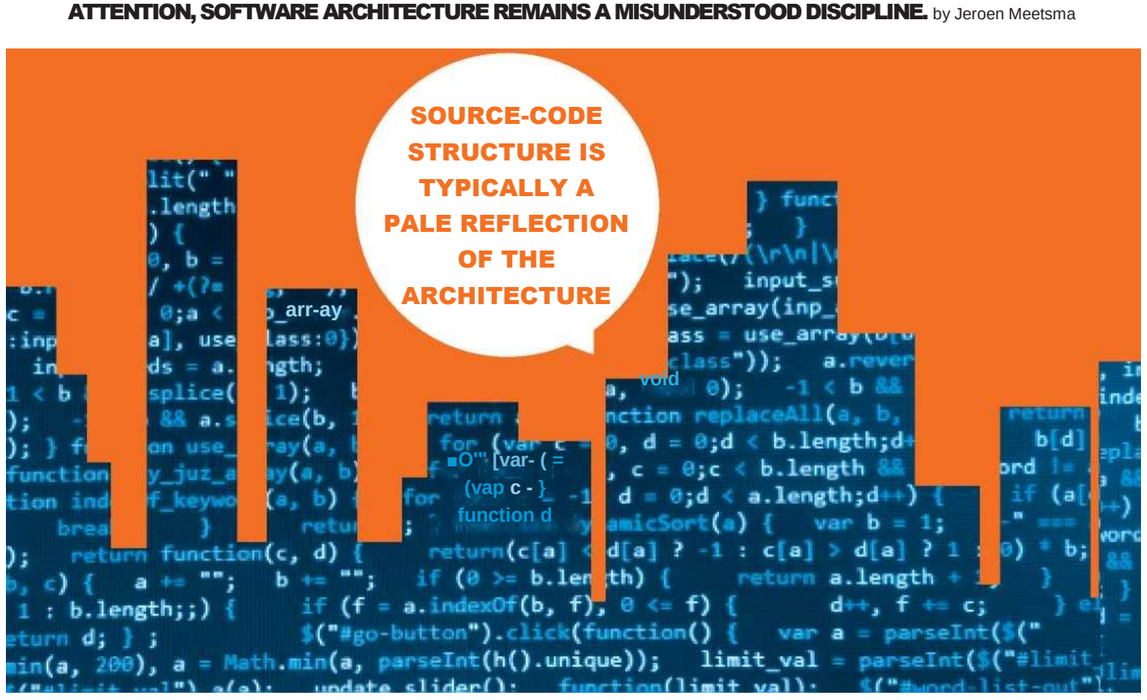
# SOFTWARE ARCHITECTURE OFT-NEGLECTED

**SOFTWARE ARCHITECTURE HAS A GREATER IMPACT ON THE QUALITY OF A PRODUCT THAN THE MEASURABLE QUALITY OF THE SOURCE CODE SOURCE-CODE QUALITY MEASUREMENTS OFFER A FALSE SENSE OF SECURITY. IT LEADS DEVELOPERS TO FIGHT THE SYMPTOMS. DESPITE ALL THE ATTENTION, SOFTWARE ARCHITECTURE REMAINS A MISUNDERSTOOD DISCIPLINE.** by Jeroen Meetsma




IN DEVELOPING AGILE SOFTWARE, THE ATTENTION IS STRONGLY focused on functionality that is visible to the (end) user. Attention to the longevity of software is limited even in the best of cases to improving source code based on the findings of statistical code analysis. Conversely, maintenance and usability for the long term are strongly determined by the quality of the software's architecture. The degree to which the architecture has been implemented in the code will have a greater impact on product quality than the quantifiable quality of the source code.

Software architecture remains a frequently misunderstood and oft-neglected discipline despite the plethora of conferences and forums dedicated to it. It could be defined as a series of design decisions resulting in the description of high-level components and how they correspond to one another within an application. Software architecture describes a given application from multiple perspectives, such as how the source code is laid out, runtime components, links to other systems and the rollout onto hardware. Architecture should keep oversight and promote the targeted implementation of functional and non-functional requirements. The correct components arranged in a well-chosen structure offer the possibility to modify an application to reflect changing circumstances without too much effort.

**CODE FRAMEWORK**

**In 2010, George Fairbanks introduced the idea of the 'model-code gap' in his book, *"Just enough software architecture"*. The term depicts that architectural models and code express the structure so differently that there will be a gap. The grammar of common programming languages offers few possibilities to describe the organization of architectural components. It requires creative applications of design patterns or the use of frameworks. One effective method for applying structure is to have a small team convert the architecture into a code framework. Such a framework conveys structure through key components with identifiable names. It also contains end-to-end coded functionality in the form of a reference implementation. The next step is to allow a larger development team to implement the supplementary features.**

The configuration and orchestration, which connect these components to one another, can preferably be read as an instruction manual on how the system is put together functionally and technically. Developers who wish to modify a component must be able to concentrate on the corresponding piece of source code. They must be able to make adjustments without having to grasp how the entire system works or causing damage.

The introduction of the concept of architecture in IT was a useful step, because architecture distinguishes between design components arranged at various levels. Yet the creation of two explicit roles of architect and developer - the latter frequently referred to by the government as a 'builder' - has also resulted in a gap between design and implementation. Software architecture bridges this gap.

## STEERING TOOL

Clients and management want to steer the development of custom software. Architecture is an ideal means to provide this guidance, as long as it appropriately translates client value and corporate objectives into a feasible technical solution.

It goes without saying, however, that in order for architecture to actually serve as a steering tool the source code must be structured in accordance with the architecture. And that, during the development process, the architecture must evolve with the structure of the code.

# SOFTWARE ARCHITECTURE: INVISIBLE & OFT-NEGLECTED

The quality of any software is partly hidden. Stakeholders either perceive an application's effectiveness positively in the form of features or negatively in the form of flaws. Agile has a strong focus on the visible aspects of software: adding new features and fixing any bugs. The use of tools has increased the visibility of overdue maintenance or technical debt as well. Software architecture is an oft-neglected aspect, relatively speaking.

| | VISIBLE | INVISIBLE |
|---|---|---|
| POSITIVE | CHARAC-TERISTIC | ARCHITEC-TURE |
| NEGATIVE | FLAW | TECHNICAL DEBT |

This infers a continuous interaction between code development, architectural function and stakeholders. In principle, every fragment of code in a custom solution can be traced back to efforts to add value and meet the company goals. In practice, however, the structure of source code is usually a pale reflection of the supposed architecture. Architecture is nothing more than a shared conceptual image that offers a lot of room for individual interpretation. The consequence is that applications can become monoliths rather than a collection of detachable components. The discrepancy between design and implemented structure is known as the 'model-code gap'.

Tools for measuring the quality of code, such as SonarQube, Fortify and Findbugs, are increasingly becoming a fixture in the development process. They guide developers through functions that are too long or complex and constructions that might lead to poor performance or that have a low degree of testing coverage. While these tools certainly provide an indication of (potential) poor quality, they can identify issues only at a microlevel. This delivers a one-sided picture. Each fragment of source code is measured against the same yardstick with no distinction being made on the function or relevance.

**AUTHOR**

JEROEN MEETSMA (j.meetsma@bon-code.nl) is co-founder of Boncode Software Remediation and specializes in software quality and software architecture. He has successfully developed a tool for measuring software quality and is currently working on an analysis that makes architecture visible in an implementation.

Also the extent in which the code is worked or how often it is called upon in production does not play a role. Software scans will generally give a positive score to code that is sufficiently modular; code that is compact and legible. This is not to say that code with a high score is easy to maintain; 'doing things right' is not necessarily 'doing the right things'. Developers who work diligently to solve issues, will not necessarily end up with code that is robust or understandable to others. In actuality, improving the code based on static code analysis is in fact only treating the symptoms.

## IMPACT OF AGILE

An iterative, agile approach to software development will address the application of architecture in the code -- at least in theory. This principle is known as emergent design. Simply make a few initial design decisions and then the architecture will develop gradually. It's assumed that developers have enough skill in refactoring or restructuring to allow them to adjust the structure so that it reflects the way the software works as new information emerges or requirements change.

In practice this is not always the case. Agile methods constantly keep developers attentive when it comes to developing features. Standup meetings, planning sessions and especially sprint demos

**REACTIONS AND CONTRIBUTIONS**

To submit reactions and new contributions for IT experts, contact:
Henk Ester
+31 (0)20 235 6415
h.ester@agconnect.nl

> “Structural improvements aren’t visible and end up being put off.”

ensure that a developer is constantly being asked to demonstrate their progress.That’s what developers do. They concentrate on all the visible aspects: greater functionality, fewer SonarQube issues and high unit testing coverage. Structural improvements aren’t visible and end up being delayed. Restructuring impedes the visible progress of other developers. Not only production code needs to be adjusted, but in many cases, the rewriting of a significant portion of unit testing code as well. If there is focus on the architecture, then there is another inhibiting factor. Agile project teams are typically non-hierarchical, which means ‘the team’ decides which issues are picked up and how. It means the team must reach unanimous decisions on matters of architecture. In practice, this leads to a good deal of discussion and compromise. This form of architecture is referred to as ‘design by committee’.

## GAP

Software architecture and its implementation in source code deserve more attention. A development team must keep implementation synchronized with a design in order to close the model-code gap. This is to visualize the implemented structure and to reduce design decisions client value, so that architecture can serve as an additional steering tool for management. An immature Agile team will be incapable of shouldering this responsibility. This assumes a new approach to the architect’s role. This requires an architect who not only designs top-down, but is also able to interpret source code and complete the feedback cycle. An architect who intervenes when structural improvements to the source code cannot be postponed any longer. Occasionally, the development of features must be put aside in favor of restructuring. This can be scheduled in separate technical sprints, in which the codebase is temporarily frozen. The advent of Agile has improved the software development process because it has made adjustments possible. In addition, software architecture has shifted to the background. Yet Agile has spared us from the kind of rigid technical design that eventually inhibits development. In fact, it offers an opportunity to bridge the model-code gap, the final obstacle to productivity. Like functionality, architecture must be a part of the iterative development cycle. By consciously increasing the visibility of architecture and integrating structure into the development process, it takes on the role of steering tool alongside functionality and general code quality. O

## 5 MISINTERPRETATIONS OF STATIC-CODE ANALYSIS

Static code analysis renders programmers’ skill and craftsmanship tangible. However measured characteristics give an overly simplistic view of how maintainable the source code will be. Without proper interpretation, the numbers will take on a life of their own.

Five misinterpretations:

## 5 PITFALLS OF AGILE SOFTWARE DEVELOPMENT

1. A lot of code has been added, so that means developers have been productive.
2. We have a high rating, so the software must be good.
3. We scored 90 percent on the unit testing coverage; the code is solid.
4. If we remove all measured issues, the code will be maintainable.
5. Product quality is meeting metric values.

Agile software development originated with a manifesto that lays out a number of best practices used by ‘excellent’ teams in order to ensure productivity. Today, Agile has become synonymous with the methods and processes named for it.

Five pitfalls of placing your trust in Agile as a methodology:

1. Visible client value is all that matters.
2. The process will always yield a result; sending on time or budget is unnecessary.
3. Any team can make the best possible decision by working together.
4. A mediocre team can excel
5. by strictly following Scrum.
6. The architecture will work itself out during the process.