Software Development Practices, Barriers in the Field and the Relationship to Software Quality

Beth Yost[1], Michael Coblenz[2], Brad Myers[2], Joshua Sunshine[2], Jonathan Aldrich[2], Sam Weber[2],
Matthew Patron[1], Melissa Heeren[1], Shelley Krueger[1], Mark Pfaff[1]

[1]The MITRE Corporation, Bedford, MA, 01730, United States
{bethyost, mpatron, mheeren, sekrueger, mpfaff}@mitre.org

[2]Carnegie Mellon University, Pittsburgh, PA 15213, United States
{mcoblenz, bam, sunshine, jonathan.aldrich}@cs.cmu.edu, samweber@cert.org

ABSTRACT

Context: Critical software systems developed for the government continue to be of lower quality than expected, despite extensive literature describing best practices in software engineering. **Goal:** We wanted to better understand the extent of certain issues in the field and the relationship to software quality. **Method:** We surveyed fifty software development professionals and asked about practices and barriers in the field and the resulting software quality. **Results:** There is evidence of certain problematic issues for developers and specific quality characteristics that seem to be affected. **Conclusions:** This motivates future work to address the most problematic barriers and issues impacting software quality.

CCS Concepts

• **Software and its engineering** • *Software and its engineering~ Software development methods* • *Software and its engineering~ Software development techniques*

Keywords

Software development; software quality; survey.

1. INTRODUCTION

Despite advances in software engineering, software systems being developed for the government continue to cost more, take longer to deliver, and be of lower quality than expected [1]. Critical infrastructure sectors such as healthcare, transportation, and energy depend on that software. To better understand the issues in practice, we conducted an exploratory study.

Using a survey, we gathered data on practices in the field for the requirements, design, build, and test phases of software development. As improving software quality in practice and improving the developer experience were key long term objectives, we asked about the barriers faced by developers and software quality. The key barriers identified motivate future work to better understand and address issues with task switching, getting enough time for development, missing documentation, understanding design rationale behind a piece of code, and finding code related to bugs and behaviors to be changed. The results provide evidence of the value of certain practices (e.g., having a clear architecture, unit testing) on specific software quality characteristics such as maintainability and evolvability. The results can be used by researchers to focus their work and managers to improve their workplaces and the quality of software produced.

2. RELATED WORK

Software quality and productivity of software engineers have been studied since at least the 1968 NATO conference [2]. Since then, researchers have attempted to understand the relationships between software engineering practices and the outcomes of software projects. In spite of this work, however, large software projects continue to fail [3, 4].

Dybå et al. argued that the context of software development is critical when evaluating the success of software development practices [5]. For example, the US government commonly acquires software via a contracting process that differs from how companies buy software. The Software Engineering Institute conducts independent technical assessments of software projects. One study of recurring problems across twelve US Air Force acquisition programs reported inadequate project management office (PMO) expertise and staff; high PMO staff turnover; requirements scope creep; inadequate requirements; and lack of functional requirements baseline [6]. The results of this study report the relationship of practices for which others have argued such as clear and stable requirements with specific quality characteristics such as software maintainability and reliability in the field.

Cleland-Huang argued that often the problem is one of requirements [7]. On the basis of experience with large software projects, Jones argued for a large number of best practices in software engineering in many areas, including requirements, architecture, and testing [8]. In addition, some experience reports exist regarding certain software development practices in government-related contexts. For example, Upender's experience report describes the difficulty of using agile methodologies over a period of time [9]. The results of this study relate practices such as unit testing with multiple software quality characteristics including evolvability and maintainability.

Of course, the causes of poor software project outcomes are typically multifaceted, which is why our survey took a broad perspective regarding causes of software project outcomes. Rather than basing recommendations on an individual's experience, our work focused on gathering data on practices in the field and correlating these with the respondents' subjective ratings of specific software quality characteristics.

ESEM '16, September 8-9, 2016, Ciudad Real, Spain.
Copyright is held by the owner/author(s). Publication rights licensed to ACM. ACM 978-1-4503-4427-2/16/09…$15.00
DOI: http://dx.doi.org/10.1145/2961111.2962614

3. METHOD

3.1 Participants

We distributed the survey through software development related mailing lists and contacts at various companies. Fifty participants voluntarily responded to the anonymous online survey. Instructions requested that all participants be over 18 years old and be involved in software development professionally. Participants had the option to participate in a raffle for an Amazon Fire tablet upon completion.

The primary job of most respondents was software developer or project lead (36 out of 50), but also included architects, designers, managers, and testers. All but one had a college degree and most had degrees in computer science, electrical engineering, and/or software engineering. Most were experienced developers, with 19 involved with software development for more than 20 years, and only 3 less than 5 years.

The participants represented developers of both government and commercial software. Thirty-seven of the participants currently work for a federally funded research and development center (FFRDC), 10 for a commercial company, and 3 for other types of companies or the government. FFRDCs operate in the public interest, free from conflicts of interest, providing objective guidance to U.S. government sponsors. Software developed by FFRDCs is often prototype software to show a proof-of-concept. Many government agencies do little software development of their own, hiring contractors to develop many software systems.

3.2 Materials

We constructed an online survey that contained 46 main questions, many with sub-questions. These were organized into three sections: background (job function, gender, age, education, years involved with development, number of programming languages, codebases used in career, category of employer), current project (customer category, domain, product category, people on project, developers on project, clear intended architecture, how often requirements change, process used, tools used, software quality characteristics), and barriers, described as "barriers or problems that you personally have in performing your job". Standard Likert scales were used to measure the extent to which tools or processes were used and for rating software quality characteristics. The software quality characteristics came from ISO/IEC 25010:2011, with evolvability and overall quality in general added. The survey was piloted with eight volunteers and updated as appropriate.

3.3 Procedure

The online survey took approximately 30 minutes to complete. The instructions requested help understanding and assessing how tools and processes impact project execution and the resultant software. The participants were instructed to answer questions based on their current or most recently finished significant software development project, for which they had good working knowledge and, if possible, to select a project that was being developed for the government.

The independent variables were the customer for the current software project, software category, clarity of requirements and design, extent of code for testing and error handling, the software processes used, the software development tools used, and the barriers. The main dependent variables related to software quality.

4. RESULTS

4.1 Software Quality

We measured quality according to subjective self-reported ratings. The first question asked: "Considering the code developed as part of this project by the whole team, please rate the following attributes:"

- Number of Software Defects (design or code errors, bad fixes)
- Severity of Known Software Defects

The second question asked: "Considering the code developed as part of this project by the whole team, please rate the following software quality characteristics:"

- Functional Suitability (functionality is complete and correct)
- Performance Efficiency (time, resource use, and capacity)
- Compatibility (software interoperability)
- Usability by users (ease of learning and use, error prevention)
- Reliability (maturity, availability, fault tolerance)
- Security
- Maintainability (modular, re-usable, modifiable, testable)
- Portability (ease of migration to new platform)
- Evolvability (ease of changing code)
- Overall Quality in general

Participants were asked to rate each on a 5-point Likert scale that went from "Very Low" to "Very High". There were also options for "Not relevant to this project" and "Don't know". Significant correlations are shown in Table 1 and are summarized next.

4.1.1 Overall Quality

The overall software quality ratings are shown in Figure 1. Responses of "Not relevant to this project", "Don't know", and blank are not shown. Functional suitability had the most "High" and "Very high" responses (34) while security had the least (13). The code defect responses are shown in Figure 2.

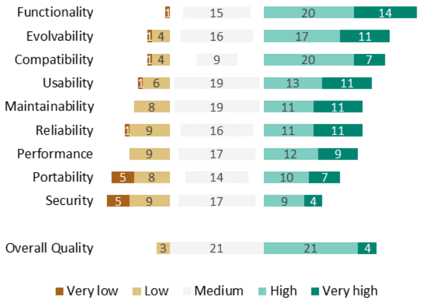

Figure 1. Software quality overall (# responses out of 50)

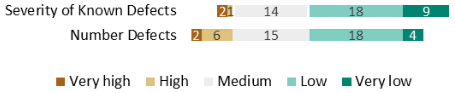

Figure 2. Code defects overall (# responses out of 50)

4.1.2 Quality by Software Customer

We compared the ratings for software developed specifically for government customers versus for commercial customers. The options allowed participants to select all customer classifications that applied and included: Internal to your company or organization, commercial company, non-profit company, military, non-military government, consumers, and other. To compare between groups, a category for Government (*n*=27) was created by combining "military", "non-military government", and one "other" response listing a civilian government agency. A category for Commercial (*n*=6) was created by combining "Commercial company" and "Consumer". We did not include responses of internal (*n*=7) or any that were combinations of categories (*n*=10).

		Number of Defects	Severity of Known Defects	Functional Suitability	Performance Efficiency	Compatibility	Usability	Reliability	Security	Maintainability	Portability	Evolvability	Overall
Rqmts	Clear requirements							.30,.037,48					
	Frequently changing requirements									-.32,.025,49			
	Clear architecture	-.31,.04,45				.33,.033,41		.31,.03,48		.41,.003,49	.38,.012,44		.41,.003,49
Processes	Waterfall					-.35,.034,38							
	Test-driven development				.32,.035,45							.41,.005,47	.33,.025,47
	Code reviews								.31,.048,42				
	Unit testing		-.32.,.034,44		.30,.042,47		.39,.005,50		.31,.043,44	.45,.001,49	.35,.02,44	.50,.000,49	.47,.001,49
	System testing				.36,.014,47								
	Iterative design					.32,.041,41					.35,.02,44		
	Usability evaluations					.45,.003,41	.59,.000,49	.32,.027,48			.31,.042,44		.38,.007,48
	QA testing								.33,.035,42				
	Writing down design decisions				.31,.039,46		.29,.042,49			.30,.041,48		.32,.026,48	.29,.049,48
Tools	IDEs								.31,.047,43				.40,.005,47
	Source version control					.46,.003,40					.36,.019,43		
	Debuggers						.29,.041,49						.30,.039,48
	Bug tracking database				.33,.024,46			.30,.044,47					
	Project management tools							.35,.019,45					
	Security assessment tools		.40,.009,41								-.32,.043,41		
	Static code analysis tools						.35,.015,47						
	Dynamic analysis tools				.33,.032,44								
	Automated testing frameworks										.34,.027,43		
Barriers	Switching tasks often due to other requests											-.32.03 47	
	Getting enough time for software development				-.42,.004,45					-.30,.038,47		-.38,.009,47	-.31,.033,47
	Documentation that is missing information		.50,.001,41	-.48,.001,47			-.42,.004,47	-.34,.023,45		-.39,.007,46	-.40,.01,41		-.41, .005, 46
	Understanding the design rationale behind a piece of code							-.30,.049,44		-.32,.034,45			
	Understanding code that I or someone else wrote a while ago	.32,.042,41		-.42,.004,46		-.34,.041,37		-.40,.008,44		-.40,.007,45			
	Convincing managers that I should spend time refactoring code									-.42,.004,45	-.34,.033,40	-.38,.011,45	
	Documentation that is out of date			-.40,.005,48		-.33,.042,39	-.31,.034,48				-.35,.023,42		-.33,.022, 47
	Finding which code is related to a bug or behavior to be changed	.45,.003,41	.35,.028,40		-.44,.003,43					-.58,.000,45	-.39,.012,40	-.44,.002,45	-.56, .000,45
	Understanding the impact of changes I make on code elsewhere	.38,.016,40	.33,.039,39	-.33.027,45	-.44,.003,42	-.36,.03,36				-.31,.039,44			
	Determining when the code has reached sufficient quality											-.43,.003, 45	
	Being aware of changes to code elsewhere that impact my code	.37,.018,40								-.31,.043,44			
	Finding duplicate code				-.37,.015,42								
	Turnover - having people important to the project leave		.38,.017,40	-.33,.027,46						-.33,.029,45			
	Usability of libraries, SDKs, or other APIs		.34,.031,41						.32,.041,42	-.36,.015,46			
	Finding who is currently responsible for a piece of code		.42,.008,39							-.31,.041,44			
	Coordinating with developers faraway geographically	.33,.041,39		-.34,.025.43									-.37, .015, 42
	Finding the best guidance online for development questions		.31,.047,42										
	Lack of tools to automate common tasks	.33,.034,42	.56,.000,41							-.36,.015,46			
	Learnability of debuggers			-.42,.005,44									
	Getting enough time with developers knowledgeable of code		.47,.005,35										
	Learnability of programming languages		.42,.006,41	-.37,.01,47									
	Finding who is currently modifying a piece of code									-.43,.004,44		-.39, .009, 44	

Table 1. Statistically significant (p< .05) correlations between design, tools, processes, barriers and software quality characteristics. Each cell contains Spearman's correlation coefficient (*r_s*), p value (*p*), and the number of responses (*n*).

Because of the small size of the Commercial group and the exploratory nature of the study, the p values were relaxed to .2 for this comparison only. We treated cases where the participant did not respond to a question as missing data. Given that relaxed threshold and corresponding tolerance of possible false positives, a Mann-Whitney test indicated that the: Severity of Known Software Defects was reported to be *lower* for software developed for Government customers (*n*=24, median= 2/Low) than for Commercial customers (*n*=5, median=3/Medium), *U*=35.0, *p*=.162. Portability was *higher* for software developed for Government (*n*=23, median=3, mean=3.14) than for Commercial (*n*=6, median=3, mean=2.67), *U*=94.5, p=.174. Usability was *lower* for software developed for Government (*n*=27, median=3) than Commercial (*n*=6, median=4/High), *U*=52.5, *p*=.189.

4.1.3 *Quality by Software Category*

We asked participants, "In which of the following categories does your product fall (the intended use of your system)?" The options were prototype, intended to be used, reference implementation, or other. Twenty-five were intended to be used and 19 were prototypes. The *reference implementation* (4) and *other* (2) responses were excluded from our analysis. Given the potential for major difference in quality between these groups, we compared the reported quality of software between them. A Mann-Whitney test indicated that the: Security was *higher* for software that was intended to be used (*n*=21, median=3, mean=3.43) than for prototypes (*n*=19, median=3, mean=2.44), *U*=95.0, *p*=.005.

4.2 Requirements and Architecture

Requirements: The survey asked participants whether their projects had clear requirements and how often requirements changed. For having clear requirements, 19 agreed or strongly agreed, 10 were neutral, and 21 disagreed or strongly disagreed. Having clearer requirements correlated with higher levels of software reliability, (r_s =.30, *p*=.037). Six said the requirements rarely, very rarely, or never changed; 19 said they occasionally changed; and 25 reported requirements frequently or very frequently changed. Having frequently changing requirements correlated with lower levels of maintainability (r_s=-.32, *p*=.025).

Architecture: The survey asked participants the extent to which they agreed that: "The codebase for this project has a clear intended architecture." As participants more strongly agreed with this, the number of software defects decreased (r_s=-.31, *p*=.04) and maintainability (r_s=.41, *p*=.003), portability (r_s=.38, *p*=.012), compatibility (r_s=.33, *p*=.033), reliability (r_s=.31, *p*=.03), and overall quality (r_s=.41, *p*=.003) all increased.

4.3 Processes

We asked participants to rate the extent to which they used various processes on a 5-point Likert scale that we then treated as scalar variables with values from 1 to 5. The question permitted a response of "Don't Know," which we treated as a missing value.

Overall Processes Used: Iterative design and system testing were used by more than half of respondents, while the waterfall model was used the least. The extent to which each type of process was used in shown in Figure 3.

Correlation with Software Quality: More extensive use of unit testing correlated with higher quality along eight software quality characteristics. The strongest correlations were between unit testing and evolvability and between usability evaluations and usability.

There were no significant correlations between quality and agile methods, but waterfall resulted in lower levels of compatibility (r_s=-.35, *p*=.034). There were more people using agile almost every time or always (22) than waterfall (5).

Other Process-Related Factors: As the number of people on the project increased, so did the number of software defects (r_s=.32, *p*=.03) and the severity of known defects (r_s=.38, *p*=.011), though the security weakly increased (r_s=.31, *p*=.039). Likewise, we asked specifically about developers on the project, and as that number increased, so did the number of software defects (r_s=.30, *p*=.043) and their severity (r_s=.35, *p*=.019).

Given the distribution in lines of code (LOC) responses (<10K *n*=9, 10K-100K *n*=22, 100K-1M *n*=13, 1M-10M *n*=4, >10M *n*=1), we regrouped the data into <100K (*n*=31) and >100K (*n*=17); we omitted the single >10M response as an anomaly. In comparing groups, there was a significant difference at *p*<.05 using the Mann Whitney U test: portability was higher when there were less than 100K LOC (*n*=29, median=3/Medium) compared to >100K LOC (*n*=13, median=2/Low), *U*=99.0, *p*=.014.

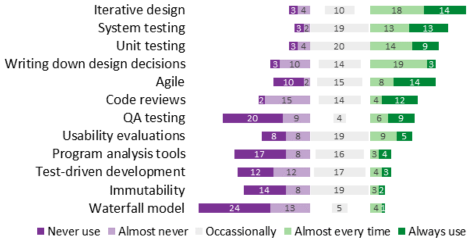

Figure 3. Extent of process use.

4.4 Developer Tools

Although adoption of version control was nearly universal, security assessment tools and program analysis tools were used infrequently. The extent to which each type of tool was used is shown in Figure 4. We also analyzed the correlation between tool usage and software quality (significant correlations are in Table 1). The strongest relationships were: use of source control was positively correlated with compatibility (r_s=.46, *p*=.003); use of IDEs was positively correlated with overall quality (r_s=.40, *p*=.005). Use of security assessment tools was positively correlated with severity of known software defects (r_s=-.40, *p*=.009). Perhaps these tools result in more knowledge of defects or these tools are being applied to systems that are known to have defects.

We asked about the criteria for selecting tools, who selected them, and how well they worked. To the extent that respondents more strongly agreed that their tools were modern and up-to-date, that significantly correlated with increases in functional suitability (r_s=.40, *p*=.004), usability (r_s=.38, *p*=.006), portability (r_s=.42, *p*=.005), and overall quality (r_s=.35, *p*=.014).

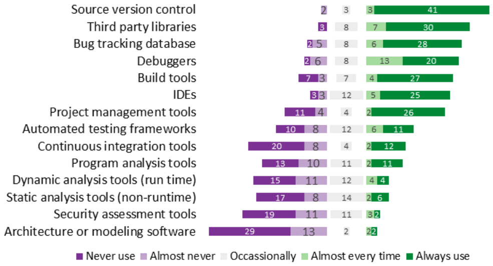

Figure 4. Extent of tool use

4.5 Testing and Error Handling

We asked, "Approximately what percent of the code is for error handling and recovery?" and "If there is extra code to test this project, for example a separate test harness or unit test, approximately what percent of the code is for that?"

On average 11% of code was for error handling and recovery, with a range from 1%-60%. On average, 14% of code was extra code to test, ranging from 0%-50% of total code. As the percent of code for error handling and recovery increased, so did the performance (r_s=.32, p=.045). As the percent of code to test the project increased, so did the maintainability (r_s=.35, p=.023).

4.6 Barriers

Participants rated how serious a problem each of the following was for them when performing their job. Figure 5 shows a sorted list of barriers across all survey respondents.

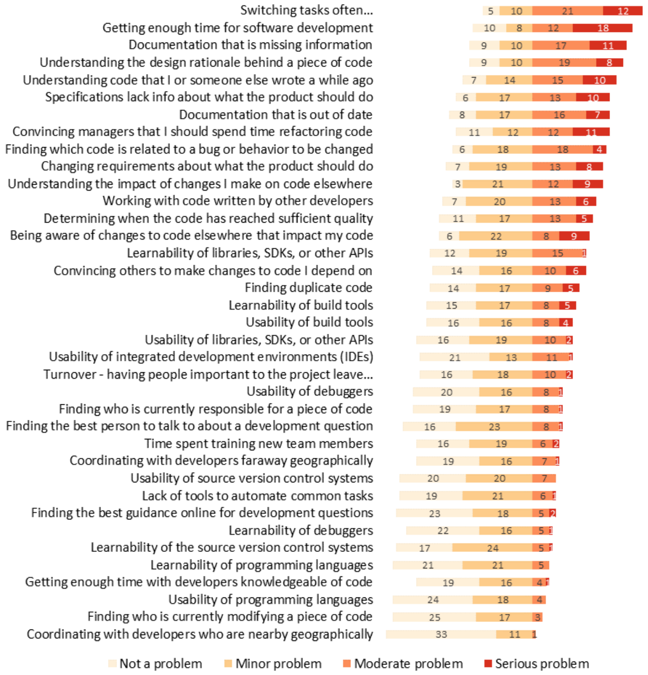

Figure 5. Barriers.

4.6.1 *Barriers by Software Customer*

The top four barriers for the government-only participants (n=27) were: getting enough time for software development, switching tasks often due to other requests from my manager or teammates, documentation that is missing information, and specifications that lacked information about what the product should do.

4.6.2 *Correlation with Software Quality*

Table 1 shows statistically significant correlations between barriers and software quality. The strongest relationships were between challenges with finding which code was related to a bug or behavior and low maintainability and overall quality.

A Mann-Whitney test was done to compare the groups that were and were not experiencing each barrier. We eliminated from the analysis the groups that were lopsided, where there were more than twice as many in the not/minor problem group or the moderate/serious problem group. For the remaining quality characteristics, there were three barriers where multiple characteristics were significantly different between groups:

Finding code related to a bug or behavior to be changed:

- Overall reported quality was higher when this was a minor problem (n=13, median=4) than when it was a serious problem (n=16, median=3), U=41.0, p=.005, effect size r=.55.
- Maintainability was higher when this was a minor problem (n=13, median=4) than when it was a serious problem (n=16, median=3), U=40.50, p=.004, r=.54.
- Evolvability was higher when this was a minor problem (n=13, median=4) than when it was a serious problem (n=16, median=4), U=44.00, p=.008, r=.51.

Understanding code that I or someone else wrote a while ago.

- Maintainability was higher when this was a minor problem (n=14, median=4) than when it was a serious problem (n=15, median=3), U=51.50, p=.02, r=.46.
- Functional suitability was higher when this was a minor problem (n=14, median=4) then when it was a serious problem (n=15, median=3), U=54.50, p=.03, r=.43.
- Reliability was higher when this was a minor problem (n=14, median=4) than when it was a serious problem (n=15, median=3), U=57.50, p=.04, r=.40.

Understanding the design rationale behind a piece of code.

- Maintainability was higher when this was a minor problem (n=15, median=4) than when it was a serious problem (n=14, median=3), U=55.00, p=.03, r=.43.
- Evolvability was higher when this was a minor problem (n=15, median=4) than when it was a serious problem (n=14, median=3), U=59.00, p=.046, r=.39.

Given that maintainability is impacted by all of these barriers, it appears that it is the characteristic that is most vulnerable overall.

5. DISCUSSION

The goal of taking a broad approach in this study was to identify promising areas on which to focus future research to improve the quality of government software, based on practices in the field and barriers faced. Follow-on studies should address specific barriers or measure increased adoption of certain best practices. The most problematic barriers require future work to address them. The results can be used by researchers to focus their work and by managers to identify changes to processes and tools that could improve the lives of developers and the quality of software being produced.

The data provide an indication of which of the many barriers we should focus on if we want to improve software quality: those problematic for the most developers or correlated most strongly with specific quality characteristics we want to improve. The most problematic barriers can generally be grouped into two categories: task-switching and getting enough time for software development; and documentation-related issues. Task-switching occurs when developers must switch among development tasks or when they work on multiple projects in an interlaced fashion. Task switching should be avoided where practical. Where not practical, switching tasks often can lead to difficulty in schedule estimates and lost time due to getting back into the zone [10]. Tools that help developers pick up where they left off and better deal with task switching may help mitigate these issues. Further study is needed to understand how to address time requirements for development. The second group of barriers had to do with missing documentation, understanding design rationale in code, or understanding code written a while ago. Tools that can generate documentation for legacy code, that encourage developers to document design rationale especially for unusual or complex modules, and that can keep the architecture models up to date as code is being written could prove particularly beneficial. Addressing these documentation-related barriers would address some of the largest reported problems and could help improve maintainability, functionality, reliability, and evolvability of the software.

We also saw the extent to which certain practices are used in the field. These correspond to opportunities to improve practice and the resulting software quality. While factors such as clarity and stability of requirements and architecture have long been known to be beneficial, our survey has tied these practices to the extent to which they are problematic in the field. We also tied them to the specific quality characteristics that may benefit from improvements in practice. Similarly, we saw the average amount of code dedicated to error handling and recovery and that the greater the percentage of code for that, the better the performance of the software, and the greater the percent of code for testing, the more maintainable. We found evidence of a move away from waterfall, especially for the development of government software: waterfall was the least-used process. Though agile methods did not appear to correlate with any increases in quality characteristics in this study, waterfall had a negative impact on quality.

We did not find evidence in favor of the hypothesis that commercial software would be rated higher quality than government software; in fact, government software was reported to have fewer known severe defects and be more portable. Commercial software was reported to be more usable. This may be because commercial companies have recognized the importance of usable systems while the government is only starting to recognize the importance. The government likely has greater need for enhanced security. In software intended for public use, there may also be greater need for more portable software given the variety of platforms used by the public. In general, the perception that government software is lower quality than commercial may not be accurate and may be a reflection of increased transparency and publicity when government software fails. Further study is needed to investigate.

6. LIMITATIONS

The study was a relatively small survey with only fifty participants. The large number of FFRDC participants may pose a threat to validity, which may be mitigated somewhat by the variety of domains represented.

Due to the small number who had a primary job function other than developer, no analysis was done to compare based on job function. While most of the responses would likely remain the same across groups (e.g., software quality), it is possible an architect or tester may use different tools or encounter slightly different barriers.

The software quality ratings were subjective and therefore may not agree with objective quality assessments. Further study should compare developers' subjective assessments to objective software quality measurements to evaluate these possibilities.

We performed a large number of statistical tests. With correlations there is no need to correct alpha because the correlation coefficient itself is an effect size. For comparisons between two groups, no correction is needed. Given the significance threshold of p<.05, however, it is likely that some of the results are random occurrences. These tests do not account for the interaction between factors. While we did exploratory regression and multi-factor analysis, we do not report the results here because more responses would be needed to produce a reliable model.

Conceptually, it is likely that development practices and barriers precede and therefore affect the software quality. However, inferring causality becomes a problem in cases where software quality may have *caused* the developers to use a particular approach or encounter a barrier.

For the exploratory comparison between government and commercial software quality, the small number of commercial product developers may cause a failure to detect important differences. Related, each group may have a systematic bias in how they see software quality. Further comparison between groups should include more developers and objective measures.

7. CONCLUSION

Our survey gathered data on development practices, barriers in the field, and their relationship to software quality. These results provide motivation for future research to address the key barriers and evidence of the extent of use and value of certain practices and tools in the field.

8. ACKNOWLEDGEMENTS

The authors would like to thank the respondents to the survey. Funding for this work comes from grants from MITRE, NSF under grant CNS-1423054 and the Air Force under Contract #FA8750-15-2-0075. This material is based upon work funded and supported by the Department of Defense under Contract No. FA8721-05-C-0003 with Carnegie Mellon University for the operation of the Software Engineering Institute, a federally funded research and development center. Any opinions, findings and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect those of the US Government.

Approved for Public Release; Distribution Unlimited. MITRE Case Number 16-1649. SEI Document Marking Number DM-0003591.

9. REFERENCES

[1] U.S. Government Accountability Office. (2013). Major automated information systems: Selected defense programs need to implement key acquisitions practices. (GAO Publication No. 13-311). Washington, D.C.: U.S. Government Printing Office.

[2] Software Engineering: Report of a Conference Sponsored by the NATO Science Committee, Garmisch, Germany, 7-11 Oct. 1968, Brussels, Scientific Affairs Division, NATO. Peter Naur and Brian Randell (Eds.).

[3] Goldstein, Harry. "Who killed the virtual case file?" *IEEE SPECTRUM* 42(9) (2005):18.

[4] Ford, Paul. The Obamacare Website Didn't Have to Fail. How to Do Better Next Time. Bloomberg Businessweek. October 17, 2013.

[5] Tore Dybå, Dag I.K. Sjøberg, and Daniela S. Cruzes. What works for whom, where, when, and why? on the role of context in empirical software engineering. In *Proceedings of the ACM-IEEE international symposium on Empirical software engineering and measurement* (ESEM '12). ACM, New York, NY, USA, 19-28.

[6] Novak, William and Williams, Ray. We Have All Been Here Before: Recurring Patterns Across 12 U.S. Air Force Acquisition Programs. Presentation at 2010 Systems and Software Technology Conference (SSTC). April 29, 2010.

[7] Cleland-Huang, Jane. IEEE Software. Don't Fire the Architect! Where Were the Requirements? IEEE Software

[8] Jones, Capers. Software Engineering Best Practices. McGraw-Hill, 2010.

[9] Upender, Barg. Staying agile in government software projects. Agile Conference, 2005, pp. 153-159.

[10] Parnin, Chris and Rugaber, Spencer. "Resumption strategies for interrupted programming tasks." *Software Quality Journal*, 2011. 19(1): pp. 5-34.