Large v Small Organization Software Development: Are Software Development Best Practices One-Size Fits All?

A thesis submitted in partial fulfillment of the requirements for the degree of Master of Science at George Mason University

By

Jeffrey Longo
Bachelor of Science
University of Mary Washington, 2005

Director: Dr. Thomas D. LaToza, Professor
Department of Computer Science

Spring Semester 2023
George Mason University
Fairfax, VA

Dedication

I dedicate this thesis to my family, whom I love most dearly. Thank you for your support during my graduate studies and research.

Acknowledgments

I would like to acknowledge the following people for helping make this research a reality.

To Dr. Thomas LaToza, my thesis director, academic advisor, and principal investigator on the research team. I am deeply in your debt for your consistent guidance through this long journey of exploration and enlightenment.

To Emad Aghayi, who contributed considerable amounts of time and effort as a member of our research team. You have my deepest thanks.

To my thesis committee, including Drs. LaToza, Offutt, and Johnson. I am extremely grateful to have you helping ensure this work is of the highest standards.

To Dr. David Rosenblum, chair of the department of computer science. Without your leadership of this exemplary department this would not be possible; my sincerest thanks.

Special thanks to my friend and colleague Margaret Redman, who inspired and pushed me to do my best work in my graduate studies.

I would also like to thank Mohammad Soleimani, my long time professional colleague and mentor. Without your encouragement I would not have begun this academic adventure.

Thanks should also go to all of the interview participants and survey takers who took time to provide the information needed to complete this research.

Table of Contents

List of Tables

List of Figures

Abstract

LARGE V SMALL ORGANIZATION SOFTWARE DEVELOPMENT: ARE SOFTWARE DEVELOPMENT BEST PRACTICES ONE-SIZE FITS ALL?

Jeffrey Longo

George Mason University, 2023

Thesis Director: Dr. Thomas D. LaToza

This research examines the differences in the way large and small organizations approach software development to better understanding if small organizations should attempt to model large organizations' practices. Practices examined include software development methodology, development tooling, how organizations deal with legacy software, documentation practices, testing practices, and how organizations differ in their code and development practice quality metrics. Mixed method research was applied starting with a qualitative semi-structured survey of 11 participants followed by a broader quantitative survey.

Our analysis showed differences that could be statistically tracked to organization size in 4 of the 6 practices researched. Semi-structured interview responses did not always match results from the survey, indicating there may be common misconceptions that do not match reality when it comes to how large and small organizations operate. In one significant finding, a statistical correlation was discovered between large organization size, worker specialization, and the use of formal test plans on a low change failure rate.

We conclude that the results indicate that small development organizations are not attempting to model larger development organizations, but are rather choosing development practices that best meet their needs. The findings from this research may be relevant to academic curriculum planners for computer science and software engineering fields. Curriculum geared towards large organization practices may not adequately prepare students for the demands of a small software development organization.

Chapter 1: Introduction

Large software development organizations are often seen as the pre-eminent standard bearers for software development tools and practices. Many large companies maintain developer blogs to tout their latest advancements and describe how they do their work. For example, Google maintains a developer blog[1] and Netflix has a tech blog[2]. Microsoft employs paid developer advocates such as Scott Hanselman to produce content in various media formats including blog postings and various video and audio formats[3]. Yet Hanselman himself noted that oftentimes these very blogs are out of touch with what many other companies and developers are doing[4]. More recently, Jean Yang tweeted[5] and blogged[6] about how she has observed that the standards and tools created by large software development organizations often do not working for small software development organizations.

There are many things that differentiate large and small software development organizations. In this paper we differentiate large and small organizations at the 100 software-related employee marker. Beyond size, large organizations typically utilize specialized workers whereas smaller organizations utilize generalized work-forces[6]. Large organizations are also much more risk adverse than smaller organizations, especially when the small organization is a startup[7]. Resources at smaller organizations can be extremely limited[8].

However, in both Yang and Hanselman's articles highlighting how large and small organizations operate differently, little hard data was presented to back up their claims. I continue to hear questions in the industry like "if a process or technology works for a large organization, why shouldn't my small organization adopt it?" Of the research that has been performed, none of it directly compares how large and small organizations perform software development differently. In addition, differing definitions of large and small make meta-analysis impossible.

In this thesis, I examine whether a small organization should try to mimic a larger organization's software development processes. In order to do this, several individual software development processes are examined from a standpoint of how they differ between large and small software development organizations. Interview data and statistical models are presented that show strong associations between the size of an organization and some of the processes examined.

The results of this research will not only provide the answer to whether small organizations can or should mimic larger organizations, but also lead to questions about how higher education students are currently being taught computer science and software engineering topics. Should the practices of small organizations be incorporated to better prepare students for what they may encounter in the workforce? Would a student that has been prepared for working at a large software organization be equipped with the skills they need to be successful at a small software organization?

Chapter 2: Related Work

The following research examined various aspects of software development relevant to my research. Some of this research directly examined the practices employed by small software development organizations with an emphasis on startups. Other research focused on the software development practices themselves that are examined in the research questions of this paper.

Research that focused on how organization size affected software development practices showed that small organizations had difficulty trying to implement the capability maturity model (CMM) - not only due to lack of funding, but also due to lack of applicability[9]. Small software organizations that are startups were found to work in a drastically different development environment. This environment was described as [8][7]:

1. lacking a historical base for estimating cost and time
2. engaging in highly risky behavior
3. working with severely limited resources
4. being highly reactive and rapidly evolving
5. lacking clear and stable requirements
6. focusing on only one product

For startups, phrases such as "done is better than perfect" and "move fast and break things" are common themes[7]. Customer involvement and ensuring the solution is fitting a real market need are key criteria for these organizations' success - much more so than a reliable product[10] [11]. For these reasons, startups often use evolutionary prototyping to ensure their product maintains market fit[7].

When software development methodology is examined, Agile and Lean methodologies are routinely found to be prevalent in small organizations [12][13]. Large organizations meanwhile utilize scrum across multiple teams that combine results either via pipe-lining or some other means[14]. Other research focuses on how to scale Agile across large organizations, which is a principal goal of SAFe[15].

In examining the kind of skill sets employed by large and small organizations, small organizations were found to do best with generalists[16], whereas large organizations used specialists and distinct development roles such as software architects, business analysts, and test personnel, in addition to personnel that focused on user communication[14].

For software quality and organization efficiency practices, Google - a large organization - has released yearly "State of Devops" reports that contain 4 key metrics to determine team equality and efficiency[17]. We use these metrics in our research. Regarding static source analysis, Google described how they were able to most effectively employ static source analysis via pull requests[18]. Static source analysis has been argued to be used more in small and medium sized business as a cost effective means of achieving better quality[19].

Software testing was found to be one of the least structured software development activities performed by small organizations[20]. Test methodology in startups was found to focus on ensuring base functionality works rather than ensuring any quality requirements are met [13]. In looking at the broader state of software testing without the consideration of organization size, most developers prefer test driven development but only 26% actually practiced it[21]. The highest percentage of software defects originating from either requirements problems or design problems[21].

A study of the state of software documentation in Agile teams showed that over half of developers in their study found documentation important or even very important[22]. At the same time, they noted too little documentation was available in their projects. This study, however, did not give a breakdown by size of organization. Regarding the use of UML, previous research has indicated that its use is not extremely widespread in industry[23].

For software development tooling, differences were uncovered between large and small software development organizations when it came to global software development tooling. Small businesses did not typically engage in global software development and therefore was not aware of the tools used, whereas large organizations had trouble with the tools themselves[24]. Much of development tooling has been observed to be developed by large software development organizations and created with specialists in mind, which makes them less suitable for use by small organizations[6].

Despite the related research cited above, there remains a significant gap when attempting to directly compare the practices of large and small organizations. Available research has generally focused on specific sub-topics of software development or on practices of either large or small organizations without directly comparing them. The ability to employ a meta-analysis to existing research is limited by the different definitions of large and small software organizations used in previous research. New research is therefore required to provide the necessary data to examine the differences between large and small software development organization practices.

Chapter 3: Method

3.1 Research Questions

In this thesis, I examine a common question small software development organizations ask themselves - "if big organizations do it, why shouldn't we?" This question examines whether small software development organizations should model their software development practices after large software development organizations. Answering this question requires first answering an even more fundamental question: *in what ways do software development practices differ between large and small software development organizations?*

Specifically, this thesis examines the following research questions:

- **RQ1**: In what ways do large and small software development organizations differ in their software development methodology?
- **RQ2**: In what ways do large and small software development organizations differ in their development tooling?
- **RQ3**: In what ways do large and small software development organizations differ in their dealings with legacy software.
- **RQ4**: In what ways do large and small software development organizations differ in their documentation practices.
- **RQ5**: In what ways do large and small software development organizations differ in their testing practices.
- **RQ6**: In what ways do large and small software development organizations differ in their code and development practice quality metrics?

3.2 Definition of Organization Size

To compare small and large software organizations, it is necessary to define each. One way to do this might be to look at definitions of small and large businesses. For example, the U.S. Government defines small businesses according to revenue or employee size based on their industry[25]. A small business may range anywhere from less than 1 million USD in revenue to over 40 million USD in revenue. Small businesses may have anywhere from 100 to 1500 employees depending on the industry or sub-industry they serve.

However, this definition is unhelpful for our purposes for multiple reasons. The U.S. Government may consider a company as a "large" business if it has 2000 employees, but only 5 of those employees may perform software related work. In this thesis, I wish to consider such a software development organization as "small." Conversely, an information services company with 999 employees where 900 of the employees all engage in software development may be classified as "small" by the U.S. Government, but this organization might still be considered a large software development organization.

In this thesis, I set aside monetary metrics and focus on organization size. I define a **small software development organization** as one that has 100 or fewer employees engaged in the creation of software. A **large software development organization** has 101 or more employees engaged in the creation of software.

3.3 Semi-structured Interviews

To understand how developers themselves view the practices in small and large software development organizations, my research team first conducted semi-structured interviews. Interview participants were recruited from the research team's personal and professional networks with care taken to recruit participants from a variety of backgrounds. 11 participants were interviewed in total. All had experience working in both small and large software development organizations over their career, with 5 currently employed in small software organizations and 6 employed at large software organizations.

The interviews were approximately 1 hour in length, were conducted remotely via Zoom, and were recorded and transcribed. Recordings were destroyed after transcription, theme coding, and coding review was complete.

The interviewer first presented the study's definition of large and small software development organizations. Interview questions then focused on examining differences participants perceived between the way large and small software development organizations approached software development. Appendix A lists the complete set of interview questions.

Themes were coded by reviewing interview recordings and transcripts and copying distinct thoughts expressed by participants to a google document. If the thought was new it would be added to the document at the end with a brief summary heading. If the thought could reasonably fit underneath an existing heading, it was put under that existing heading. These headings became the coded themes. Another research team member then reviewed the theme coding to ensure that themes incorporated into the thesis were coded correctly.

3.4 Survey

Based on analysis of the semi-structured interview data, my research team conducted a survey to examine if the perceptions of interview participants of practice differences held amongst developers in large and small software organizations. Respondents were asked to complete 33 survey items in total. The survey questions are provided in Appendix B.

Survey participants were asked to respond with regards to their current position only. After completing 14 items characterizing the current organization in which they worked, respondents then were asked to complete 19 questions on their use of tooling, legacy software, development duties, code quality, testing, documentation, and software development process. The survey was designed to take approximately 15 minutes to complete. My research team piloted the survey questions with a colleague of the author, revising questions as appropriate. Finally, I deployed the survey to respondents using Qualtrics.

Respondents were recruited from the research team's personal and professional networks as well as social media including Facebook, LinkedIn, professional coding communities, Twitter, and Reddit. Respondents were given an option to provide their email address in order to receive the survey results and participate in a raffle. Respondents were eligible to participate in a raffle for one of two $100 Amazon gift cards.

I received 507 completed survey responses. Upon analysis of the responses, I found that the responses likely included spam. To exclude spam responses, I developed a set of exclusion criteria, categorizing surveys responses as spam that included:

- Nonsense textual responses in free-form text fields. These responses were often duplicated amongst multiple entries. Example: "Be willing to receive information with a positive and open mind"
- Duplicate email addresses provided with different responses/IPs
- Duplicate IP addresses
- Start and end times that were within 5 minutes of other entries. Example: 5 entries all start at the exact same time and end within 5 minutes of each other.

After removing spam responses, 162 responses remained and became the working set. This working set represented a reasonably broad range of software development organizations. 107 of the responses were from small organizations and 57 were from large organizations. The business domain in which the respondents worked included 39 engaged in developing software for retail or as a service, 69 consulted or contracted with governmental entities, 34 consulted or contracted with non-governmental entities, 9 who were not primarily software organizations, and 7 that were "other." Respondent roles in the software engineering discipline ranged from 60 being software engineers, 22 being in IT, 48 being business analysts or product management, 19 from QA or testing, and 13 from management. For application domain, 35 respondents reported developing cloud native software, 46 made user installable software, 44 open source software, 22 embedded software, 10 a library or framework, and 5 said "other." Location data was not collected formally as part of the survey, however, in discussions with some of the respondents it is clear that responses were received from a global audience.

The working set was analyzed primarily as ordinal data using logarithmic regression. Likert scales were reduced to binary results for use as dependent variables in logarithmic analysis through separating the 50% of responses that responded most positively from the 50% that responded most negatively. Chi^2 significance tests were applied to determine significance of correlation of the variables selected. If the model was deemed significant ($p<.05$), then a model was construed with any additional independent variables that did not reduce the model's significance and added significance to the model with p values that were also less than .05. Descriptive analysis of the data was also applied in some situations where it was noteworthy. If Chi^2 significance tests return a p value $> .05$, no correlation of significance is observed.

Chapter 4: Results

4.1 RQ1: How do large and small software development organizations differ in their software development methodology?

Participants reported differences in the way large and small organizations used software development methodologies. Several participants suggested that larger organizations continue to employ a Waterfall development methodology despite often claiming to be Agile adherents. P11 used the term Agile-fall while P7 described it as *"Waterfall with sticky notes."* In speaking about his large organization experience, P10 said *"they pretend a lot of times to do Agile development when it's really Waterfall."* However, P6 pointed out that larger organizations do not always require adherence to the same methodology across all teams. P2 supported this, reporting one team to be practicing Waterfall while observing another team practicing Agile Scrum.

Participants reported smaller organizations to be less regimented in their adherence to software development methodologies - though P6 and P7 both noted that no company in their experience did everything "by the book." P10 reported on a small startup that followed the Agile Manifesto, but not any defined methodology. P8 and P2 agreed that that stand-ups and meetings were less frequent at smaller organizations. In contrast, much coordination work occurred in scheduled meetings at larger organizations. P10 went further: *"I remember one of the things that hit me the most [...] after the acquisition was we started having meetings about meetings."* P1 also related a larger organization acquiring a smaller org: "*I have been involved in such transitions [...] particularly the mothership organizations when they purchase a smaller company; like when [small company] was bought by [larger company],* *there was a lot of training and hand holding of change of processes that the bigger organization demands of the of the people there [being] assimilated into that new organization."*

Analysis of our survey responses revealed that the use of Waterfall was roughly equivalent between large and small organizations. A logistic regression revealed that organization size was not a significant predictor of the use of Waterfall (p=.68). However, large and small organizations differed in their use of Agile/Scrum ($\text{Chi}^2(1) = 7.86$, p .005, n = 162) and Agile/Rapid Prototyping ($\text{Chi}^2(1) = 8.48$, p .004, n = 162). Scrum was used by 38.2% of large organization respondents and only 17.8% of small organization respondents - a spread of 20.4%. Rapid Prototyping was used only by 18.4% of large organizations but 40.2% of small organizations - a spread of 22% [Fig 4.1].

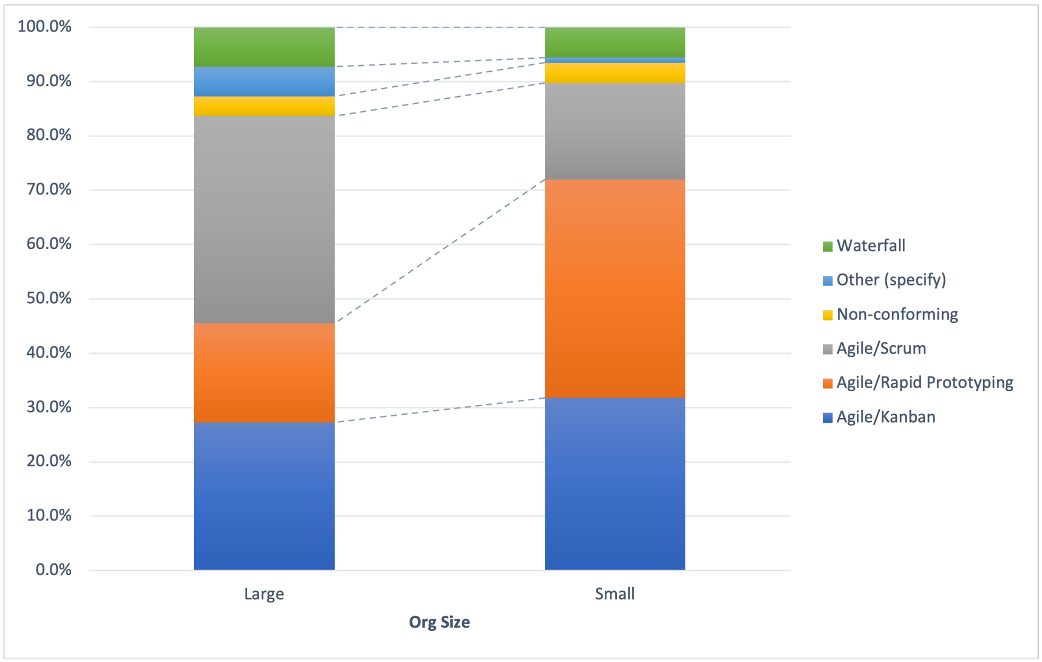

Figure 4.1: Use of Software Development Processes by Large and Small Software Organizations

Table 4.1: Logistic Regression Model Correlating Independent Variable Organization Size to Dependent Variable Scrum Use

	Coefficient B	Standard error	z	p	Odds Ratio	95% conf. interval
Large Organization	1.05	0.38	2.8	.005	2.86	1.34 - 5.97
Constant	-1.53	0.25	6.06	<.001		

Table 4.2: Logistic Regression Model Correlating Independent Variable Organization Size to Dependent Variable Rapid Prototyping Use

	Coefficient B	Standard error	z	p	Odds Ratio	95% conf. interval
Large Organization	-1.11	0.4	2.76	.006	.33	0.15 - 0.73
Constant	-0.4	0.2	2.02	<.044		

4.2 RQ2: How do large and small software development organizations differ in their development tooling?

Participants reported that tooling at small software organizations was more likely to be purchased or open source without external support. Larger organizations were more likely to use open source that was supported by an external party or to build their own tooling. P10 explained that his small startup wanted to "*quickly scale before we can afford to build some of it out ourselves.*" P6 expanded how many managed open source project makes its money: "*Customers saying well I don't want to run this myself it's too painful. But when the company gets big enough you start seeing companies that decide that it's worth it, they save enough money to have their own in house team to run bare metal infrastructure or their own you know get hosted git repositories instead of using github or you know develop their own messaging stuff instead of using slack.*" P9 offered a different perspective to why larger organizations may invest more in their own tooling. "*What I see large companies* *doing is looking for ways to leverage investment in infrastructure [...] They do that to try to get leverage because that's something that small companies can't afford to invest in, and you can't buy it off the shelf right, so it gives [them] a competitive advantage to have an investment in that kind of development."*

Amongst survey respondents, there was no significant correlation between organization size and the tooling used [Fig 4.2]. Both large and small organizations used self supported open source most frequently, with commercially supported open source and creating their own following closely behind. Commercial off the shelf tooling was used least frequently.

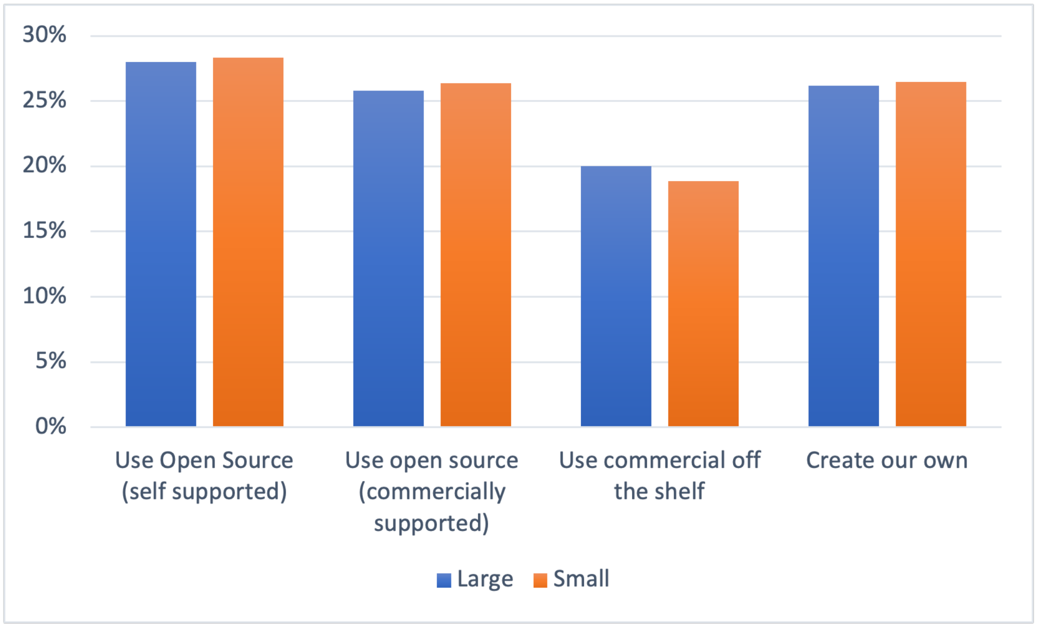

Figure 4.2: Survey Descriptive Results on Software Development Tooling

4.3 RQ3: How do large and small software development organizations differ in their work with legacy software?

Participants were split on whether large and small organizations differed in their bias to refactor legacy software. P4 reported an example of a small family-run organization where the founder had written core legacy code and the organization was very hesitant to update it. In contrast, when a large organization realized they needed to do a massive rewrite to make their website mobile-responsive, the organization recognized the need and embarked on the challenge. However, P7 described how a large organization continued to use Microsoft's Internet Explorer well past its last date of support, and the large organization mandated that support for the specific version of Internet Explorer must continue.

There was general agreement that a substantial barrier to updating legacy code was that no one knew how it worked or what it did. P9 reported that "*The reason we're not rewriting this is because we don't understand it, because it probably wouldn't be that hard to rewrite if we did understand it. But instead it's a big tangled mess of you know, things that are piled up over the years.*" P7 agreed, reporting "*One time it happened to be a cgi script that was screwing things up and we never got to fix that problem you said well how do we replace this we don't know what other things, the cgi script is holding up because there's no documentation for it.*" P8 reported a situation where legacy software at a large company was used in order to employ more modern tools. In a situation where Java 8 was required in an otherwise Java 15 environment, they used containerized development to contain the legacy code.

Survey respondents were ask to report whether or not they had worked with legacy code, how likely they were to refactor it, how well they understood it, and how well their tools supported their work with legacy code. There was no correlation between organization size and the way organizations worked with legacy code (p=.159). However, organizations that rated their legacy tools higher than the 50th percentile of the survey and organizations that rated their understanding of the legacy software higher than the 50th percentile of the survey were 6.69 times and 4.97 times more likely to refactor their legacy code respectively.

Table 4.3: Logistic Regression Model Correlating Independent Variables Legacy Tools and Legacy Understanding to Dependent Variable Probability of Legacy Refactoring

	Coefficient B	Standard error	z	p	Odds Ratio	95% conf. interval
Legacy Tools	1.9	0.6	3.18	.001	6.69	2.08 - 21.56
Legacy Understanding	1.6	0.62	2.57	.01	4.97	1.46 - 16.86
Constant	-1	0.65	1.54	.123		

4.4 RQ4: How do large and small software development organizations differ in their documentation practices?

A common theme among participants was that they felt that larger organizations had better documentation practices than smaller organizations. P1, P2, P4, P7, P8, P10, and P11 all associated larger organizations with maintaining "better" documentation. P2 noted that one large organization provided numerous templates to help encourage documentation. P8 and P10 reported that the specialized work-forces of larger organizations included roles for specialists in writing documentation. P6 agreed that large organizations had better public facing documentation, but felt that all organizations regardless of size suffered from poor internal documentation. P2 and P4 felt that small organization documentation was more "at the whims" of the developer and such documentation was geared towards business continuity. When asked about whether they saw a difference in whether UML was used by large or small organizations, multiple respondents said that they had not seen UML used at all in any of the organizations they had worked at - large or small (P2,P3,P6) - with one saying that they had only seen it at a large organization (P7).

A minority of participants believed that the application domain of the organization influenced its documentation practices more than its size. P3 and P9 both noted that federal contracting/sub-contracting can require extensive documentation, with P9 also suggesting that safety-critical software required more extensive documentation. P1 provided an example from the space industry "*when you're developing software that runs on a satellite about 222,000 km from earth that rigor that you put on that and the documentation you put on that is very different [...] you essentially have a network switch in the sky that's not an easy chore to figure things out and how you make sure you can [...] do a reset on this baby and it was quite a bit of documentation - it was a massive amount of documentation [...] any requirement was numbered and every number of the requirement was tested.*"

While interview participants discussed documentation in general terms, survey respondents were asked to evaluate the strength of three specific documentation types: requirements, technical, and support documentation. Requirements documentation is considered user stories, formal specification documents, or other documents that contain user requirements. Technical documentation contains documentation that helps the implementation team develop the system. These can be Entity Relationship Diagrams, Message Flow diagrams, technical architecture diagrams, code contract documentation, and similar. Support documentation is documentation needed for support personnel to identify and remedy issues that may occur at runtime. This may include a technical user manual, an installation guide, or similar.

While no correlation was found between organization size and requirements documentation, there was discovered correlation between public ownership, what kind of software the organization developed, and whether the organization practiced good requirements documentation. A logistic regression found that the model as a whole is significant (Chi^2(7) = 17.47, p .015, n = 162).

Table 4.4: Logistic Regression Model Correlating Independent Variables Public Ownership and Type of Software to Dependent Variable Strong Requirements Documentation

	Coefficient B	Standard error	z	p	Odds Ratio	95% conf. interval
Public Owned	0.95	0.38	2.51	.012	2.58	1.23 - 5.43
Type of Software(Cloud Native)	2.06	1.19	1.72	.085	7.84	0.76 - 81.32
Type of Software(User Installable)	2.49	1.19	2.1	.036	12.06	1.18 - 123.53
Type of Software(OSS)	2.95	1.2	2.45	.014	19.08	1.8 - 201.94
Type of Software(Embedded Software)	2.96	1.26	2.35	.019	19.22	1.63 - 227.15
Type of Software(Library or Framework)	2.24	1.34	1.67	.094	9.37	0.68 - 128.62
Constant	-2.02	1.17	1.73	.084		

This model shows that while organization size and organizations whose application domain is cloud native, user installable, or library/framework software may not be predictive of strong requirements documentation, organizations that are publicly owned and whose application domain is either open source software or embedded software had better requirements documentation.

There was no correlation discovered between organization size or any other variables and technical documentation. When it came to support documentation, a correlation between organization size, public ownership, and whether the organization practiced good support documentation was found. Logistic regression analysis shows that the model as a whole is significant (Chi^2(2) = 8.62, p .013, n = 162). This indicates that small organizations that are publicly owned have the greatest likelihood of creating good support documentation according to our survey respondents. Note that an organization that was publicly owned had a 3.44x greater opportunity of having stronger support documentation. Organization size meanwhile only had a .28x greater opportunity, indicating public ownership had a much stronger effect in the model.

Table 4.5: Logistic Regression Model Correlating Independent Variables Organization Size and Public Ownership to Dependent Variable Strong Support Documentation

	Coefficient B	Standard error	z	p	Odds Ratio	95% conf. interval
Large organization	-1.26	0.54	2.35	.019	0.28	0.1 - 0.81
Public Owned	1.24	0.55	2.26	.024	3.44	1.18 - 10.05
Constant	1.96	0.37	5.23	<.001		

4.5 RQ5: How do large and small software development organizations differ in their testing practices?

Several participants reported that large organizations often had dedicated testers while smaller organizations relied on developers for testing. P4 noted "*in large organizations and the mid sized organization we had dedicated QA people for testing. Small organizations [sic] had eight developers [...] and again they weren't doing very good testing anyway, but when they were doing their own testing like they were doing their testing themselves. But in large organizations there are QA teams and they're [sic] embedded QA so it's [a cross-functional] approach. And most of these experiences, the developers writing the unit tests and sometimes you have [sic] people in the QA team that are handling [sic] selenium tests [...] or integration or end to end tests.*" P7 described dedicated QA personnel with separate QA teams rather than cross-functional teams. P8 reported dedicated QA personnel who were empowered to block releases, whereas in small organizations it was not unusual to push software into production either without running tests or where tests were failing.

P9 and P10 both reported that small and large organizations were equally likely to employ unit testing by developers. P3 went so far as to suggest that they did not see any difference at all in large and small organizations.

Survey responses revealed a strong correlation between organization size and the use of independent testers. More specifically, large organizations with specialized work-forces and formal test plans had a high correlation of utilizing independent testers. The logistic regression model is significant (Chi^2(3) = 40.89, $p < .001$, $n = 162$).

Table 4.6: Logistic Regression Model Correlating Independent Variables Organization Size, Specialized Workforce, and Formal Test Plans to Dependent Variable Use of Independent Testers

	Coefficient B	Standard error	z	p	Odds Ratio	95% conf. interval
Large Org	1.66	0.51	3.26	.001	5.25	1.94 - 14.23
Specialized Workforce	1.14	0.5	2.27	.023	3.12	1.17 - 8.34
Formal Test Plans	2.17	0.53	4.14	<.001	8.79	3.14 - 24.59
Constant	-3.97	0.6	6.62	<.001		

4.6 RQ6: How do large and small software development organizations differ in their code and development practice quality metrics?

Multiple participants noted that they saw large companies focused more on quality. P10 notes the "*smaller company is more much more of the like let's ask for forgiveness attitude, you know and a larger company never wants to get to that.*" In small organizations, P6 reported that "*your business objective is to prove value right which is not done through reliability typically right it's also not done through - depends on the company, sometimes through usability right, what is your differentiator is it that you make something that's so much easier to use than everyone else.*"

With regards to static source analysis, multiple participants agreed that if it was used, it was used by large organizations. P2, P4, P6, P9, and P10 all noted how their experience at large organizations included the use of static analysis, with P1, P4, P6, and P10 all noting that the smaller organizations did not use static analysis tools.

The survey responses revealed no correlation between organization size and the use of static analysis tools. However, there was a correlation between both test documentation formality and requirements documentation formality and the use of static analysis. Logistic regression analysis shows that the model as a whole is significant ($\text{Chi}^2(3) = 30.63$, $p < .001$, $n = 162$).

Table 4.7: Logistic Regression Model Correlating Independent Variables Documentation Formality to Dependent Variable Static Analysis

	Coefficient B	Standard error	z	p	Odds Ratio	95% conf. interval
Formal Test Plan Use	1.57	0.6	2.63	.009	4.83	1.49 - 15.61
Formal Specification Documents	2.88	0.74	3.91	<.001	17.87	4.21 - 75.82
User Stories via JIRA	1.14	0.66	1.72	.086	3.13	0.85 - 11.49
Constant	-0.63	0.62	1.03	.305		

There was a correlation between organization size, worker specialization, formalized test plans, and requirement document formality on whether the organization achieves the optimal 0-15% change failure rate. Logistic regression analysis shows that the model as a whole is significant ($\text{Chi}^2(5) = 43.26$, $p <.001$, $n = 162$).

Table 4.8: Logistic Regression Model Correlating Independent Variables Organization Size, Worker Specialization, and Test and Requirements Formality to Dependent Variable Change Failure Rate

	Coefficient B	Standard error	z	p	Odds Ratio	95% conf. interval
Large organizations	1.66	0.51	3.26	.001	5.25	1.94 - 14.23
Specialized Workforce	1.14	0.5	2.27	.023	3.12	1.17 - 8.34
Formal Test Plans	0.91	0.39	2.34	.019	2.48	1.16 - 5.29
Formal Specification Documents	-2.58	0.86	3	.003	0.08	0.01 - 0.41
User Stories via JIRA	-2.14	0.87	2.47	.014	0.12	0.02 - 0.64
Constant	-3.97	0.6	6.62	<.001		

4.7 Other Findings

A common theme among participants was that they observed greater rigor and structure in larger organizations than smaller organizations. P1 first characterized the difference between large and small organizations as *"rigor and the structure, I think, is probably what comes to my mind immediately."* P2 reported *"I feel like it's a smaller company it was a lot less structured; a lot more like just[...] fly by the seat of your pants sort of thing"*.

This rigor was often explained by a sense that larger organizations were more risk adverse. P7 noted *"this doesn't just speak to software engineering departments, but companies in general tend to accept less risk the bigger that they get,"* with P10 expressing similar sentiments. P4 suggested one reason *"this starts happening is because there are a lot of lawsuits that started getting flung around;"* indicating that larger organizations are a larger target for such lawsuits. A more common explanation, however, was that the risk aversion was due to the increased scale of the detrimental effects poor quality could have on a large organization's reputation or brand (P1, P2, P4, P6). It was also noted by several participants that risk adversity may be explained by other factors beyond size, such as practicing in a business domain of government contracting (P7,P11), or if they are publicly traded (P4,P10). Government contracts require substantial risk mitigation in them, and the US SEC requires disclosure of risk to shareholders. This large organization rigor and risk aversion is contrasted with small organization drive to prove value. P10 felt that smaller organizations have *"much more of the like let's ask for forgiveness attitude."*

Two participants reported that smaller organizations generally had a more generalized workforce, whereas larger organizations were more specialized. P2 explained "*you kind of got to like wear more hats at the smaller company. So, [sic] in addition to just developing software I also did user training and reaching out to users and see what they want, [sic] troubleshooting with them stuff like that.*" P4 also specifically noted the "multiple hats" worn in smaller organizations.

Chapter 5: Discussion

This thesis has examined whether small software development organizations should model their practices on larger software development organizations. We sought to answer this by first determining how practices differ in small and large organizations.

I found that four out of six areas of software development practice I examined differed significantly with organization size, where small organizations were not trying to follow the practices of larger software development organizations. This is despite some large organization practices getting objectively "better" results. Large organizations that employ specialists and formal test plans had the lowest change failure rate of 0-15%, indicating that large organizations are better at reducing defects in software. What is not considered is the monetary or human capital cost of achieving this. However, this was achieved without compromising other dimensions of quality, as there were no correlations were observed between size and other quality metrics. However, this may be economically out of reach for smaller organizations due to their limited resources[7].

The finding that large organizations have more frequent defect-free software releases may be related to their general intolerance for risk. Participants, as well as previous research[7], reported that smaller organizations do not necessarily have the same risk intolerance, especially when the small software organization is a startup. Several previous studies[11][10] have found that startups must constantly ensure they fit a market need, and that often leads to focusing on shipping software out the door rather than ensuring the software is defect-free[7].

I believe a better question organizations of all sizes can ask themselves is, "what software development practices should I use that best fit my organization's needs?" The applicability of this question is aptly shown in the difference identified in the large and small software development organization's development methodologies. The fact that small development organizations utilized Rapid Prototyping more often may be due to the fact that there is a significant overlap between small software development organizations and startups. In a startup environment, use of Rapid Prototyping makes sense to prove value quickly.

Our findings with regards to correlations between large organization size, worker specialization, formalized test plans, and the use of independent testers is consistent with previous research on software testing[20].

Some of my findings suggest the need to modernize curriculum to adequately prepare software engineers for modern software development practices. While the use of UML has long been a major focus of courses in Requirements Analysis and Specification, only one participant reported encountering the use of UML in industry which echos previous research[23]. Software Project Management courses traditionally walk students through how a large project undertaken by a large company might be organized by using concepts such as earned value management. While project management was not specifically studied in this research, earned value management is a practice requiring a high degree of rigor, and interview and survey results pointed to larger organizations practicing higher degrees of rigor than smaller organizations. Will students who take such courses be prepared for a smaller software development organization's less structured method of software development? Are university courses preparing students for providing specialized software skills to a larger organization or to become a jack of all trades in a smaller organization?

Chapter 6: Conclusion

I found that small software organizations as a whole are not attempting to emulate the practices of large software organizations. Instead, small software organizations appear to approach software development with practices tailored to their needs. Small organizations who attempt to mimic large organization processes purely because they believe these organizations are doing it the "right" way should be aware that such attempts would not match their similarly sized and resourced peers. Based on the reality of modern software development, further consideration of how software engineering curriculum prepare students for the workforce is warranted to ensure that students are prepared to work at both large and small software development organizations.

Appendix A: Interview Questions

1. What do you think are the biggest differences between the way big and small development organizations approach software development?
2. Why do you think large and small software development organizations differ? Do they care more (or less) about reliability? Time to-market? Cost? Regulatory requirements? Developer expectations?
3. What do you think could be other contributing factors to the way a company approaches software development other than organization size?
4. Did you ever see small development organizations trying to adopt tools or processes taken from larger software organizations? (If participant doesn't know where to start, then suggest the following examples: microservices, containers, container orchestration)
5. In the next few questions, I'll go through a couple of software development practices. And, for each, how do you feel they may differ between large and small development organizations?
   - (a) Documentation and UML
     - i. Are there differences in documentation formality and UML specification adherence between large and small orgs?
   - (b) Testing
     - i. Is there a difference in when software testing occurs between different sized orgs, and by whom?
   - (c) Static Source Code Analysis
   - (d) Open Source Software Dependencies (Were there restrictions on OSS use in larger orgs?)

(e) Software Development Processes

i. What kind of software project processes were used and where? Agile/Scrum? Agile/Kanban? Rapid Prototyping?

(f) DevOps (Was Dev(sec)Ops and CI/CD pipelines used in one vs the other?)

6. What domain characteristics do you think play into differences in the way companies approach software development? Scale (data) Performance Metrics? Web vs Embedded?

7. In what ways does legacy code or software practices prevent the adoption of more modern tools and processes?

Appendix B: Survey Questions

All questions required the selection of ONE option unless otherwise noted.

1. Approximately how many software development professionals are in the company you are responding for? A software development professional for the purposes of this survey is anyone whose primary responsibilities contribute to the creation or maintenance of software.

   (a) 26-50

   (b) 51-100

   (c) 101-500

   (d) 501-1000

   (e) >1000

   (f) I do not know

2. How large is the software development team that you work with for your project/product?

   (a) 1

   (b) 2-5

   (c) 6-10

   (d) 11-20

   (e) 21-30

   (f) >30

   (g) I do not know

3. What is the primary business domain of your company?

   (a) Software creation for sale by end users, either as a service or retail

(b) Technology/Software consulting/contracting for a government entity

(c) Technology/Software consulting/contracting for a non-governmental entity

(d) Non-software related products

(e) Other (specify)

4. My company is publicly owned (stock is traded on a public exchange) True/False

5. What is your role in your organization?

   (a) Software engineering

   (b) IT

   (c) Business analyst / product management

   (d) QA / software testing

   (e) Management

   (f) Other (specify)

6. How many years have you worked for your organization?

   (a) $<$ 1 year

   (b) 1-3 years

   (c) 3-5 years

   (d) 5-10 years

   (e) $>$ 10 years

7. My organization has a:

   (a) Chief Technology Officer (CTO)

   (b) Chief Information Officer (CIO)

   (c) Both a CTO and a CIO

(d) Neither a CTO nor a CIO

8. The primary software product/deliverable that I contribute to is best classified as:

    (a) Cloud native

    (b) User installable software

    (c) Open Source Software

    (d) Embedded Software

    (e) Library or Framework meant to be used within a larger application

    (f) Other (specify)

9. How many business stakeholders (individuals or groups) does your organization have that are influential enough to drive software development priorities?

10. Which of the following roles do you have represented on your team? (select all that apply)

    (a) Full Stack Software Engineer/Developer

    (b) Scrummaster or Project Manager

    (c) QA/Test

    (d) Business Analyst

    (e) Product Management/Product Owner

    (f) UI/UX

    (g) Database or Data specialist

    (h) Specialized Front End/Back End Developer

11. For the primary application or service you work on, how often does your organization deploy code to production or release it to end users?

    (a) Between once per month and once every 6 months

   (b) Between once per week and once per month
   (c) On-demand (multiple deploys per day)

12. For the primary application or service you work on, what is your lead time for changes (i.e., how long does it take to go from code committed to code successfully running in production)?
    (a) Between 1-6 months
    (b) Between 1 week to 1 month
    (c) Between 1 day to 1 week

13. For the primary application or service you work on, how long does it generally take to restore service when a service incident or a defect that impacts users occurs (e.g., unplanned outage or service impairment)?
    (a) Between 1 week and 1 month
    (b) Between 1 day and 1 week
    (c) Less than 1 day

14. For the primary application or service you work on, what percentage of changes to production or released to users result in degraded service (e.g., lead to service impairment or service outage) and subsequently require remediation (e.g., require a hotfix, rollback, fix forward, patch)?
    (a) 46-60%
    (b) 16-30%
    (c) 0-15%

15. In order to address the lack of a specific tool or platform, my organization will do the following to address it (rank according to how likely your organization would be to use the given strategy).

    (a) Create its own tools/platform
    (b) Use open source tools/platforms
    (c) Use open source tools/platforms supported by another organization
    (d) Use commercial closed source tools/platforms supported by another organization

16. My organization must work and integrate with software written over 10 years ago that has not been recently refactored (considered "Legacy") (Yes/No)

17. How do you rate the tools available to work with such legacy software?
    (a) Excellent
    (b) Good
    (c) Ok
    (d) Not good
    (e) Poor
    (f) N/A I do not work with legacy software

18. How good is your software organization's understanding of the legacy software?
    (a) Excellent - the documentation is up to date and relevant and/or there is developer expertise readily available
    (b) Good - the documentation is mostly relevant with some missing pieces. There is developer expertise occasionally available
    (c) OK - the documentation is hit or miss. Very little developer expertise available.
    (d) Not good - Very little documentation. Developer knowledge of system is limited.
    (e) Poor - No relevant documentation available. No developer knowledge is available.
    (f) N/A: I do not work with legacy software

19. Have you ever seen your organization put serious effort into modernizing or refactoring legacy software?

(a) Always

(b) Often

(c) Sometimes

(d) Rarely

(e) Never

(f) N/A I do not work with legacy software

20. In relation to your work duties, how often do you perform additional tasks outside of your core scope. Examples may include: - Software engineer performing system administration duties - Software engineer performance tuning a database - Business analyst performing significant QA tasks

    (a) Daily

    (b) Weekly

    (c) Monthly

    (d) Rarely

    (e) Never

21. Who generally makes changes to your production systems?

    (a) Software development team, manually

    (b) Software development team, via automated CD pipelines

    (c) IT or other team

    (d) Don't know

The following 7 questions were 7-point Likert scale questions from Strongly Disagree to Strongly Agree

22. My organization develops and executes against formal, documented test plans

23. My organization utilizes independent software testers that are not developers of the product.

24. My organization utilizes automated unit testing including benchmarks for code coverage

25. My organization utilizes static source analysis to improve code quality

26. My organization maintains up to date requirements documentation that accurately reflects the functional and/or technical requirements of the software

27. My organization maintains up to date technical documentation for developers that accurately reflects the software architecture, message flows, or external system dependencies, to assist developers in understanding the higher level system design.

28. My organization maintains up to date support and troubleshooting documentation that helps support personnel perform advanced functions to support the product.

29. How does your team receive software requirements?

    (a) Formal specification documents

    (b) User Stories via a product like JIRA

    (c) Other (specify)

30. What software development process does your organization follow (choose best answer)

    (a) Agile/Scrum

    (b) Agile/Kanban

    (c) Agile/Rapid Prototyping

    (d) Waterfall

    (e) Non-conforming

    (f) Other (specify)

31. How closely does your organization follow the process specified in the previous question?

    (a) Extremely Loosely

    (b) Mostly loosely

    (c) Sometimes loosely

    (d) Neither closely or loosely

    (e) Sometimes closely

    (f) Mostly closely

    (g) Extremely closely

32. OPTIONAL: Enter any additional comments or feedback you would like to provide to the researchers

33. Would you like to enter the raffle or receive the survey results? (checkboxes, then an opportunity to enter email address)

Bibliography

[1] "Google developers." `https://medium.com/google-developers`.

[2] "Netflix techblog." `https://netflixtechblog.com/`.

[3] S. Hanselman, "Scott hanselman." `https://www.hanselman.com/`.

[4] S. Hanselman, "Dark matter developers: The unseen 99%." `https://www.hanselman.com/blog/dark-matter-developers-the-unseen-99`, Mar 2012.

[5] J. Yang, "Jean yang status update 11/2/21." `https://twitter.com/jeanqasaur/status/1455589141299675139`, Nov 2021.

[6] J. Yang, "Building for the 99% developers." `https://future.com/software-development-building-for-99-developers/`, Jun 2022.

[7] C. Giardino, M. Unterkalmsteiner, N. Paternoster, T. Gorschek, and P. Abrahamsson, "What do we know about software development in startups?," *IEEE Software*, vol. 31, no. 5, pp. 28–32, 2014.

[8] M. E. Fayad, M. Laitinen, and R. P. Ward, "Thinking objectively: software engineering in the small," *Communications of the ACM*, vol. 43, no. 3, pp. 115–118, 2000.

[9] J. Brodman and D. Johnson, "What small businesses and small organizations say about the cmm," in *Proceedings of 16th International Conference on Software Engineering*, pp. 331–340, 1994.

[10] S. G. S. G. Blank, *The four steps to the epiphany : successful strategies for products that win.* Hoboken, New Jersey: Wiley, fifth edition. ed., 2020.

[11] C. Giardino, X. Wang, and P. Abrahamsson, "Why early-stage software startups fail: A behavioral framework," pp. 27–41, 06 2014.

[12] E. W. Tegegne, P. Seppänen, and M. O. Ahmad, "Software development methodologies and practices in start-ups," *IET Software*, vol. 13, no. 6, pp. 497–509, 2019.

[13] G. C. Lapasini Leal, R. Prikladnicki, C. Ebert, R. Balancieri, and L. Pompermaier, "Practices and tools for software start-ups," *IEEE Software*, vol. 37, pp. 72–77, 01 2020.

[14] T. Dingsøyr, N. B. Moe, T. E. Fægri, and E. A. Seim, "Exploring software development at the very large-scale: A revelatory case study and research agenda for agile method adaptation - empirical software engineering." `https://link.springer.com/article/10.1007/s10664-017-9524-2`, Jun 2017.

[15] R. Knaster, "About." `https://scaledagileframework.com/about/`, Mar 2023.

[16] L. Boedecker, "Small business need generalists.," in *Business Education Forum*, vol. 28, pp. 28–9, ERIC, 1974.

[17] D. DeBellis and C. Peters, "Dora 2022 accelerate state of devops report now out | google cloud blog." `https://cloud.google.com/blog/products/devops-sre/dora-2022-accelerate-state-of-devops-report-now-out`, Sep 2022.

[18] C. Sadowski, E. Aftandilian, A. Eagle, L. Miller-Cushon, and C. Jaspan, "Lessons from building static analysis tools at google," *Communications of the ACM*, vol. 61, no. 4, pp. 58–66, 2018.

[19] M. Gleirscher, D. Golubitskiy, M. Irlbeck, and S. Wagner, "On the benefit of automated static analysis for small and medium-sized software enterprises," in *International Conference on Software Quality*, pp. 14–38, Springer, 2012.

[20] S. Gruner and J. Van Zyl, "Software testing in small it companies: a (not only) south african problem," *South African Computer Journal*, vol. 47, no. 1, pp. 7–32, 2011.

[21] M. Kassab, J. F. DeFranco, and P. A. Laplante, "Software testing: The state of the practice," *IEEE Software*, vol. 34, no. 5, pp. 46–52, 2017.

[22] C. J. Stettina and W. Heijstek, "Necessary and neglected? an empirical study of internal documentation in agile software development teams," in *Proceedings of the 29th ACM international conference on Design of communication*, pp. 159–166, 2011.

[23] M. Petre, "Uml in practice," in *2013 35th International Conference on Software Engineering (ICSE)*, pp. 722–731, 2013.

[24] M. Niazi, S. Mahmood, M. Alshayeb, and A. Hroub, "Empirical investigation of the challenges of the existing tools used in global software development projects," *IET Software*, vol. 9, no. 5, pp. 135–143, 2015.

[25] *Small Business Size Standards*, vol. 13 CFR 121. 1996.

Curriculum Vitae

Jeffrey Longo

3068 Hazelton St
Falls Church, VA 22044

LinkedIn: https://www.linkedin.com/in/jeffreylongo/

Education

Master of Science: Software Engineering Spring 2023 (expected)
George Mason University, Fairfax VA
Graduation GPA: 4.0

Bachelor of Science: Computer Science Fall 2005
University of Mary Washington, Fredericksburg VA

Experience

Kastle Systems Intl June 2009 - Present
Director of Software Development (2/21 - Present)
Responsibilities include directing US-based software and QA team tasks and priorities. Coordination with hardware development and offshore development teams. Technology varied from embedded (C++ on FreeRTOS/Zephyr) to "traditional" Client/Server/Relational Database, to AWS and Azure Cloud, and iOS/Android app development. Employee performance appraisal, review, coaching, and maintaining a professional yet fun environment. Led transition to DevSecOps through the transition from SVN -> Git, adoption of Atlassian Cloud, and use of CI pipeline build system with integrated dependency analysis and source linting.

Software Engineering Manager (3/15 - 2/21)
Managed US based software and QA teams. Provided management and technical guidance and contribution to Kastle Safe Spaces initiative towards healthy re-opening during the COVID pandemic. Managed from concept through launch and support of Kastle's latest hardware and software stack. Lead the creation of Kastle's FIPS 201-2 and ICAM compliant solution for the US Government to the GSA APL. Lead a team of developers in launching of Kastle's Multifamily offering, including mobile device integration with Allegion (Schlage) Engage-enabled (NDE, Control, LE) hardware. Lead a team of developers to implement an AWS Cloud app for quick human categorization of video analytics-based alarms. The

solution was cloud native and utilized Amazon SQS, S3, IAM, and EC2 with ELB and Autoscaling.

Senior Software Engineer (6/09 - 3/15)
Numerous contributions to Kastle software. Technologies utilized include TCP/IP and UDP socket networking, C++ on Linux, C# .Net, PHP, Java and JNI, contribution to RESTful standards.

CSC June 2003 - May 2009
Technical Lead (1/07 - 5/09)
Instrumental in leading the DCL transition from ISO 18000-6b UHF RFID tags to EPC Gen2 (ISO 18000-6c) RFID tags. Trialed several different readers including Intermec IF5, Symbol XR400, and ThingMagic M5, and wrote software that would interface with all 3 readers to give the client choice in vendor. Directed the development efforts of 2 other team members and integrated other teams working on disparate architectures (Mainframe CICS, IBM WebSphere Application Server).

Software Engineer (8/05 - 1/07)
Developed software (Visual C++, MFC) for the US Customs and Border Protection's (CBP) SENTRI and NEXUS Dedicated Commuter Lane (DCL) program. The software I worked on consisted of client-side software including a touchscreen program for CBP Officer use and a program that interfaced with the various hardware in the lane.

Software Engineering Intern (6/03 - 8/05)

IBM June 2001 - August 2001
Software Development Co-Op (6/01 - 8/01)