silverthread

Play offense: How to pull the trigger on a software refactor or rewrite

Measuring design quality can help you act decisively and manage change.

Why refactor or modernize a code base?
A software system is well designed when its code base adheres to certain principles that enable agility, maintainability, and understandability. These include modularity, layering, hierarchy, inheritance, and reuse. When designed well, code bases have properties that allow individual parts to be changed separately without being overwhelmed by unintended consequences. Unfortunately, design quality tends to degrade as a code base grows and evolves. It becomes difficult or impossible for developers to visualize the system and understand how it works. When this happens, estimation fails, defect rates increase, productivity drops, schedules slip, and costs rise.

Efforts to improve and stabilize design quality can yield substantial rewards in agility, maintainability, and cost. Figure 1 shows the result of an effort to improve design quality in a video game development studio. After a refactoring effort where design quality was improved in objectively measurable ways, the enterprise could release many more revenue-generating games per year.

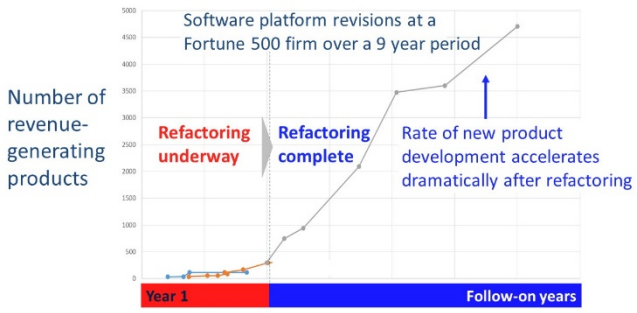

Figure 1: Revenue impact of one successful refactoring effort

Refactoring is an untapped opportunity.
Despite the benefits, there is a systematic underinvestment in software modernization. Leaders play defense instead of offense. Some of the reasons include:

- **Targeting:** Selecting which systems in a portfolio should be done first is a complicated decision.
- **Resources:** Refactoring impacts resources allocated to immediate concerns, such as new features and bug fixes.
- **Attribution:** The software economic impacts of poor design often result in finger-pointing and blame.
- **Problems found late:** Risk, quality, productivity, and morale problems appear long after code has degraded.
- **Track record:** Past refactoring or rewrite efforts failed to improve design quality or software economics.
- **Uncertainty:** The economic returns of modernization are long-term and are forecast with some uncertainty, whereas the costs are near-term and tangible.

Large-scale refactoring efforts have traditionally presented significant technical risk. Some development teams have gone down blind alleys and produced new systems no better than the ones being replaced. This is because managers and architects have struggled to:

- Identify structural issues in the code base accurately and objectively
- Prioritize the elements with the highest ROI
- Determine which changes will fix underlying problems

During 15 years of Harvard/MIT research capturing a diverse spectrum of systems, we found recurring patterns of good and bad quality and correlated their impact on software economics. Figure 2 shows the results of one study of a large commercial software product. We found that teams developing and maintaining code with better design quality were more than twice as productive and spent far more time playing offense (implementing features) than defense (fixing bugs).

Figure 2: Design quality impacts economic outcomes significantly.

When developers tell you there's a problem
Developers and architects know when a refactoring effort should be considered. They can tell you when team productivity, quality, and morale are suffering. Many times, developers will not be able to explain the reasons for these problems. This is because poor design quality is caused by *indirect* linkages coupling the system in ways developers are not able to understand. Their mental models will often differ from how the system is actually structured.

Silverthread CodeMRI® scans analyze a code base so that you can quantify and visualize design quality problems. This includes a breakdown of modularity, unacknowledged dependencies, cyclical dependencies, and other complexities that make systems challenging.

Focus on design quality, not code quality
Figure 3 shows Silverthread design quality visualizations for two systems: the before and after of a very successful rewrite conducted by the U.S. Air Force. The left side of the figure shows a C++ system with a core component (the red box) of files that were cyclically interdependent. Modularity and controlled dependencies had degraded, and this component experienced unintended consequences whenever it was changed. In the refactoring effort, the team consciously chose to improve design quality rather than

replicating the original design. After the rewrite, the new system showed significantly less cyclicality, better hierarchical control, and a significant improvement in maintenance productivity and agility. Improved design quality resulted in better software economics.

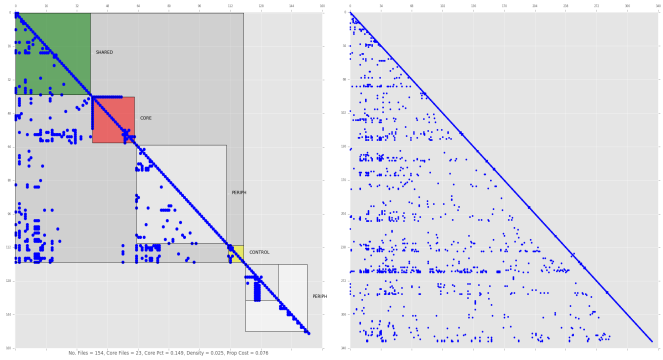

Figure 3: Before and after views of a successful refactor effort

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. Many code quality tools measure McCabe complexity. SonarQube identifies "bad smells" such as "if" or "elsif" constructs with no final "else"; Coverity identifies likely buffer overflows; and HP Fortify identifies likely security vulnerabilities. But these tools do not quantify design quality. That is Silverthread's forte.

When software economic models tell you there's an ROI

Successful modernization should result in improved design quality and pay dividends in better productivity, lower cost, lower risk, and greater business agility. Some systems are challenged, and can benefit from refactoring. Others have degraded to the point that rewriting from scratch makes more sense. A refactor or rewrite isn't always called for. Sometimes the most financially responsible choice is to continue incremental maintenance, because the cost of a rewrite or refactoring might be greater than the benefit.

Silverthread's CodeMRI® technologies help you explore the ROI of refactoring, rewriting, or leaving a system as-is. Figure 4 shows benchmarks from a system that suggest significant challenges. The scores in the right-hand column identify the percentage of code bases in our empirical database (compiled from thousands of scanned projects) that score better than the code base being assessed. Two of three high-level design quality metrics score poorly relative to comparable systems. Based on a design quality analysis, statistical models project that every new 1,000 LOC feature developed in the system should be expected to take 80+ days to complete, cost over $50,000, and introduce or expose bugs in 400 LOC. A system such as this is a strong candidate for refactor or rewrite.

Software economics		**8%**
	Value	Score
Maintainability	379	8%
Number of bug lines of code per 1000 LOC		
Agility	83	8%
Days to produce 1000 lines of new code		
Cost	$54,806	8%
Cost to produce 1000 lines of new code		
Design quality		**38%**
		Score
Modularity		8%
Based on propagation cost		
Cyclicality		8%
Based on core file density		
Complexity		100%
Based on high McCabe density		

Figure 4: Benchmarks from a Silverthread CodeMRI® report

Give your leadership team the insights they need.

Without diagnostic tools to measure and manage change more objectively, software maintenance teams will flounder in subjective guesswork and indecision. Refactoring alone is no guarantee of success. Successful steering requires leadership to:

- Capture and visualize the design to quantify complexity and design quality
- Give developers objective information about the architectural structure and the difference between design intentions and the as-is coded reality
- Allow development teams to remove design problems and prevent new ones from emerging
- Monitor progress of design quality improvement and subsequent software economic benefits

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