Failure is becoming the norm (and what you should do about it)

David Gelperin
david@clearspecs.com
480-296-3559

Abstract:

The software industry has a major quality problem. There are project and enterprise tactics for solving this problem.

"… 43% of companies worldwide have reported breaches in the past year…"

USA Today 12/7/2014

"… the malware that was used [in the Sony breach] would have … probably gotten past 90% of internet defenses that are out there today in private industry …"

Joseph Demarest, assistant director of the FBI's cyber division 12/12/2014

"Sale crashes Southwest Airlines website"

Fox News (6/3/15)

Most failures don't make the news and are caused by quality rather than functional deficits.

Some software systems have significant quality deficits. For example, some systems can't be reliably changed after a few years. These deficits are self-inflicted and becoming systemic as Agile is more widely adopted.

This paper describes quality deficits and tactics for reducing them. It specifies what must be done to make failure a rare exception.

Keywords: quality attribute requirement, quality goal, quality deficit, quality awareness, quality-aware tactic, quality knowledge base, quality achievement strategy, quality verification strategy

Taxonomy Keywords: Requirements (D.2.1), Methodologies (D.2.1.e), Software Verification (D.2.4), Software Quality (D.2.19)

Part 1: System failures and quality deficits

We refer to quality attribute requirements ("ilities") as "quality goals". These include security goals, dependability goals, and understandability goals. "Quality deficits" refer to the undefined or unachieved quality goals for a software system.

The scope and nature of the software quality problem is suggested by a number of factors.

A number of systems have been breached and others have failed under heavy loads, suggesting that some quality goals, are difficult to achieve and verify. They are difficult to achieve and verify for many reasons including the fact that quality goals have little in common with functional goals (Figure 1).

© 2015 ClearSpecs Enterprises

Figure 1: A comparison of quality goals and functional goals

Qualities	Functions
• are rarely part of the rationale for the application	• are the primary rationale for the application
• are detailed by architects, designers, or developers	• are elicited from customers and application and domain experts
• are pervasive properties	• are local properties
• affect the choice of architecture or the implementation of multiple domain functions i.e. pervade the code	• affect individual modules i.e. are localized in the code
• compliance can occur at different levels e.g. high, medium or low security, or be binary i.e. yes or no	• compliance is binary
• can only be fully demonstrated within later iterations	• can be fully demonstrated within an iteration
• require analysis, technical reviews, and measurement along with testing for adequate verification	• can often be adequately verified with thorough testing alone
• change infrequently	• may change frequently
• may not be changed at a reasonable cost late in the project	• may be changed at a reasonable cost late in the project
• are achieved using a set of supports that are quality and degree specific	• can be decomposed into subfunctions
• may support or conflict with one another	• are often independent of one another
• implementation may entail reuse of system function modules (from libraries) within and across iterations e.g. reuse of exception handling module	• reuse is limited to similar applications
• cost of delayed identification can be significant due to (1) lack of consideration in architecture decisions, (2) lack of time to learn about supports, (3) cost of regression testing when supports are added, and (4) incremental delivery of software that may have valuable functions, but is missing valuable qualities	• cost of delayed identification is often small
• delayed implementation has increasing delay cost across iterations	• delayed implementation has a fixed delay cost across iterations

When comparing the characteristics of quality and functional goals, we find many critical differences. To be successful, developers must understand these differences.

© 2015 ClearSpecs Enterprises

Consider the following:

- The number of quality goals relevant to a product ranges from 15 to over 40 and each quality goal has over 15 characteristics such as: indicators, measures, challenges, mitigations, supporting qualities, conflicting qualities, additional support functions, verification tactics, and priority. These characteristics are rarely documented.
- Some quality goals (e.g. availability, performance, privacy, reliability, safety, security, and usability) are **much harder to understand** than other quality goals. This is because each is a software engineering subfield with a large body of knowledge.
- **Understanding one quality goal does not help in understanding another**. Each goal (e.g. safety, reliability, and performance) has its own set of concepts that must be understood to achieve and verify the goal.
- Some pairs of qualities **conflict** e.g. understandability and performance, while others **support** e.g. reliability supports safety
- Some quality goals entail **levels of achievement** e.g. security, privacy, and performance. For example, if security is relevant, should it be **minimal** using a user name and password, **moderate** using user name, password, and security questions, or **strong** using a retinal scanner?
- Understanding quality goals entails **understanding their achievability** e.g. can 99.999% reliability be achieved in the application's environment? Since quality goals may conflict, two goals may be achievable individually, but not as a pair.
- Goals for quality attributes are **achieved using at least four types of supports**; (1) goals for supporting qualities (2) supporting functions e.g. logging and exception handling (3) sets of rules e.g. coding standards and design patterns and (4) warning labels e.g. "are you sure that …".
- Achieving quality goals can be **more difficult than achieving functional goals**. Even when the quality goal is precise and the candidate supports are known, you must select the specific supports needed to achieve the goal based on the system-specific execution environment.
- Verifying quality goals can be **much more difficult** than verifying functional goals. Quality goals, levels, and tactics must be verified with composite strategies that include technical review, analysis, measurement, and test.
- Effective verification of some quality goals (e.g., security, safety, robustness, reliability, recoverability, modifiability, reusability, and extensibility) **requires accurate anticipation of future events**.

While specific root causes of quality deficits may not be known, the following are likely candidates.

- There is a natural functional bias among developers created by responding to customer priorities. Quality goals are not what customers focus on, but failure to comply with (perhaps unstated) expectations will be quickly noticed.

- Detailed information about quality attributes is scattered and education is meager. There is only one comprehensive (over 50 goals), detailed (over 20 characteristics per goal), and succinct survey of quality attributes [1]. One result of this is that most developers have only a shallow understanding of quality goals [2].

- Most developers understand testing, but not verification of most quality goals (i.e. analysis, technical reviews, and measurement as well as four modes of testing). Testing alone is inadequate for quality verification.

© 2015 ClearSpecs Enterprises

- There is little leadership and guidance inside most software development organizations for many of the quality goals.
- Most quality goals are poorly supported by development methodologies [3]

Part 2: Project and enterprise tactics for reducing quality deficits

We now propose tactics for improving quality outcomes. The order of these tactics is meant to suggest their impact. The last three tactics help teams become quality aware.

A. Establishing a Quality & Productivity Support Group

B. Promoting quality awareness

C. Creating and using an enterprise knowledge base for quality attributes

D. Sketching and recording achievement and verification strategies for quality goals

E. Recording quality lessons

F. Providing guidance on fundamentals of quality attributes and goals

G. Running study groups on "high-risk" quality goals

H. Holding brown-bag lunch sessions to share quality experiences

We provide details of tactics A through E. Details about the others, as well as other resources, can be found at [4].

A. Establishing a Quality & Productivity (Q&P) Support Group

   The main objectives of a Q&P group are to develop quality competence in enterprise staff and quality appreciation in all stakeholders. The group should also focus on improving productivity while focusing on its quality objectives.

   The primary responsibilities of such a group are:

   1. Provide guidance in quality achievement and verification
   2. Acquire (and record) quality expertise

      If expertise is acquired from outside the enterprise, the strategy should be to transition the expert from trainer, to coach, to reviewer, to unnecessary.

   3. Identify and develop quality leaders

      The group should grow leaders, not be leaders.

   4. Promote quality-aware and verification-driven development (see tactics B and D)
   5. Acquire and evolve knowledge base(s) for quality attributes (see tactic C)
   6. Collect and adapt development standards

© 2015 ClearSpecs Enterprises

7. Acquire and record quality support and verification patterns

8. Participate in technical reviews

9. Track product behavior and team productivity

10. Acquire quality and productivity tools and training

11. Provide “safe ears” i.e. what happens in …, stays in …

Q&P groups are not test groups and must be involved with their project teams from the beginning.

They should not be confused with typical quality (QA or QC) groups. The primary responsibility of these other groups is system or acceptance testing. Often, they aren’t involved until late in a project. If your quality group is involved early and fulfills most of the responsibilities above, you are fortunate.

B. Promoting quality awareness

The challenge is to raise stakeholder awareness of quality goals to the same level as their awareness of functional goals. Quality awareness operates at the system and component levels.

System-level awareness implies early and continuing understanding of:

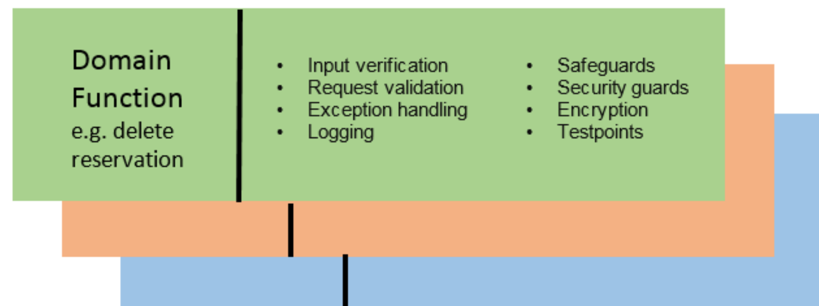
- high-priority quality goals and their characteristics
- conflicts between qualities and how they should be resolved
- critical supports for each quality level
- effects of the critical supports on each domain function
- how qualities will be verified

Quality-aware (Q-A) development refers to any development methodology starting and continuing with activities aimed at helping stakeholders increase their quality awareness. For example, Q-A Agile, described in [3], begins with the identification of project quality goals along with their levels, priorities, supports, and verification strategies.

Component-level awareness implies that developers understand that the code performing a domain function is just the beginning of acceptable software. Code such as input verification, exception handling, logging, safeguards, security guards, encryption, and software test points that supports quality goals is also needed (Figure 2). Quality supports may crosscut functional components.

© 2015 ClearSpecs Enterprises

Figure 2: Contents of software components



If domain functions are coded before most quality goals are identified, there will be significant technical debt associated with each function.

C. Creating and using an enterprise knowledge base (KB) for quality attributes

Start by acquiring a generic KB for quality attributes [1]. Tailor this generic KB to the specifics of your enterprise by:

1. Deleting irrelevant qualities and characteristics
2. Changing unfamiliar terminology
3. Change the remaining qualities to reflect enterprise-specific issues
4. Reorganizing the qualities to fit the enterprise quality vision
5. Adding stakeholder group priorities
6. Adding quality support and verification strategy patterns by level

The purpose of the enterprise KB is to help developers:

1. Identify relevant quality attributes and their characteristics including dependencies
2. Define feasible quality requirements with indicators or measures
3. Define an achievement strategy including attribute-specific support tactics
4. Define a verification strategy including attribute-specific verification tactics
5. Save any usable attribute-specific information from lessons learned or other sources
6. Access attribute-specific references

Begin each project by tailoring the enterprise KB into your project quality specifications. Tailoring should catalyze critical conversations with customers.

D. Sketching verification strategies for project quality goals

Verification assesses whether a solution complies with its requirements. A verification strategy is comprised of tactics including test, review, analysis, and measurement.

The following description of verification tactics and strategies is extensive because these strategies are unfamiliar to many readers.

Test tactics take many forms including predesigned testing, exploratory testing, built-in testing, and limited release testing.

© 2015 ClearSpecs Enterprises

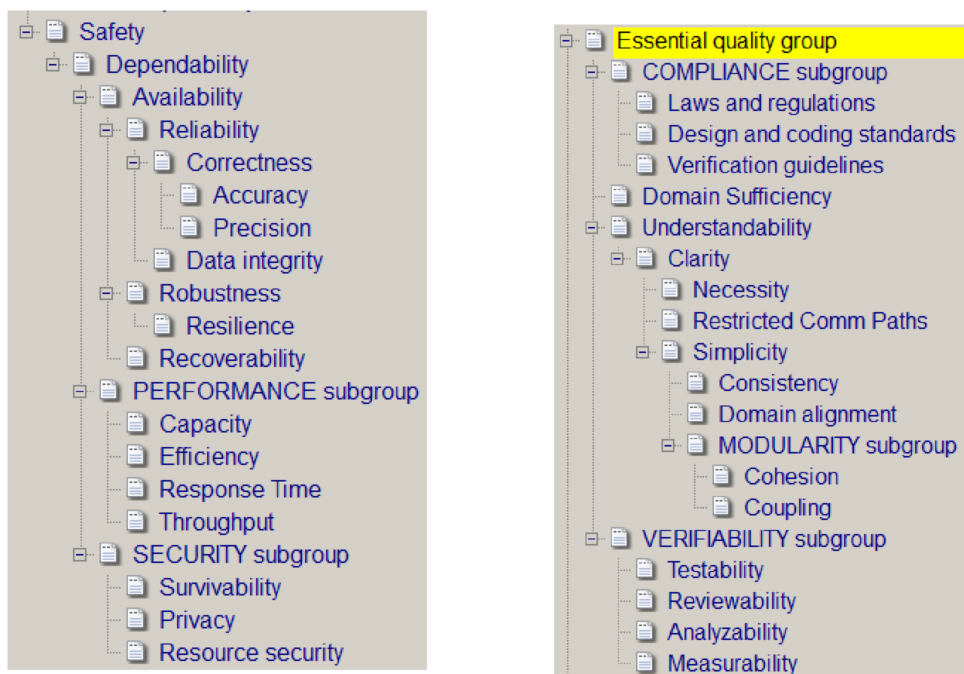
Review tactics [6] also have many forms including desk-checking, overview meetings, review meetings such as walk-throughs, and combinations such as software inspections. Reviews also include quality attribute audits such as safety, security, and scalability audits.

Verification-focused analysis tactics for quality goals include:

- subgoal identification analysis e.g. mitigations for hazards and threats
- quality goal exploration including vulnerability, usability, and performance
- program analysis including analysis of structure, control flow, and test coverage to assess code understandability and test adequacy
- data quality analysis
- formal methods, including model checking and symbolic execution, applied to quality functions such as caching and mitigations such as safeguards

Verification-focused measurement for quality goals entails assessment of the goal from a specific perspective in a specific context. For example, measuring the ease of handling of a specific aircraft flying at Mach 1. Some quality goals (e.g. understandability) may be measured indirectly by measuring indicators such as code complexity. Assessment of a characteristic (or its indicators) needs someone to select measures, scales, meters, and acceptance criteria [7, 8].

Figure 3: Fragments of a rich quality attributes model



Verification of a quality goal entails verification of its supporting goals and subgoals (Figure 3). For example, the left fragment of the quality attributes model shows that safety is explicitly supported by many other quality goals. The right fragment shows that safety is implicitly supported by “essential quality goals” i.e. those that support most quality goals. Safety verification requires the verification of all these supporting goals.

© 2015 ClearSpecs Enterprises

The sketch of a verification strategy for a quality goal includes:

- identity of the application, its usage environment, the quality goal, and the stakeholder perspective
- reference to supporting goals and subgoals
- specific test, review, analysis, and measurement tactics to be used for verification
- specific targets of each tactic
- project-specific details of a tactic, e.g. exactly how will usability be measured
- project-specific acceptance criteria for each tactic

A verification strategy is described by a set of sketches including those for supporting goals and subgoals. A verification strategy demonstrates the level of understanding of its associated goal.

The following example outlines a verification strategy for a safety goal. A sketch will have more detail.

Start Outline

Application: <application name>
Usage environment: <environment description>
Quality goal: safety
Stakeholder perspective: users

Supporting goals: dependability, availability, reliability, correctness, accuracy, precision, data integrity, robustness, resilience, recoverability, error resistance, performance subgroup, security subgroup, and all essential qualities

Supporting subgoals: mitigations for each hazard, threat, data corruption scenario, extreme usage scenario, and unstable environment scenario

Development tasks

Review scope: completeness and accuracy of Hazard Analysis [9] and Failure Modes and Effects Analysis [10] and their resulting safety hazards along with the effectiveness of proposed mitigations

Predesigned test scope: each subgoal safeguard and built-in test guards. Application misuse and unusual use tests [11].

Test analysis scope: modified condition/decision coverage (MC/DC) [12] for safeguards and access conditions

Predesigned test acceptance criteria: failure-free execution of all tests with complete MC/DC coverage as assessed by a coverage analyzer

Preproduction tasks

Limited-release testing with monitoring for safety-related incidents. Incidents include problems with any supporting quality goal.

Limited-release test acceptance criteria: At least 120 accident-free days, if each previous loss is less than $1,000 USD, otherwise at least 240 accident-free days.

© 2015 ClearSpecs Enterprises

Production tasks

Aggressive monitoring of reported accidents, with their rapid analysis, and rapid correction, if appropriate.

Performing extended root cause analysis on each accident and look for patterns.

Tracking: accident-free days, mean time between accidents, mean-loss per accident, and greatest loss.

End Outline

Verification strategies should be sketched and carried out early.

"IT projects that applied NFR (Non-Functional Requirement) verification techniques relatively early in development were more successful on average than IT projects that did not apply verification techniques (or applied them relatively late in development)" [13]

Sketching verification early is a natural start to prevention-oriented verification i.e. a broadening of prevention-oriented testing [14] from predesigned testing to all verification techniques.

E. Recording quality lessons

Enterprise KBs for quality attributes (and development standards) are natural places to record stakeholder priorities, achievement and verification strategies, and other quality lessons. Without recording, lessons must be relearned on each project.

Recording quality lessons is particularly important during initial experiences with enterprise KBs. Early project specs will be inaccurate as well as incomplete. Defects will also be introduced by the two tailoring processes. Early quality lessons across projects will incrementally correct many inaccuracies in your enterprise KB and its tailoring.

Conclusion

Prerequisite to any improvement effort is the recognition and acknowledgement that improvement is needed and the commitment to improve at the enterprise, project, and stakeholder levels.

Quality improvement requires serious collaboration. Help by spreading the word and getting others to spread the word about quality goals.

Let's make failure a rare exception, rather than the norm.

References

1. Gelperin, David (2015) **LiteRM Quality Knowledge Base** freely available at [4].

2. Private communication with staff at Software Engineering Institute

© 2015 ClearSpecs Enterprises

3. Gelperin, David (2015) "Agile is a Quality Anti-Pattern"

4. … www.quality-aware.com

5. … (2011) http://iso25000.com/index.php/en/iso-25000-standards/iso-25010?limit=3&limitstart=0

6. Wiegers, Karl (2002) *Peer Reviews in Software* Addison-Wesley

7. Gilb, Tom (2005) *Competitive Engineering* Elsevier [Chapter 5: Scales of Measure]

8. Simmons, Erik (2001) "Quantifying Quality Requirements Using Planguage"

9. Kamm, Daniel (2005) "An Introduction to Risk/Hazard Analysis for Medical Devices"

10. Mikulak, Raymond J. McDermott, Robin and Beauregard, Michael (2008) *The Basics of FMEA*, 2nd edition Productivity Press

11. Hope, Paco, Gary McGraw, and Annie Anton (2004) "Misuse and Abuse Cases" *IEEE Security & Privacy* May/June

12. Hayhurst, Kelly et. al. (2001) A Practical Tutorial on Modified Condition / Decision Coverage

13. Poort, Eltjo, Nick Martens, Inge van de Weerd and Hans van Vliet (2012) "How Architects See Non-Functional Requirements: Beware of Modifiability" *REFSQ* [Best Paper]

14. Gelperin, David and Hetzel, Bill (1988) "The Growth of Software Testing" CACM June

David Gelperin is Chief Technology Officer & President of ClearSpecs Enterprises. He has more than 40 years of experience in software with an emphasis on requirements risk management and software quality, verification, and testing (SQVT).

During the last 10 years, David has focused on requirements development. Before that, David was an SQVT consultant, coach and instructor, quality support manager, verification lead, project lead, and programmer.

In 1986, David co-founded Software Quality Engineering (www.sqe.com, www.stickyminds.com), the leading provider of software quality information worldwide. He also catalyzed the launch of Better Software magazine.

David has a PhD in Computer Science, but has managed to help people do useful things anyway.

ClearSpecs Enterprises
2425 Zealand Ave. N.
Golden Valley, MN 55427

© 2015 ClearSpecs Enterprises