Health (forest) $\neq$ $\sum$ health (individual trees)

Practitioners and leadership must objectively understand both code quality and design quality.

The JDOE project is a long-lived, complex software system
Like many legacy software systems, JDOE has a checkered history. JDOE (not its real name) is a ground-based command and control system. It was originally written in the JOVIAL programming language over four decades ago and maintained for a decade or two before being modernized and rewritten in Ada. This "rewrite" involved no refactoring and was performed simply by translating JOVIAL entities and relationships into Ada entities and relationships. The Ada version has since been maintained for a decade or two and supplemented with additional in Java. The current version is a few million lines of code. It probably doesn't surprise anyone that decades of maintenance and a line-by-line translation has resulted in a very complicated system to understand. The amount of architectural entropy that creeps in to a large-scale codebase over decades of change is significant. And recoding a Jovial-based architecture in Ada undoubtedly added substantially more entropy since the semantic differences between the languages were difficult to systematically comprehend. Finally, any system that survives 40+ years has undergone substantial personnel changes and organizational handoffs such that the original architectural intent has been lost.

Why is this case study interesting? It is not as extreme as one might think. Many organizations have legacy systems that have undergone similar evolution. Perhaps some of the parameters are extreme but the pattern is common. We end up with a convoluted architecture that has been degraded by short-sighted decisions, natural entropy, discontinuity of personnel, and adaptation to new technological foundations.

Complexity confusion led to mistrust
When Silverthread leaders were asked to diagnose the health of the JDOE software, it was perceived as *very challenging* by everyone:

1. Enterprise leadership: Sustainment of JDOE was perceived as very high cost per unit value compared to other benchmarks of productivity within the enterprise. Deliveries of new versions to fix problems or to add new features were unpredictable, always over budget, and late.
2. Project management: The pressure from leadership to improve efficiency, coupled with the pressure from practitioners to commit to more realistic forecasts, put mid-level managers into a no-win situation where they were forced into gaming forecasts and progress reports.
3. Technical teams: Over two thirds of their duty cycle was mired in overhead activities that were boring and low value. They felt choked by minutia and process controls. Management seemed out of touch with the difficulty of changing a codebase where every change they made had unpredictable unintended consequences elsewhere. They had no objective evidence to persuasively make the case that their job is harder than it seems.

Practitioners begged us to expose their architectural complexity
After presenting a seminar on managing complexity and design quality, a few of the practitioners of JDOE requested that we scan their system. They explained their situation in stark terms.

> *Our team's software change productivity is low compared to other systems being maintained in our organization and other external benchmarks. Our leadership believes that our Ada codebase is relatively simple because our code quality metrics are good. They question the capability of our team and our process as the suspected reasons for our low productivity. They want us to attend SCRUM training and improve our methods with "more agile techniques". We know our team is strong and we are competent in modern agile methods. We believe the reason we are unproductive is because our codebase is extremely complex, the architecture is degraded, and each change we make introduces unintended consequences in other components that we cannot foresee. Can you please, please, scan our system? We are sure that it will provide objective insight that help us all understand the true situation.*

Quantifying design complexity opened people's eyes
Silverthread's scans and reports provided objective measures, or more honest measures, of both code quality and design quality that helped each constituency understand the situation better. The code quality across JDOE components (Figure 1) showed that from this aspect, JDOE did appear to be relatively simple when benchmarked against Silverthread's empirical database of 1000s of projects. Since JDOE tracked code quality by measuring McCabe complexity for each component, project management and enterprise leadership felt confident that the codebase was not that complex.

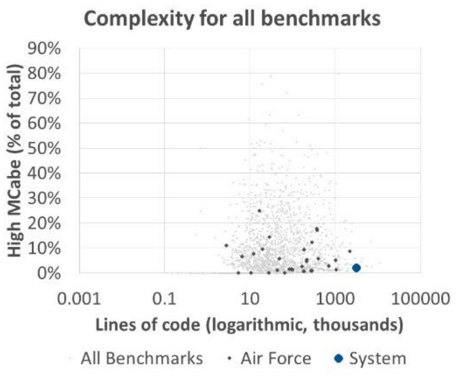

Figure 1: Code quality measures compared very favorably

Silverthread's scans also analyze and report on architectural complexity and design quality. This was a perspective that was new to JDOE teams. Figure 2 is a visualization of JDOE from a Silverthread CodeMRI® report. The red portion of the diagram shows a "core" of 11,000 files that are circularly interdependent on each other, directly or indirectly. The core is the component that is most complex and least hierarchical.

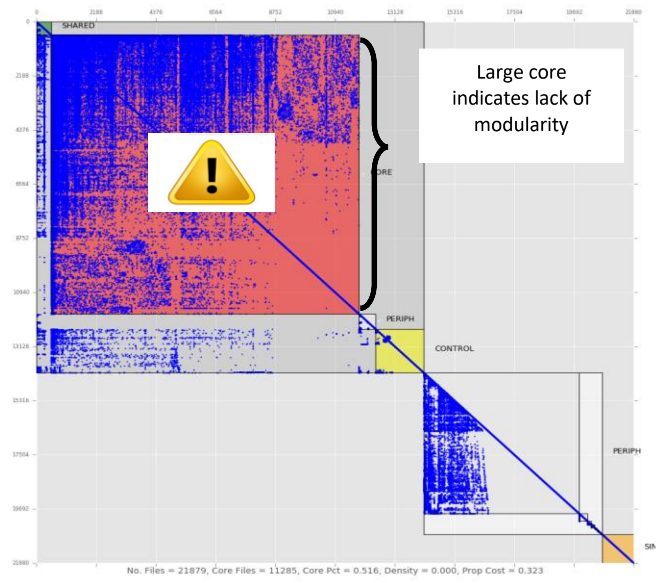

Figure 2: What a very complex code base looks like

The design quality of JDOE (Figure 2) was off-the-chart complex when benchmarked against Silverthread's empirical database. This was the missing link in understanding the true JDOE situation.

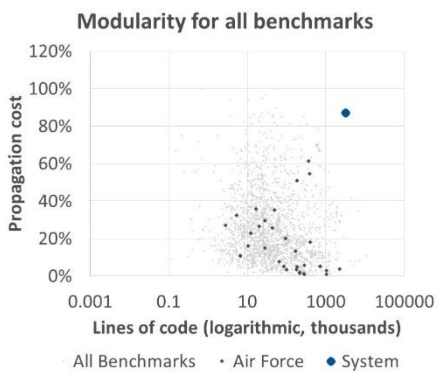

Figure 2: Design quality measures were off-the-chart high

How is design quality different from code quality?
Code quality and design quality are complementary product measures, and both can be extracted objectively from an evolving code base. Code quality assessments analyze the parts; design quality assessments analyze the whole. Developers can identify and fix code quality issues without having much impact on design quality. Code quality tools identify issues within a part by scanning and analyzing specific lines of code. But these tools do not quantify design quality. That is Silverthread's forte.

Silverthread's design quality assessments provide insight into the architectural properties of code bases that make them more manageable and understandable. When you quantify the relationships between the parts and the larger scale structures that they form, you can understand the macro-level health of the forest along with the micro-level health of the trees. Design quality tools identify complexity issues by scanning and analyzing dependencies among all the parts. These insights include visual summaries of modularity, cohesion, and coupling and quantification of hierarchy and cyclical dependency.

By scanning a code base, you can extract the structure of a system as it really is, which is frequently much different than what it was intended to be. Design documents or design models capture intentions, but because they are supplementary artifacts that rely on manual change propagation, they are frequently wrong or out of synch with the evolving coded product.

Enabling more honest conversations and building trust
JDOE experienced significant maintenance difficulties for years. Developers felt choked by waste, rework, complexity, unanticipated side effects, and low morale. Program leadership suffered from an inability to meet user expectations within a reasonable timeframe and the loss of credibility in forecasting cost, quality, and release targets. When complexity grows and measurements are largely guesswork, trust between development teams and program leadership dissolves. Quantifying design quality directly from the evolving code base delivers a critical quid pro quo: less overhead for practitioners and more insightful dynamic control for management. When practitioners and managers use the same measures, trust grows. Increasing trust enables leaner production by reducing sources of overhead, unnecessary rework, and waste. Trust is the currency of lean engineering efficiency.

Most organizations like JDOE already use traditional project management measures like those shown in Figure 4. These process measures provide only half of the insight you need. Design quality and code quality are the other half, the more important product measures that teams need to steer software outcomes more predictably.

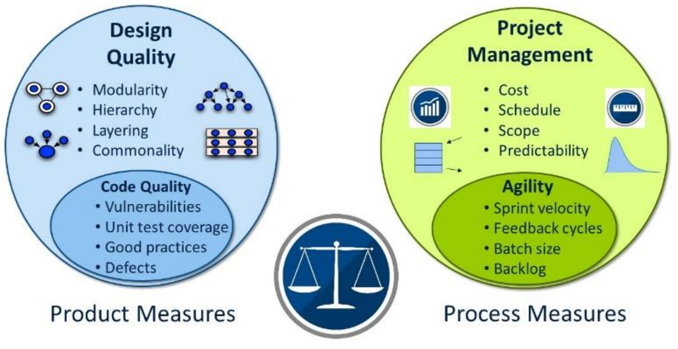

Figure 4: Product measures complement process measures

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