tools such as stress testing tools. But implementing these tools can be challenging and can add unwelcome process steps to your SDLC. Use testing tools, but choose to implement only the tools that will have a measurable impact on improving software quality.

ENDNOTES

[1] Forrester defines user experience as "users' perceptions of the usefulness, usability, and desirability of a Web application based upon the sum of all their direct and indirect interactions with it." See the September 4, 2009, "Best Practices In User Experience (UX) Design" report.

[2] Forrester published a report that notes that after the recession, the new normal will remain "lean and mean." See the January 4, 2010, "The Top Five Changes For Application Development In 2010" report.

[3] Siberia is actually a lovely place if you are fond of the beautiful, rugged landscape. However, Siberia has also become synonymous with penal labor camps set up by the former Soviet Union's government agency known as the Gulag. Source: Wikipedia (http://en.wikipedia.org/wiki/Gulag).

[4] Inserting quality into each phase of the SDLC enables teams to determine not just the application's potential but also its testability and ability to meet customer expectations. It also enables teams to prepare production support for potential issues. See the February 2, 2009, "The Dawn Of Dynamic Software Quality Assurance" report.

[5] Organizations can better focus on customer needs and business process requirements by deepening their commitment to developing strong requirements and by involving customers in design and testing. See the September 4, 2009, "Best Practices In User Experience Design" report, and see the February 3, 2010, "Best Practices: Your Ten-Step Program To Improve Requirements And Deliver Better Software" report.

[6] Testing tools are not just for functional testing anymore. Today's testing tools cross the SDLC, enabling teams to push quality to the forefront of requirements, design, and development processes rather than hoping to catch all possible defects during the testing phase. See the April 27, 2010, "The Testing Tools Landscape: 2010" report.

[7] Archimedes was a renowned mathematician and inventor in ancient Greece, 280 to 211 BC.

FORRESTER®

Making Leaders Successful Every Day

Headquarters

Forrester Research, Inc.
400 Technology Square
Cambridge, MA 02139 USA
Tel: +1 617.613.6000
Fax: +1 617.613.5000
Email: forrester@forrester.com
Nasdaq symbol: FORR
www.forrester.com

Research and Sales Offices

Forrester has research centers and sales offices in more than 27 cities internationally, including Amsterdam; Cambridge, Mass.; Dallas; Dubai; Foster City, Calif.; Frankfurt; London; Madrid; Sydney; Tel Aviv; and Toronto.

For a complete list of worldwide locations visit www.forrester.com/about.

For information on hard-copy or electronic reprints, please contact Client Support at +1 866.367.7378, +1 617.613.5730, or clientsupport@forrester.com.
We offer quantity discounts and special pricing for academic and nonprofit institutions.

Forrester Research, Inc. (Nasdaq: FORR) is an independent research company that provides pragmatic and forward-thinking advice to global leaders in business and technology. Forrester works with professionals in 19 key roles at major companies providing proprietary research, customer insight, consulting, events, and peer-to-peer executive programs. For more than 27 years, Forrester has been making IT, marketing, and technology industry leaders successful every day. For more information, visit www.forrester.com.

FORRESTER®