silverthread

Is your code structured as you intended? Probably not.

Hidden and unintentional complexity causes waste and overhead. Identify architectural impurities early and consistently.

Software architectures diverge from intentions over time.
A software architecture must be measured regularly and objectively to stay healthy. Architectural evolution often occurs organically, with many constituents changing individual elements separately. Schedule pressures, individual motivations, competing performance measures, and the immense complexity of most systems result in long chains of sequential and parallel incremental change of the parts without a commensurate understanding of the whole. Eventually, there is a gap in understanding between the intended architectural structure and the system's actual architecture evolving in the coded elements. This architectural entropy leads to unpredictability and confusion. Engineers flounder trying to understand the ever-increasingly complex systems that their enterprises maintain and depend on. Managers feel out of control when seemingly straightforward changes cause malignant breakage and protracted timelines.

Software talks; supplemental artifacts walk.
Most software teams rely on supplemental artifacts disconnected from the code base to communicate the design structure. More honest measurements and up-to-date insight come directly from the code and test base itself.

Supplemental artifacts such as hand-written documentation, separate design models, PowerPoint charts, or build files can indirectly describe design quality, but engineers may subvert these indirect controls. Sometimes they do so cleverly. Most of the time, however, they do so unknowingly. A more reliable method periodically scans the evolving code-base directly with quantified network analytics. Such analysis provides a measured representation of the actual architecture for review and control.

Mind the understanding gap
When a team's understanding of the code base aligns well with the actual software, less overhead is required, less waste and rework is experienced, confidence and trust increase, and efficiency improves dramatically. Less overhead and waste translates into higher team morale and improved economic outcomes.

An overemphasis on measuring process artifacts creates distracting noise and unnecessary overhead. Measuring things that are useful to management but not to practitioners (or vice versa) erodes trust.

- *The old way.* Measures of the process and other supplemental artifacts are indirect indicators and more subjective guesses. They are noisier, and easier to game.
- *The new way.* Measure the dynamic characteristics of the product pipeline, not the process pipeline. Direct measures of the code/test base are objective facts and mostly signal.

The gap between design intentions and coded releases can be measured and understood so that technical debt can be minimized.

The importance of frequent, macro-level measurement
Well-architected systems help ensure the confident, independent evolution of software components over time, providing a durable framework where engineers can add value to an evolving software system. *Design quality* quantifies architectural attributes that enable efficient and effective incremental change. Silverthread research has demonstrated that when these attributes degrade, business outcomes degrade commensurately. When this degradation is identified early, efficient resolution is practical. When it goes unnoticed and uncovered until later, the unintended consequences can be unrecoverable.

Figure 1 is a file-level visualization from a Silverthread CodeMRI® report. SubsystemX is one subsystem of a code base developed and maintained by a large air travel services organization. This subsystem consists of 70 components, each with direct and indirect dependence on other components. The software team's lack of agility, as well as their inability to reuse valuable utilities located within this component, was a serious concern. The organization was preparing to move its software into the cloud and required an objective measure of the readiness for this component to work effectively in its new microservice architecture. Figure 1 captures the actual structure of these files and components, including thousands of unexpected illegal dependencies appearing above the diagonal.

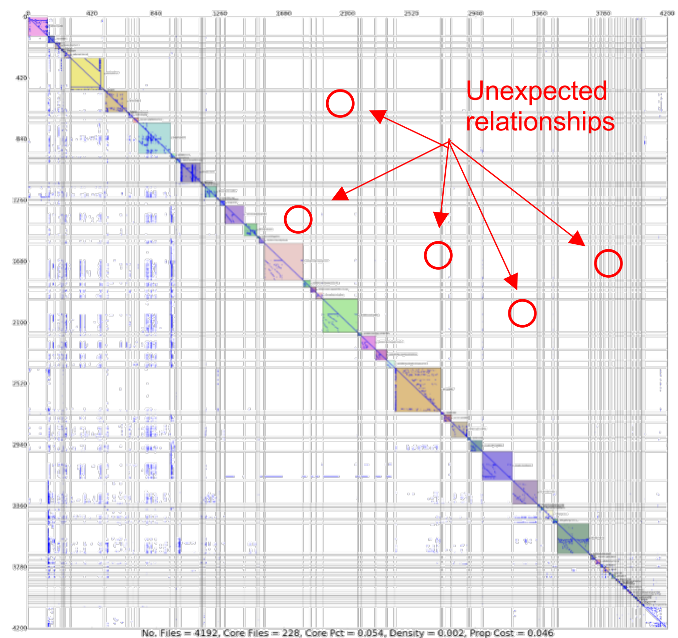

Figure 1: Insights revealed by network analysis

Figure 2 shows the same subsystem relationships abstracted up to a higher component level. It also shows the intended structure understood by engineers and captured in supplemental artifacts.

©2016 - 2020 Silverthread Inc. | 200 Portland Street, Suite 500, Boston, MA 02114 | www.silverthreadinc.com | info@silverthreadinc.com | Ph: (617) 603-0075

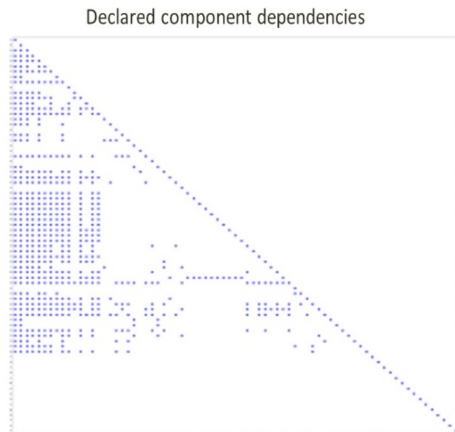

Figure 2: SubsystemX intended structure

Figure 3 shows the actual structure extracted from the code base, including all hidden complexity caused by illegal relationships. The degree of unexpected cyclicality was alarming, both to practitioners and to leadership.

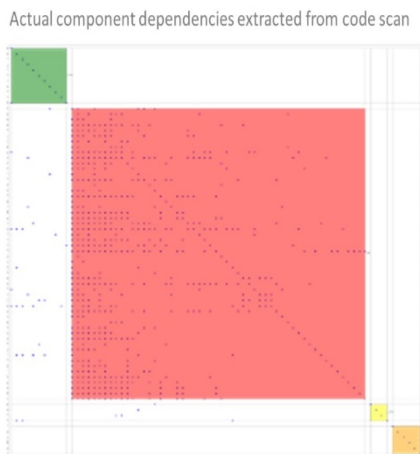

Figure 3: SubsystemX intended vs. actual structure

With this new insight, the team could act on a primary root cause of their inefficiency and lack of reuse. The amount of unintended coupling also exposed what this subsystem's challenges would be if it were to be refactored for microservices.

Assessing and addressing hidden complexity
Silverthread's know-how and tooling can help visualize and quantify hidden and unintended complexity. Clients can expose the relationships responsible for the design understanding gap by their degree of architectural impact:

- Green relationships are safe, existing in both the intended design and actual code base.
- Yellow relationships deserve review. These are suspicious relationships that do not introduce cyclicality.
- Red relationships deserve critical review. These are suspicious relationships that cause unintentional cyclicality.

Figure 4 shows a diagnostic view of this subsystem's component relationships highlighted as described. Key relationships are now obvious, providing diagnostic insight to architects about where unexpected complexity has appeared in their system. Such visualizations promote more meaningful discussion between software designers and programmers, and can become the basis for periodic review and control. Teams can now balance the (previously misunderstood and nebulous) macro-health of the forest with the (well-understood and quantified) micro-health of individual health of the trees.

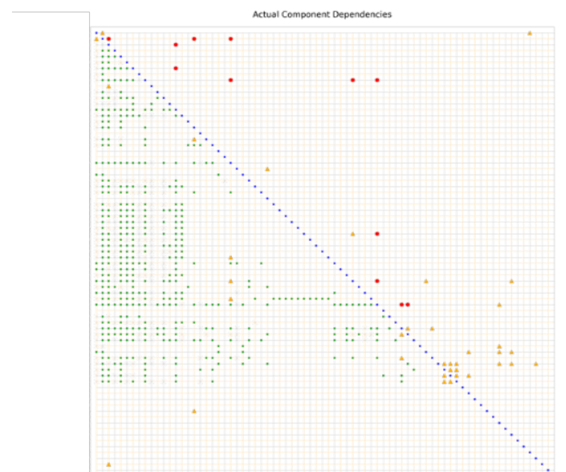

Figure 4: Architectural component dependencies

Ensuring architectural health through objective measurement
When confronted with these newly visible insights, the development team was understandably skeptical, having taken great pains to control their structure effectively via supplemental documentation and build orders. Follow-up analysis by the developers verified the findings and resulted in process improvements and several change initiatives to improve the efficiency of resolving technical debt.

Unexpected architectural divergence can be avoided through periodic diagnostic assessments of the evolving software structure. Unintended cyclicality should be systematically addressed as early as practical to prevent an extensive understanding gap. Silverthread analytics are well positioned to provide these insights through our actionable diagnostics. Armed with such insight, software leaders, architects, and developers can have more meaningful, honest conversations about software economics.

Contact Us

Silverthread's mission is to advance the state of software measurement practice by quantifying complexity and design quality. Our measurement know-how can establish a more trustworthy foundation for improving software economics.
http://silverthreadinc.com