Week 4: Mobility Bibliography

Nate Bachmeier

TIM-7140:Software Engineering

March 21, 2021

Northcentral University

# Mobility Bibliography

## Evolution of Mobile Apps

Zhang et al. (2013) compare the evolution of mobile development against traditional desktop engineering. The basis of their assessment comes from Lehman’s law, which claims software is always (1) continuing to change, (2) increasing in complexity, and (3) declining in quality. Next, they measured four open-source application’s code churn, total commits, feature enhancements, and bug fixes. The authors quantified each data point by examining GIT commit messages and applying heuristics. Afterward, plotting these metrics shows that the projects are continuing to evolve with more features and defects. Ultimately, the central goal of proving Lehman’s law remains inconclusive.

Their study is deeply flawed for multiple reasons. First, the four chosen applications are two programs built upon two platforms. Any results are not generalizable without increasing the sample set. Second, a direct correlation between complexity and commits does not exist. This limitation comes from the unique-style that individual developers follow. For instance, some users commit changes every block versus others who wait until the end of the day. Another set of inherent challenges arise from merging and squashing branches. These standard behaviors skew the total lines within each change.

There are also several limitations to using defects resolved as a measurement of quality and complexity. Most software engineering processes refrain from resolving defects that are not severe enough to warrant the regression risk. This “bug bar” naturally fluctuates in response to the business cycle (e.g., before versus after a major release) and can generate misleading signals. Assuming a static bug bar, an increase in defects likely represents new investments into app features.

## Mobile Development Processes

Corral et al. (2013) examine the growing interest in agile processes within mobile development. They perform a light survey of competing strategies such as Mobile-D, Scrum, and Lean Six Sigma. Under Mobile-D, businesses began following an iterative waterfall-like process for building mobile software. This approach was wildly successful in the early 2000s, with variations like MASAM (Mobile Application Software Agile Methodology) reducing life cycle overhead. After examining the evolution of agile methodologies, the authors attempt to map the different requirements of mobile development to agile characteristics.

Most of these agile permutations lack real-world sponsorship residing within academic journals at the time of writing. Since then, Scrum and Lean methodologies became everyday staples while Mobile-D fell to the wayside. The primary driver of those processes’ success is the ability to remove waste and improve time-to-market. Businesses that can release features more quickly and economically have the opportunity to be more competitive. However, those same organizations need to align their delivery cycles with customer’s ability to consume those changes.

Consider Philip Hue’s Smart Light App that runs on Android devices and controls Smart LEDs’ coloration. Regardless of Philip’s Continuous Integration and Delivery (CI/CD) pipeline, customers will not update this app frequently. Unless that behavior changes, the engineering team’s incentive is to release higher quality versions at a slower pace. In contrast, Facebook’s Android app renders XHTML responses from a remote server. Since Facebook users instantly consume any change, FB can risk quality and promote innovation quicker. When a feature regression occurs, the operations team centrally deploys an update, restoring the previous experience.

## Challenges with Mobile Development

Joorabchi et al. (2013) conducted semi-structured interviews with twelve mobile development professions. Responses from participants qualitatively explore challenges across different aspects of mobile engineering. For instance, a subset of questions examines the impact of Android versus iOS platforms. There is also a particular interest in how developers spend their time (e.g., designing versus testing). After collecting and analyzing the ground truths, the researchers conclude that heterogeneous platforms create the most overhead. Developers are also overwhelmed needing to test their code across various hardware configurations.

Ratification of the HTML 5 standard took place two years after this paper. Developers adopting this framework can abstract many platform-specific details, freeing teams to focus on value-differentiating aspects of the programs. There are still niche areas, such as 3-D gaming, that cannot embrace those features. When engineering groups must use native platforms, they can often rely on open-source libraries that implement boilerplate code. Since the mobile world is far more eight-years more mature, it would be interesting to revisit this portion of the interviews.

The last decade also brought significant innovation into mobile testing. Previously developers would purchase dozens of dedicated devices to confirm hardware dependencies or rely on generic simulators for finding defects. Modern simulators can verify more scenarios than ever before. Additionally, cloud services enable renting devices for short periods on-demand. Outside of raw technical improvements, the software industry embraces combined engineering patterns (e.g., DevOps) because it improves efficiencies. When an engineering team owns their code from end-to-end, it eliminates any assumptions that someone else will find the issues. Businesses need to promote that accountability as it results in higher quality software.

## Suitability of Agile for Mobile

## Improving Mobile Capabilities