Agile is a Quality Anti-Pattern (and how to fix it)

David Gelperin, CTO
ClearSpecs Enterprises

I. Quality attributes
II. Agile & its problems
III. A solution

My Goal and Strategy

Goal

Help you adjust the way you develop (or acquire) software – whether Agile or not.

Strategy

Explain why adjustment is needed and how to do it

Quality attributes

There are over 50 software quality attributes (ilities) including survivability, safety, security, and robustness.

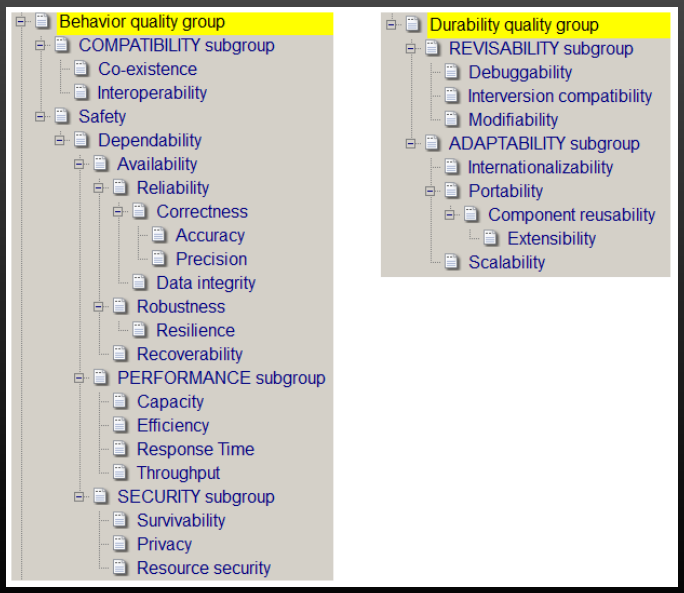
Each quality attribute has over 20 characteristics including priority, conflicting qualities, supporting qualities, and achievement and verification tactics.

This is true across all applications and all domains.

You must identify the attributes relevant to your software and understand their characteristics, including conflicts.

Attribute examples -- 1 of 2

Attribute examples -- 2 of 2



 5

Characteristic examples -- 1 of 2

Reliability

Definition Degree to which system effectively performs requested functions under all conditions

Software subfield reliability engineering

Assumptions/Rationale

Indicators -- leading & trailing measures of supporting and directly supported attributes

Measures -- leading & trailing
Mean time to failure (trailing)
Mean time between failures (trailing)
Failure rate (trailing)

Aspect of availability, modifiability

Supporting qualities correctness, data integrity

Associated with robustness, recoverability

Conflicting qualities performance, many adaptation qualities

Threats Defective code or data, hardware failure, excessive customization

Mitigations Formal review, data analysis, thorough testing, reliability measures tracking

 6

Characteristic examples -- 2 of 2

Additional achievement tactics

- Monitor and control system states and data integrity (e.g. output)
  - Specialized interfaces
- Comprehensive exception handling
- Maximize reuse of reliable components

Constraints Design and coding standards that limit complexity, verification guidelines requiring technical reviews, measurement, analysis, and comprehensive usage, code, and data test coverage as well as exploratory testing

Verification tactics review standards compliance, review and test code including exception handlers, analyze data, measure and track mean time to failure, verify all supporting qualities

Elicitation Questions

- Which system aspects threaten reliability
- What is acceptable reliability
- Which functions require ultra-high reliability
- How will unreliability of inhouse and third-party software be detected and communicated
- How to handle unreliable external systems

Resources

The Quest for Software Requirements -- section 5.5

Software Reliability Engineering

Automated Source Code Reliability Measure OMG/CISQ

Use of Relative Code Churn Measures to Predict System Defect Density

 7

Quality goals are poorly understood

Required quality attributes (quality goals) are difficult to achieve and verify.

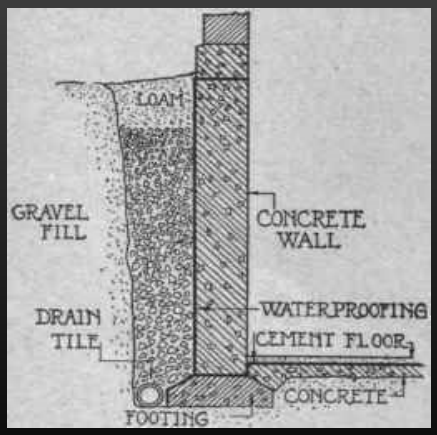
Many developers (and their managers) have little understanding of how to do either.

Most developers understand testing, but not verification. Unfortunately, testing alone is inadequate for the verification of many quality goals.

Quality verification entails analysis, technical review, and measurement, as well as four modes of testing.

 8

Developers (& their managers) neglect foundations



When a wall cracks (insufficient footings) in your home or your basement floods (insufficient drainage or waterproofing), you understand the need for a solid foundation.

When your system is hacked or crashes under high volume, you understand the need for a solid foundation i.e. set of quality supports.

Functions are familiar and fun. Qualities are neither.

Agile

Agile is a set of values and principles for software development. It is not a development methodology.

Agile methodologies embody Agile values and principles.

All Agile methodologies entail iterative, incremental development of functionality, driven by customers and evolving understanding i.e. change.

Which Agile methodology?

There are 4 types of Agile methodologies

1. Named e.g. Scrum and XP
2. Pure-hybrid i.e. blend of 2 or more named methodologies
3. Mixed-hybrid i.e. blend of 1 or more named methodologies and non-Agile practices (e.g. defining quality goals first)
4. Relabeled i.e. calling whatever you are doing "Agile"

Each type has two forms:

1. Written about i.e. specified Agile
2. Actually done i.e. actual Agile

Agile is a Quality Anti-Pattern means

named or pure-hybrid, specified Agile is a Quality Anti-Pattern

Agile's 5 problematic principles -- 1 of 2

2. Welcome changing requirements, even late in development

Welcoming (rather than avoiding) late changes to crosscutting quality requirements e.g. security, is a bad idea. It is akin to harnessing a porcupine – the pain will linger. These expensive changes usually result from voluntary ignorance rather than the emergence of unimaginable quality goals. This principle can be fixed by adding "functional" to "requirements".

6. The most efficient and effective method of conveying information to and within a development team is face-to-face conversation

Face-to-face conversation is great. It is not the most efficient nor effective method of sharing information about quality goal definitions nor about quality achievement and verification strategies. This principle can be fixed by adding "many kinds of" to "information".

Agile's 5 problematic principles -- 2 of 2

11. The best architectures, requirements, and designs emerge from self-organizing teams

Quality requirements don't need to emerge because they can be selected early from a quality knowledge base (www.quality-aware.com). Emergence is great, when experience and understanding are lacking. It is inefficient and expensive, when the choices are known. This principle can be fixed by adding "functional" to "requirements".

3. Deliver working software frequently, from a couple of weeks to a couple of months, with preference to the shorter timescale

7. Working software is the primary measure of progress

What is "working software"? Without identifying quality goals, developing achievement and verification strategies, and implementing crosscutting quality supports FIRST, the early increments can't be defect-free nor satisfy their quality requirements. These principles can be fixed by defining "working software" as "a possibly-fragile prototype sufficient to demonstrate the functionality to be delivered"

 13

Agile quality – what does it mean?

Scrum – the most popular methodology – provides no guidance on quality goals

Extreme Programming (XP) practices include:

- pair programming and thorough code review and unit testing of all code
- test-first development i.e. planning and writing tests before each increment
- automated testing
- coding standards – 2/3 not doing in 2010
- simple design
- refactoring

XP practices:

- focus on clean, understandable, and effective code thus supporting acceptable functionality, reliability, and understandability i.e. focus on 3 out of 50 attributes
- provide necessary, but insufficient support for most other attributes

 14

Agile's bottom-up functional bias

- Discourages specifications because code and tests are considered satisfactory
  - No specification of quality threats (e.g. to safety or security) and strategies for achievement and verification
- Discourages up-front analysis and design for fear of waste, including gold-plating
- Focus on testing, rather than verification
- Emphasis on incremental design, which is very inefficient for crosscutting quality supports
- No mention of
  - risk management
  - resolving quality conflicts
  - designing crosscutting quality supports

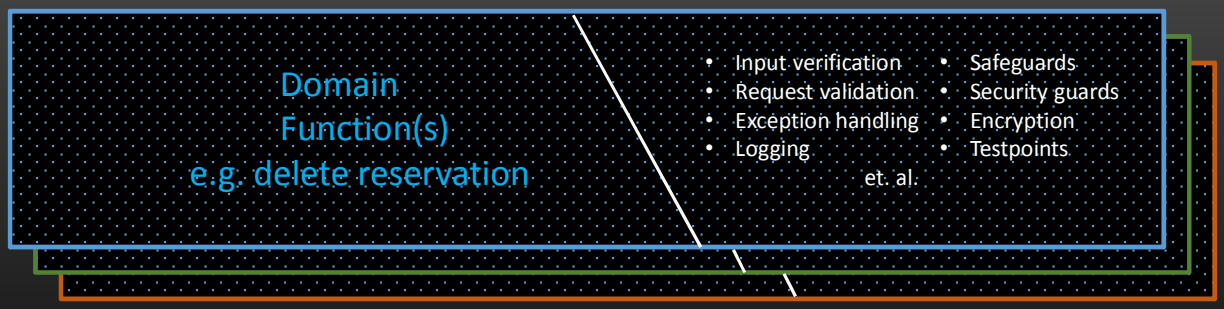
Agile is terrible at achieving most quality goals

XP, done well, is wonderful at achieving functional goals and supporting three quality attributes.

All Agile methodologies are terrible at achieving and verifying the other 50 quality goals, because:

1. Agile emphasizes functions and de-emphasizes most quality goals to the point of invisibility.
2. Agile emphasizes testing and, except for code and tests, de-emphasizes technical reviews, analysis, and measurement to the point of invisibility.
3. Nothing assures that all high-priority quality goals will emerge before product delivery.

Agile* causes reckless short-term technical debt

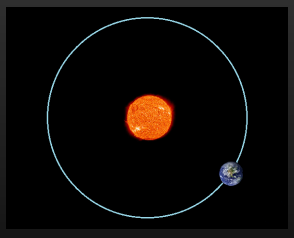


"Complete" functional components must contain quality support code

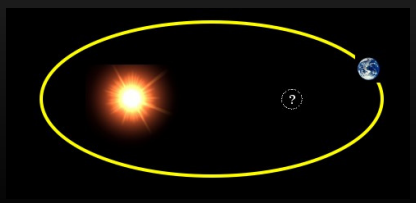
Late identification of quality goals causes reckless short-term technical debt

Agile methodologies over-simplify

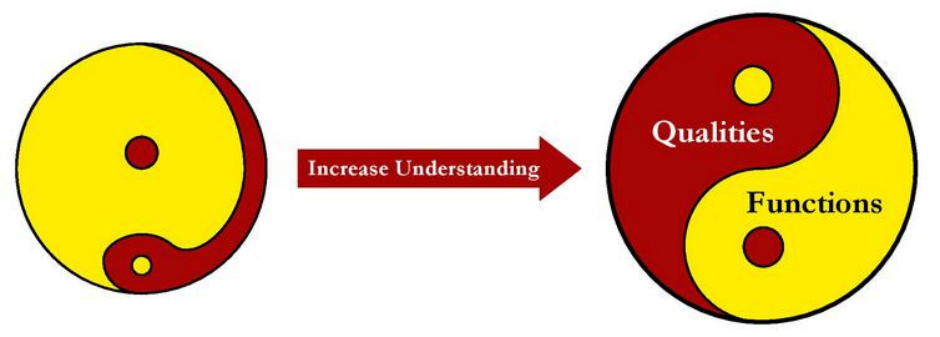
For every complex problem there is an answer that is clear, simple, and wrong.
H. L. Mencken



Ideas should be made as simple as possible
-- but no simpler.
paraphrasing A. Einstein



A Solution -- Quality before functionality



 19

Identify quality goals first

Most quality goals can be accurately identified from knowledge of the software's operating environments and basic mission e.g. flight control, internet gaming, or stock trading, and use of a quality knowledge base.

Early identifications may need to be adjusted as understanding deepens, but there is no way to predict when you have enough information to accurately determine a quality goal without waiting until all functional code has been written.

Waiting is an expensive alternative to early identification.

 20

Quality-Aware Agile

Quality-Aware development is NOT a development methodology, but a 3-part supplement to whatever you are doing now or intend to do

1. It begins with a quality goals sprint in which you identify quality attribute requirements, their levels, priorities, challenges, mitigations, and achievement and verification tactics. Using a quality knowledge base, it should take less than a week to draft a preliminary quality specification.
2. In each iteration, you reassess and carry out the quality strategies.
3. Finally, you collect the quality lessons during project retrospectives and record them in the enterprise base of quality knowledge and/or in the development standards.

Wrap up

Big Requirements Up-Front (BRUF) is an Agile anti-pattern, because BRUF is inconsistent with evolving understanding (of desired functionality) caused by incremental development

We recommend Quality Goals First (QGF) i.e. a mixed-hybrid strategy, because understanding of quality goals should evolve little. Failure to practice QGF always results in significant short-term technical debt.

Checkout www.quality-aware.com

