Architectural_complexity

September 17, 2024

1 The Cost of Architectural Complexity

System design and the cost of architectural complexity, **a 2013 dissertation by Daniel J. Sturtevant for MIT**. This study set out to measure the link between architectural complexity (the complexity that arises within a system due to a lack or breakdown of hierarchy or modularity) and a variety of costs including developer productivity, software quality, and turnover.

1.1 Summary of the paper

It is well known that architectural patterns (including hierarchies, modules, and abstraction layers) should be used in design because they play an important role in controlling complexity. However, design is not easy, straightforward, or free.

Previously there had been little quantitative research focused on helping managers and designers understand the cost of complexity in an architecture. Without this evidence, it is hard to place a value on hierarchy and modularity in a system design. It is also hard to objectively weigh the value of refactoring efforts aimed at asserting (or reasserting) various principles of large-scale system design. This study sought to fill that gap.

1.2 Methodology and definitions

The study was conducted within a representative and commercially successful software firm with a large codebase that is both mature and growing rapidly. A variety of research methods were chosen to explore the links between costs and complexity. To summarize some of the core techniques and definitions used:

To measure architectural complexity, this study leveraged prior research that devised metrics for capturing the level of coupling between each software file and the rest of the system. The complexity metric leveraged is the McCabe score, which breaks the levels of complexity into the following categories: low, medium, and high complexity, and untestable.

There is also an architectural complexity classification that appears in this paper, which says each file can fit into one of the following categories: peripheral, utility, control, and core. Core files are the most architecturally complex. Peripheral components do not influence and are not influenced by much of the rest of the system.

Data to measure cost drivers (quality, productivity, and turnover) was obtained by mining corporate version control systems, change tracking systems, human-resources databases, and source code.

To explore the relationship between each of the three cost drivers and architectural complexity within the codebase, three regression-based analyses were conducted.

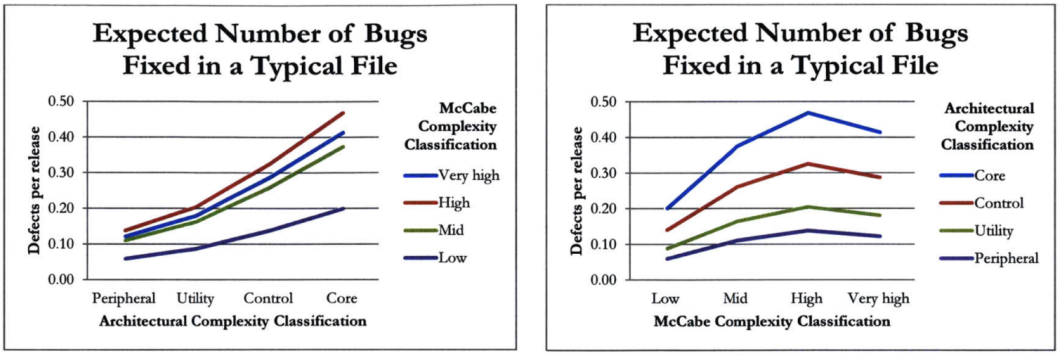
1.3 Here are the key findings from the paper:

Quality: Architectural complexity accounts for a three-fold difference in defect density

TL;DR: Architecturally complex source code files have a much higher defect density. The most complex code was found to have triple the defect density of the least complex code.

Summary of the analysis: The first phase of this study tested the hypothesis that architecturally complex files experience more defects. The study explored the relationship between the architectural complexity of individual files and the amount of bug-fix development activity that occurs in those files.

Specific findings: Files with high McCabe (complexity) scores are expected to have 2.1 times as many bug fixes as files with low McCabe scores. Changes in architectural complexity have an impact of roughly the same order of magnitude. When compared against the periphery, utility files have 48% more defects, control files have 2.4 times as many defects, and core files have 3.4 times as many defects.



Core files (the most architecturally complex files) have 3.4 times as many defects as do Peripheral files (which do not influence much of the rest of the system)

Productivity: Architectural complexity accounted for differences in developer productivity of 50%

TL;DR: Architectural complexity impairs the productivity of software engineers working with it. If a hypothetical group of engineers working in the least architecturally complex regions of the codebase were to be moved into the most complex regions, their productivity would decline (conservatively) by 50%.

Summary of the analysis: The next phase of the study explored the hypothesis that when developers work in architecturally complex files, their productivity is impaired. To test that hypothesis, the study looked at the relationship between the fraction of lines of code an individual contributes to "core" files during a release and their total number of lines of code produced during that release.

Note that this paper uses the term "productivity" however the study is limited to developer activity.

Specific findings: The results suggest that the effect that architectural complexity has on developer productivity is quite strong. "All else being equal, architectural complexity accounts for a near halving of the lines of code that can be produced by an individual in any given release as one moves from the periphery to the core."

Turnover: Architectural complexity has a significant impact on developer turnover TL;DR: Architectural complexity causes staff turnover. Software engineers working in the most complex regions of the code had a probability of leaving the firm that was ten times greater than their peers working in least architecturally complex code.

Analysis summary: The third analysis explored the hypothesis that developers working in architecturally complex files have a greater likelihood of leaving the firm. The study specifically looked at the relationship between the fraction of lines of code an individual contributes to "core" files, relative to peers, and whether that person left the firm during the 8 release windows studied or during the subsequent 4 years.

Findings: The link between architectural complexity and turnover was "surprisingly strong." A variety of controls were included in this analysis, each representing a sound alternative hypothesis for why a developer might leave the firm — and none yielded a stronger effect on developer turnover than architectural complexity.

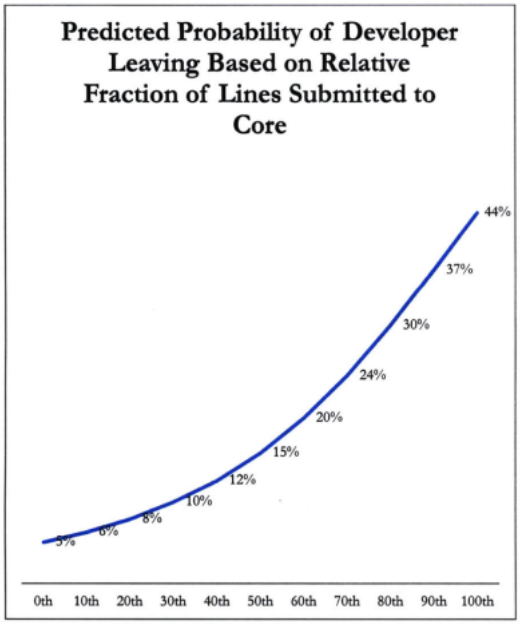


Figure 45: Predicted Developer Turnover (1)

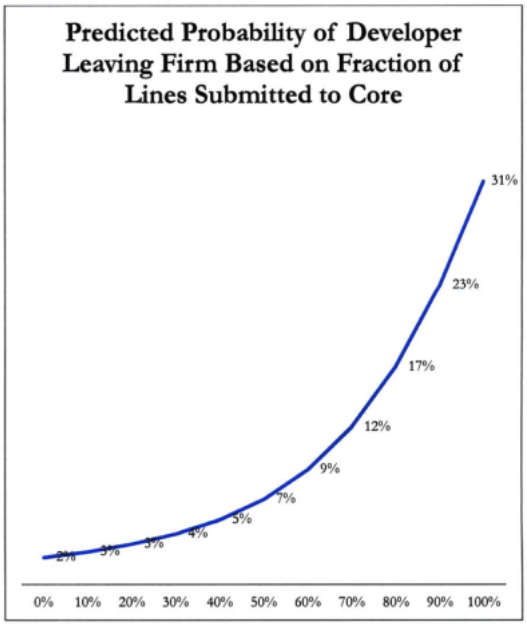


Figure 46: Predicted Developer Turnover (2)

Two sets of simulations were run to determine the expected probability that a developer would leave the firm (voluntarily or involuntarily) as a result of the complexity of the code they are working in.

1.4 Final thoughts

This study provides a convincing argument that refactoring efforts aimed at reducing architectural complexity have the potential to create enormous value for a business. If a study that says differences in architectural complexity could account for 50% drops in productivity and threefold increases in defect density isn't enough, take the final point on turnover. The impact of architectural complexity on turnover was found to be not only to be substantial and statistically significant,

but also to be of greater importance than a developer's tenure, fraction of effort in new (vs. legacy) code, and fraction of effort working on bugs.

One additional takeaway from this paper is a mindset shift: **While it is partially true that people cause bugs... We propose that complex architecture causes bugs, impairs productivity, and thwarts understanding. We propose that developers are in some senses its victims.** This healthier mental model gives organizations the ability to address the root causes behind defects and project failures.