Software Development Should Have A Dual-Focus

David Gelperin
ClearSpecs Enterprises

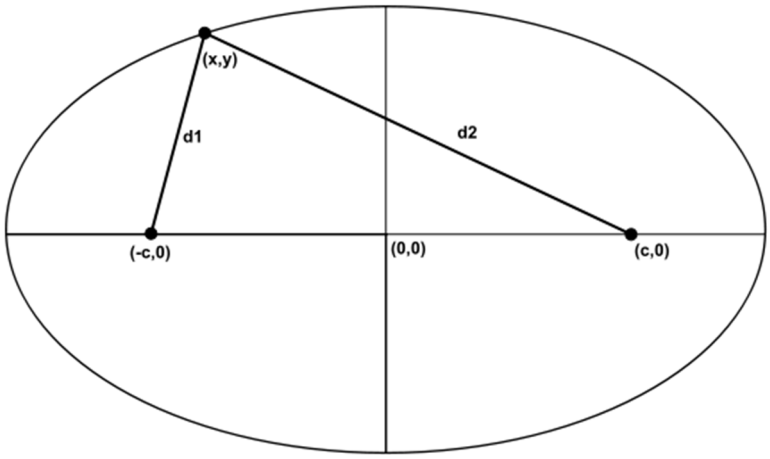
> For every complex problem there is an answer that is clear, simple, and wrong.
>
> H. L. Mencken

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

Simple as possible

Once upon a time, the earth was flat and "flat" worked – as long as you took short trips. Then a spherical earth circled the sun and "circled" worked – almost. Kludgy calculations using circular orbits produced fairly accurate celestial predictions. Then elliptical orbits removed most kludges.

In geometry, an ellipse is a plane curve for which the sums of the distances of each point on the curve from two fixed points, the foci, are equal. If the two foci are a single point, the curve is a circle.



-- but no simpler

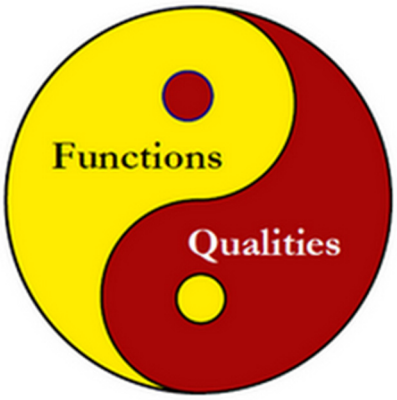
Current approaches to software development are too simple, because they focus almost exclusively on domain functionality. The achievement and verification of requirements for quality attributes i.e. quality goals, such as those for security, availability, and ease of use, are kludged. Software quality failures resulting from a functional bias often appear in the news.

So what's wrong with a few kludges, except that they may not work? Since there are more than 20 quality attributes of interest in many applications, there are more than a few kludges. In addition, kludges are hard to (1) verify i.e. analyze, review, and test (2) understand and explain (3) debug, and (4) learn much from, when they fail.

Some view failure as the unavoidable result of developing large, complex systems. While some failures may be unavoidable, their current frequency suggests a different problem.

Few developers know how to produce dependable systems. The problem is that dependability is not a function and it relies on many other quality attributes (i.e. non-functions). Since the achievement and verification of quality goals is poorly understood, failure is becoming the norm [1].

What is needed is a dual-focus on domain functionality and quality goals. One example of this dual-focus is Quality-Aware Development [2] a framework for focusing on quality goals, their achievement, and their verification as well as domain functionality. This approach seeks a balanced emphasis on functions and qualities.

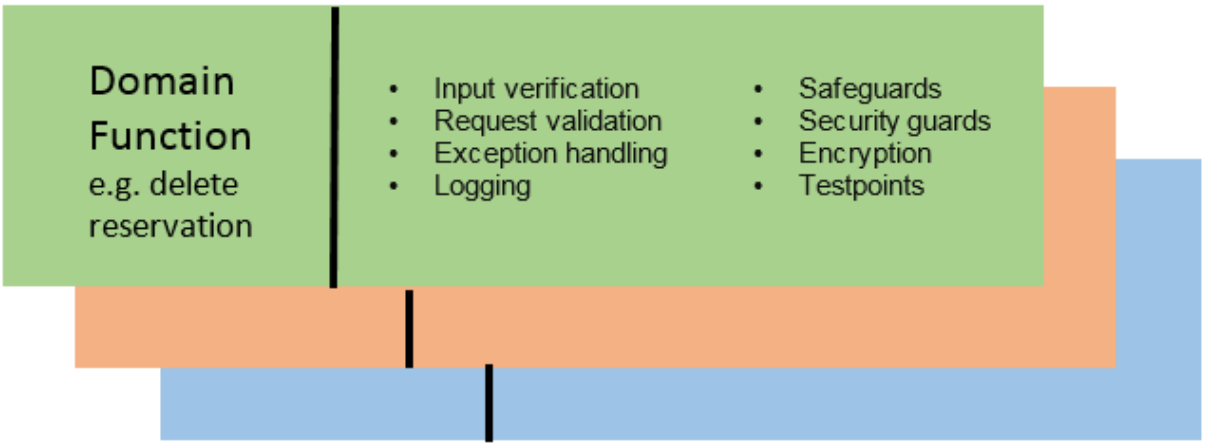


Key quality concepts

- **Quality attribute** – a system attribute such as reliability, security, safety, and ease of use.
- **Quality goal** – a requirement for a quality attribute.
- **Quality level** – some quality goals are binary, such as safety i.e. safe or unsafe, while others are scalar, such as security and reliability e.g. 99% reliable vs. 99.99% reliable.
- **Quality support** – a function e.g. safeguard, rule, warning label, or quality attribute that helps to achieve quality levels for specific attributes.
- **Crosscutting quality supports** – supports, such as exception handlers, that reappear throughout an application so as to achieve a quality goal.
- **Quality achievement strategy** – a collection of quality supports that achieve a specific quality goal.
- **Quality verification tactic** – various forms of analysis, technical review, test, or measurement that help check the achievement of quality goals.
- **Quality verification strategy** – a collection of verification tactics used to check the achievement of specific quality goals.
- **Quality attribute characteristic** -- a characteristic of a quality attribute such as definition, priority, measures, conflicting qualities, challenges, mitigations, supports, achievement strategies, and verification strategies.
- **Rich quality attributes model** – an extensive collection of information about (50+) quality attributes including their (20+) characteristics and relationships as well as their achievement and verification strategies. The model is updated based on project and fielded application experience.

- **Rich quality goals model** – a model of the specific quality goals for an application derived by tailoring a rich quality attribute model.
- **Quality-Aware development** – refers to both the early project tasks of identifying quality goals, levels, achievement strategies, and verification strategies as well as the iterative tasks of supplementing or changing any of these as needed and carrying them out.

To understand the differences between single and dual focus, we compare them at the unit level. Many units contain both code for domain functions and code for quality attribute supports. A satisfactory unit must get both right.



The following process outlines provide detailed examples of single and dual focus development.

Single-Focus Unit Development

We use Test-Driven Design as an example of single-focus development. Test-Driven Design is an iterative process for developing domain functionality. It consists of:

1. unit test specification and execution (expecting failure)
2. incremental design and coding
3. unit test execution and debugging as needed and
4. refactoring

Dual-Focus Unit Development

We use Quality-Aware Development with pair programming as an example of dual-focus development. Quality-Aware unit development is performed by a Verifier and Coder partnership. Each partner is mostly responsible for one of the foci.

For **domain functionality**, iterate

1. [V] design clear unit test(s) (i.e. executable specs for domain functionality)
2. [C] review unit test(s) for boundary coverage, etc. and [V] change as needed
3. [C] design and code domain functionality
4. [V] review domain code and [C] change as needed

5. [V] perform unit test(s) and change [V] tests and [C] code as needed
6. [V, C] analyze test report for branch (or stronger) coverage and change tests and code as needed

For **quality goals**,

The initial quality sprint (for the team)

1. Select relevant quality attributes including their supporting attributes from your quality model
2. For each selected attribute:
   - identify its required level
   - identify its challenges (e.g. using threat analysis), mitigations, and supports
   - assess its feasibility
   - specify and review its achievement and verification strategies
3. Acquire and verify a library of crosscutting support components e.g. exception handlers. Acquisition entails both reuse and development.

Iterative tasks (for coder and verifier)

For each iteration-relevant quality goal:

1. [C,V] reassess its achievement and verification strategies and update as needed
2. [C] carry out its achievement strategy, clearly identifying quality support code
3. [V] verify its achievement and [C] change as needed

Conclusion

Both approaches to domain functionality are similar. Only dual-focus development explicitly addresses quality goals and provides a consistent approach to achieving them. Replacing your collection of kludges with dual-focus development (that identifies qualities before functions) will improve the quality of your software.

References

[1] David Gelperin "Failure is Becoming the Norm" available at Quality-Aware.com

[2] Quality-Aware.com