“Uncovering Emerging Information Trends in Information Technology: An Industry Survey”

Eric M. Pastore

Submitted December 4, 2015

Submitted as a Capstone Project Report in Partial Fulfillment of a Master of Science Degree in Professional Studies at the Rochester Institute of Technology

Contents

Abstract

Given the importance of information technology and software in every area of modern life, this Capstone Project identified previously unknown issues facing professionals in the software industry. These unknown issues were discovered through in-person and remote interviews with 20 software engineers (see Appendix B: Interview Schedule). Software engineers were each asked an identical set of questions (see Appendix A: Interview Questionnaire) and their responses were recorded and analyzed.

The primary issue identified was that developers lacked training in coding best practices, and frequently used shortcuts and hacks to complete projects (18% of all identified issues). This lack of best practices is a major problem because it makes code difficult to update and maintain, causing schedule and budget overruns in the long term.

The chosen solution was to develop a professional certification focused on coding best practices. Developing a certification will create a common standard which industry can use to validate developer skills in critical quality areas. It will also drive improvements in developer education and training which will lead to improved job satisfaction and software quality.

Problem Background

Of the issues facing the software industry, failure to follow best practices is the most widespread and foundational. This trend results in the primary issue discovered in the Analysis phase, which is Technical Debt (10% of identified issues); a term used by several engineers to describe the hacks and shortcuts developers use to complete code projects on time and under budget. Technical Debt is caused by a lack of best practices, and results in code that works but is difficult to understand, maintain, and update. Technical Debt leads to problems on many projects because the code is difficult to understand by the developer or is simply not built well enough to allow for future development. Time is wasted fixing problems and relearning solutions rather than building value-added features.

Many other issues identified were related to a lack of knowledge about best practices. A secondary issue (8% of identified issues) is that developers are not trained in design and best practices, which results in the type of shortcuts identified in the primary issue (Technical Debt). Other identified issues suggest that developers lack training and knowledge in several related areas, such as an inability (or unwillingness) to keep up with new methods (5%) and create documentation (4%).

In spite of the lack of adherence to best practices, many standards defining best practices do exist and are consistent across organizations and technologies. Table 1 lists three examples of subject areas which are consistent across standards. These areas are Application Design, Readability, and Tools/Automation.

Table 1 – Subject Areas

Subject Area	**Description**	**References**
Application Design	Application Design refers to the creation of the underlying architecture of an application or system. In the four standards referenced at right, the architecture is always created directly from project requirements, and is documented (CENELEC 2006). In MIL-STD-498, for example (US DoD 1994), the design is documented in a Software Design Description (SDD). Modularity and abstraction are also major concepts described in the ECSS and IEEE Handbooks.	• European Cooperation for Space Standardization (2013). Space Engineering Software Engineering Handbook • IEEE Computer Society (2015). Software Engineering Body of Knowledge (SWEBOK) V3. • European Committee for Electrotechnical Standardization (CENELEC) (2006). BS EN 62304:2006 Medical Device Software – Software Life-Cycle Processes. • US Department of Defense (1994). MIL-STD-498: Software Development and Documentation.
Readability	Readability describes the ability of a maintenance developer to understand software code. Both Oracle (Oracle 2015) and Microsoft (MSDN 2015) set naming conventions, use of parenthesis and curly braces, and length limits for functions and files, among other standards. Holzman also includes length limits in his Rules for Developing Safety-Critical Code (Holzman 2015).	• Microsoft Developer Network (MSDN) (2015). Coding Techniques. • Oracle Corporation (1999). Code Conventions for the Java™ Programming Language. • Holzmann, Gerard. "The Power of 10: Rules for Developing Safety-Critical Code"
Tools/Automation	Developers use various tools to ensure code quality, including static analysis and the compiler itself. Rule 2 of NASA's JPL standard (JPL 2009) and other requirements in its Handbook (NASA 2015) state that code must be compiled with all warnings on and that static analysis tools must be used. Holzmann's Rule 10 contains an identical requirement (Holzmann 2015) along with section 5.2.4 of the FDA Software Guidance (FDA 2002). Ganssle further describes which static analysis tools are available in the marketplace in his article on the Embedded Muse newsletter (Ganssle).	• National Aeronautics and Space Administration (NASA) (2015). Software Engineering Handbook. • Holzmann, Gerard. "The Power of 10: Rules for Developing Safety-Critical Code"Ganssle • Jet Propulsion Laboratory (JPL) (2009). JPL Institutional Coding Standard for the C Programming Language. • Food and Drug Administration (FDA) (2002). General Principles of Software Validation; Final Guidance for Industry and FDA Staff.

Table 1 demonstrates the harmony of various software development standards, handbooks, and other references. For example, Microsoft and NASA both publish software development standards and agree on several points (NASA 2015) (Microsoft 2015). Moreover, other experts have written about the need for standards and their application in software projects (Twine 2003).

Despite the proliferation of standards and their obvious value to the software industry, Technical Debt and development shortcuts are still widespread issues. Evidence of these problems can be found in modern blog sites such as Embedded Muse and Mindprod.com which publish articles (both serious and parodic) related to software engineering. One article by Ganssle, for example, explains the tools that exist to ensure code quality (Ganssle). Another article by Green lists, in comic fashion, all the ways to make code as unmaintainable as possible (Green 2015).

A related problem is that while standards exist, developers have few options to be certified on them. While many certifications exist for developers in particular skills, few exist that cover software development from a broader perspective. A short list of these certifications is below:

- International Software Architecture Qualification Board (iSAQB) Certified Professional for Software Architecture (CPSA) (iSAQB 2015)
- IEEE Computer Society Certified Software Development Associate (CSDA) (IEEE 2015)
- Software Engineering Institute (SEI) Personal Software Process (PSP) (SEI 2015)

These certifications are comprehensive, but do not focus exclusively on best practices. Also, training for the CPSA certification is only offered in Germany according to the iSAQB website.

Overall, much has been defined and documented in the last decade regarding software engineering standards, but these standards are still not fully implemented in the work of the average software developer. This is reflected in the responses from interviews with software engineers and in articles published by industry professionals. This is not due to confusion in standards, but rather to a lack of industry focus and developer training and skills in best practices. This gap can best be corrected with expanded training and certification options, and the creation of the Quality+ certification will help to create these options.

Project Description

The purpose of this Capstone Project was to discover emerging issues in the software industry, and design a training or documentation solution to address one particular issue selected from the key issues identified. As such, the project had the following phases, modeled after the Analysis, Design, Develop, Implement, and Evaluate (ADDIE) method of instructional design.

Analysis

The Analysis phase was designed to identify the major problems in software development. A total of 20 software engineers were interviewed (Appendix II) to determine what issues they saw in the software industry. The engineers identified 107 issues, which were then analyzed to find duplicates. Overall, 46 unique issues were analyzed using a Pie Chart, Pareto Chart, and Fishbone Diagram. These diagrams are included as Figure 1, Figure 2, and Figure 3, respectively.

Design

The purpose of the Design phase, according to the Capstone Project Proposal, was to identify the primary issue the Capstone Project will solve. Based on the results of the Analysis phase and application of the 80/20 rule (MindTools 2015), the primary issue was determined to be a lack of using best practices driven by a lack of training and experience with best practices.

Most importantly, this phase examined the data and conclusions generated by the Analysis phase to decide on a solution and a target audience. The solution is a best practices certification targeted to developers based on justifications provided in the Problem Background section.

Since the final deliverable is a training solution, the learning objectives were identified along with the intended audience. Finally, specific project milestones were set. Once this information was documented, a report was sent to each of the software engineers interviewed in the Analysis phase.

Develop

In the Develop phase, the Capstone Project deliverables were developed to define the proposed certification:

- **Training Plan**: The Training Plan included a full set of performance objectives, a complete audience analysis, and a breakdown of the subject areas covered in the certification exam.
- **Sample Exam Questions:** To illustrate the type of questions to be asked on the exam, a total of 10 sample questions were developed. These questions were a combination of varying questions types, including multiple choice, fill in the blank, lists, and simulations. These question types were used to illustrate the knowledge areas to be covered. In a true exam, however, most questions would be simulation-type questions to take advantage of developers' visual and kinesthetic learning styles.

Implement

In this phase, the project deliverables were deployed to a WordPress site for review by the Capstone Consultant and the 20 engineers interviewed. The engineers were given one week to evaluate the solution and leave their comments to be included in the final deliverable in the Evaluate phase. After one week, 4 engineers responded and left their comments, along with the Capstone Consultant. These comments were incorporated in the next revision of the Training Plan.

Evaluate

In this phase, the results of the project were analyzed and documented, and any lessons learned recorded in the final Capstone Project Report. The engineer's feedback on the project deliverables was included in the final version for submission, and the final Capstone Project Report was written summarizing all feedback in the Project Results section.

This evaluation continued an iterative process of evaluation which has occurred since the project began. After each interview, an email was sent to recap the issues identified and ensure the correct understanding of what was discussed. Later, a Design phase summary was sent to the engineers laying out the conclusions from the Analysis phase and the proposed solution from the Design phase. Finally, the project deliverable was posted for evaluation. These actions ensured that the solution was aligned with the identified issues from the beginning.

Project Results

The stated goal of this Capstone Project was to uncover previously unknown issues in software development. To accomplish this goal, this graduate student performed an independent literature review of various articles related to the software industry. This literature review was then used to develop an interview questionnaire (Appendix I) used to interview 20 software engineers working in the industry. The resulting data was then analyzed using various methods and combined with research from online articles, code standards, and other reference materials. The Project Results section will discuss what was discovered as a result of the interviews, data analysis, and supplemental research. This results section is divided into the following four areas: Data Analysis, Interviews, Reactions, and Supplemental Research.

Data Analysis

After identifying the unique issues and generating the Pie and Pareto Charts, the primary issue was identified. The primary issue to solve was Technical Debt (10% of identified issues) along with a closely related secondary issue of a lack of training in design and best practices among developers (8% of identified issues). Using the 80/20 Rule (MindTools 2015), these two issues were determined to cause most of the problem. For a complete listing of issues, see Figures 1 and 2 which display each issue and its share of the overall issues identified.

The next step was to perform a deeper analysis and attempt to determine root causes. While the graduate student attempted to brainstorm ideas based on knowledge of the subject matter, the most significant root causes were those identified by the engineers interviewed. When arranged on a Fishbone Diagram (Figure 3) the linkages between issues illustrated that most issues were the result of a lack of training, knowledge, and experience.

For this reason, the decision was made to develop a professional certification. Per Figure 3, lack of knowledge and inconsistent training impacted many of the root causes which led to a lack of best practices in software development.

Interviews

The interview questionnaire included questions which were specific to discovering issues in software development but broad enough to discover as many avenues as possible. Because of this, interviewees gave a range of answers. Most were technical, but others dealt with turnover (4%), job security (2%), sales (1%), and other non-technical issues. These responses underlined an important theme agreed upon by 19 out of 20 engineers: that human issues are always more difficult to solve than technical issues, which was one question asked in the interviews.

Another direct result of the interviews was the invalidation of the issues identified in the initial literature review (Appendix I). When asked to rank the issues by importance, most engineers left out issues and rarely ranked all issues. Even when some issues were ranked highly (1 or 2), they were rarely identified in the other interview responses, with the exception of cybersecurity which was identified as an issue by 3 engineers. Overall, the trend ranking exercise did not yield any useful insights.

Reactions

An important phase in the project was the Evaluate phase, where engineers had the chance to give feedback on the final deliverable. While the engineers were cautiously optimistic regarding a new certification, 3 engineers stated that they saw certifications as either an HR tool, too broad, out of date, or imperfectly implemented. This feedback was implemented in the training plan to ensure the certification had real value and was continuously updated to stay consistent with current practice.

The engineers were mainly concerned with 2 issues: principles vs. practice and documentation. There was some disagreement over how code comments should be written and where the comments should be placed in the code. They were also concerned with the standards in general, and wanted the certification to cover pros and cons of various techniques rather than prescribing a particular approach.

Supplemental Research

The most surprising result from the literature research was not only the number of code standards which exist, but the degree to which they agree. This is important to note because it eliminates a possible cause of issues in software development, which is confusion in software standards. Table 1 illustrates the software references used to develop the Training Plan and the areas that they cover. There are many areas of overlap.

The most important conclusion from the literature research was the amount of work which has already been done to standardize software development practice. Many organizations, along with their partners, suppliers, and customers, have thought out, documented, and published detailed standards and references describing how software development should be performed. In spite of this, code standards can still make a much larger impact on academia and the workplace.

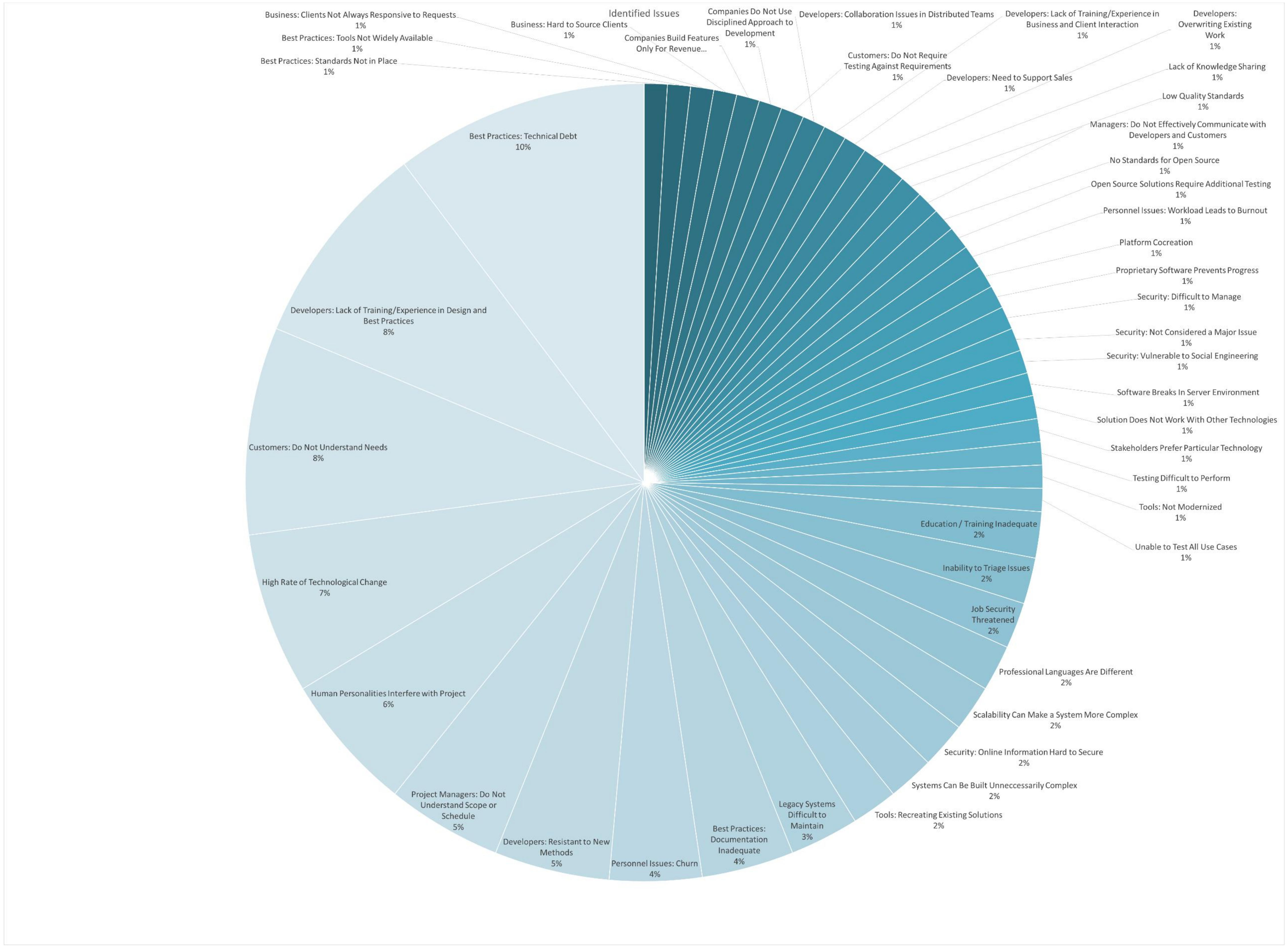


Figure 1: Identified Issues Pie Chart

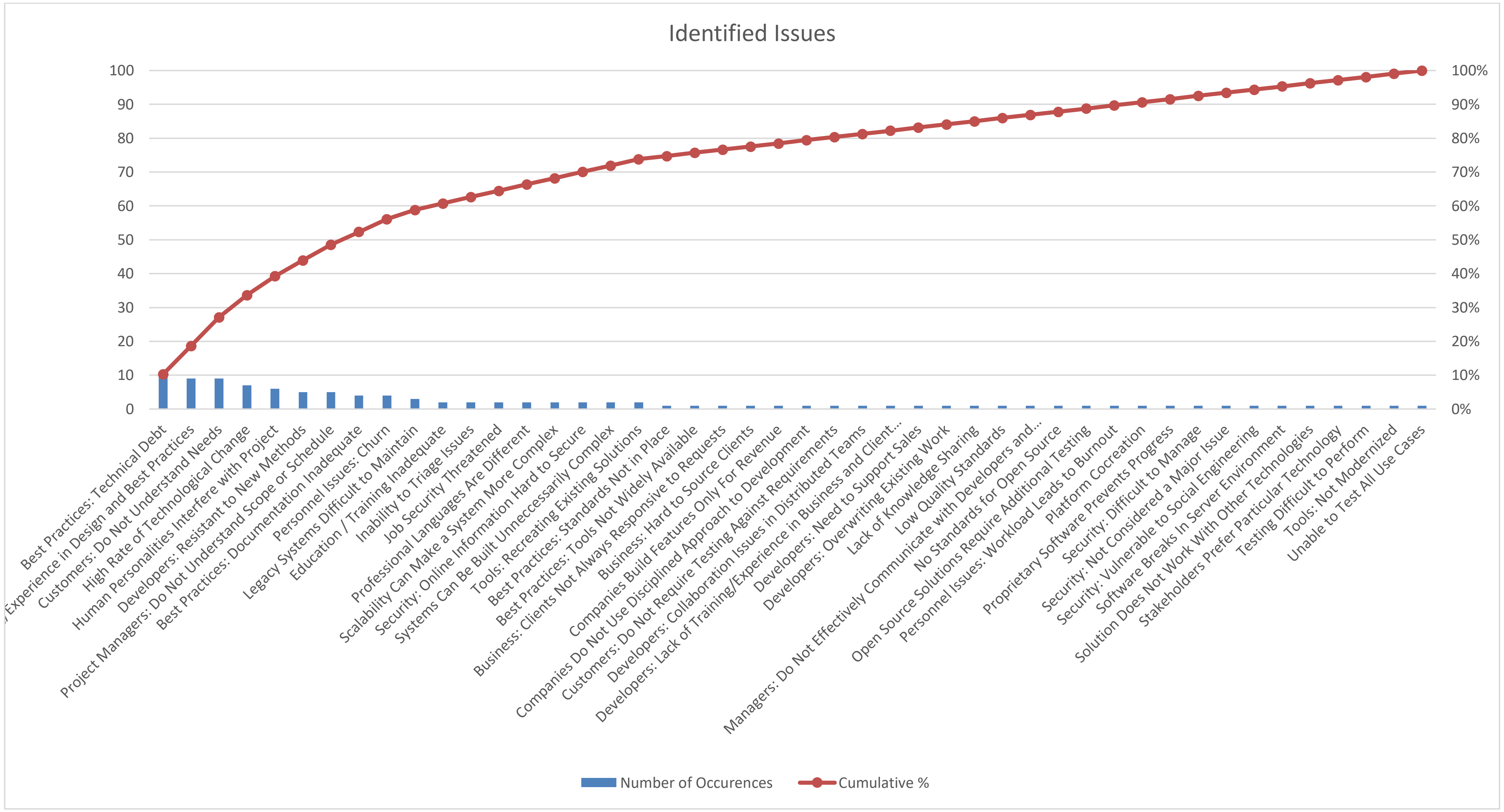
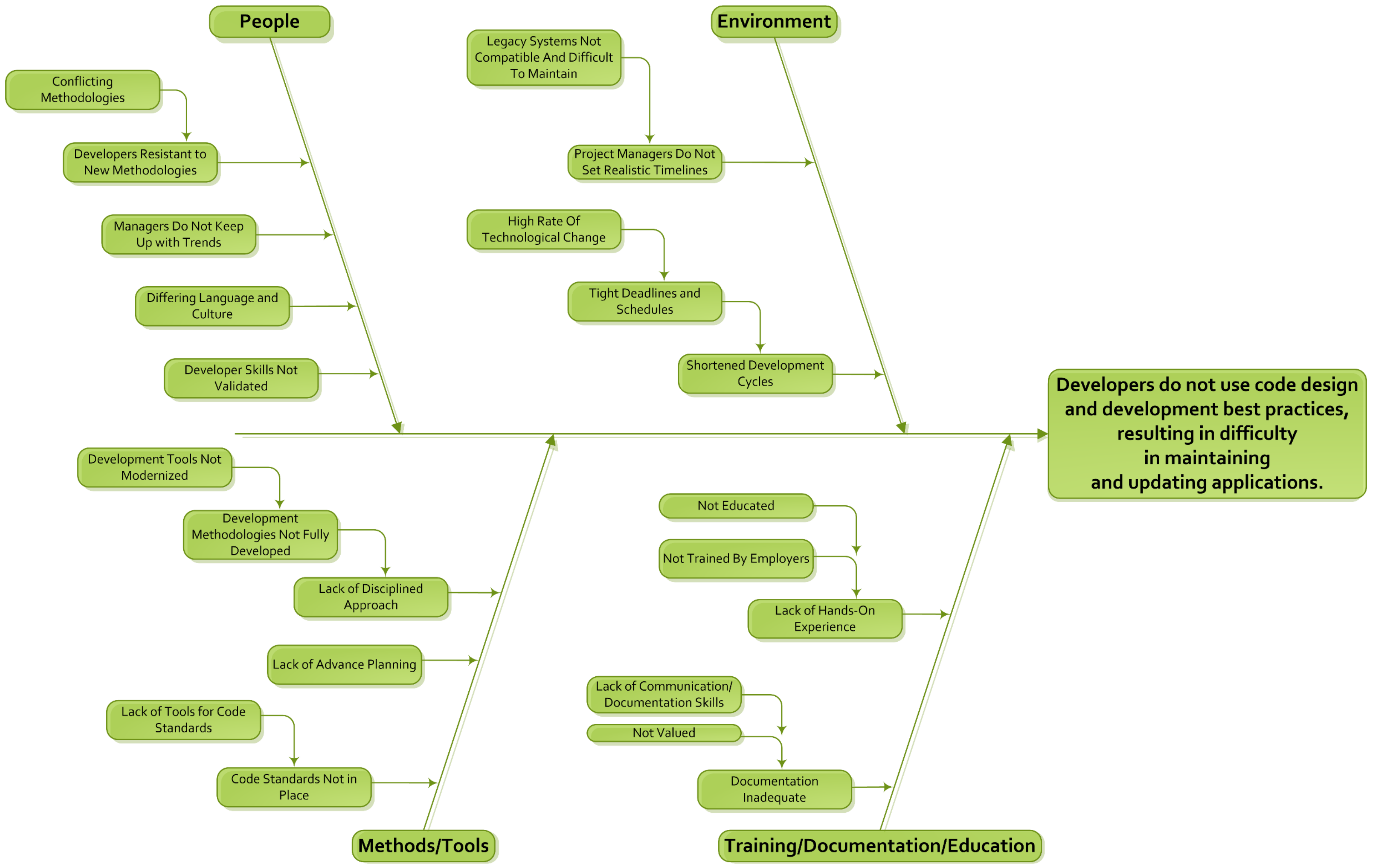


Figure 2: Identified Issues Pareto Chart



People
Environment
Conflicting Methodologies
Developers Resistant to New Methodologies
Managers Do Not Keep Up with Trends
Differing Language and Culture
Developer Skills Not Validated
Legacy Systems Not Compatible And Difficult To Maintain
Project Managers Do Not Set Realistic Timelines
High Rate Of Technological Change
Tight Deadlines and Schedules
Shortened Development Cycles
Developers do not use code design and development best practices, resulting in difficulty in maintaining and updating applications.
Development Tools Not Modernized
Development Methodologies Not Fully Developed
Lack of Disciplined Approach
Lack of Advance Planning
Lack of Tools for Code Standards
Code Standards Not in Place
Not Educated
Not Trained By Employers
Lack of Hands-On Experience
Lack of Communication/ Documentation Skills
Not Valued
Documentation Inadequate
Methods/Tools
Training/Documentation/Education

Figure 3: Identified Issues Fishbone Diagram

Conclusion and Recommendation

Problem Summary

The final project deliverable is based on a graduate Capstone Project with 5 phases: Analysis, Design, Develop, Implement, and Evaluate. The Analysis phase was an exploratory phase designed to identify the major problems in software development. A total of 20 software engineers were interviewed to determine what issues they saw in the software industry. Overall, 46 unique issues were identified and analyzed in the Analysis phase. Based on the results of this analysis, two issues to solve were identified:

- Technical Debt (10% of identified issues): Hacks and shortcuts which cause later problems in software projects.
- Lack of Training in Design and Best Practices (8% of identified issues): Developers are not trained in software development best practices.

Based on the 80/20 Rule, which states that 20% of issues cause 80% of problems on average (MindTools 2015), the above two issues were determined to have the largest impact. This was further reinforced by Figure 3, which shows how lesser issues contribute to the primary issues.

Solution Summary

The interviews, analysis, and supplemental research performed resulted in the final project deliverable, a Training Plan and Sample Exam Questions for the proposed Quality+ Certification.

Overall, much has been defined and documented in the last decade regarding software engineering standards, but these standards are still not fully implemented in the work of the average software developer. This is reflected in the responses from interviews with software engineers and in articles published by industry professionals (Green 2015). This is not due to confusion in standards, but rather to a lack of industry focus and developer training and skills in best practices.

This gap can best be corrected with expanded training and certification options, and the creation of the Quality+ certification will help to create these options. Quality+ is targeted to best practices and based on professional references such as NASA's Software Engineering Handbook (NASA 2015). If adopted by industry, the requirements of the certification exam will be adopted by academia and training organizations, resulting in more education and training opportunities for developers.

Recommendation

Many organizations have code standards, but most of these are in aerospace, embedded systems, medical device, and other industries where safety and reliability are critical. The rest of the software industry can benefit from these more rigorous standards. Industry should take note of the value delivered by standards and best practices and require that their developers understand and use these practices. Individual developers must also make a commitment to learning and using best practices in their everyday work. These are necessary steps if a certification like Quality+ is to become a reality.

To further prove the findings of this project, similar interviews should be performed with a more well-defined sample and population. While this project did make use of some statistical analysis techniques, the sample was not well-designed, thus opening up the findings to challenge. A stratified sample should be taken from multiple industries, locations, and other characteristics to create a true representative sample which can withstand scrutiny.

As the project stands, however, individuals and organizations should take notice of the value provided by having a solid foundation in best practices. Lack of understanding and application of best practices results in costly Technical Debt which slows down software projects and creates issues for developers further in the project's life cycle. In contrast, the use of best practices results in software which is readable, maintainable, well-documented, and designed for future upgrades and improvements. In short, quality practices result in quality software able to satisfy human needs.

List of References

1. Carnegie Mellon University Software Engineering Institute (SEI) (2015). Personal Software Process (PSP) Developer. (Reviewed online 11/16/15 at http://www.sei.cmu.edu/certification/opportunities/psp/index.cfm).
2. European Committee for Electrotechnical Standardization (CENELEC) (2006). BS EN 62304:2006 Medical Device Software – Software Life-Cycle Processes. (Reviewed online 11/24/15 at private standards repository).
3. European Cooperation for Space Standardization (2013). Space Engineering Software Engineering Handbook. (Reviewed online 11/24/15 at http://www.ecss.nl/).
4. Food and Drug Administration (FDA) (2002). General Principles of Software Validation; Final Guidance for Industry and FDA Staff. (Reviewed online 11/16/15 at http://www.fda.gov/RegulatoryInformation/Guidances/ucm085281.htm).
5. Ganssle, Jack. Tools for Clean Code. The Embedded Muse. (Reviewed online 11/01/15 at http://www.ganssle.com/articles/Toolsforcleancode.htm).
6. Gerard J. Holzmann, "The Power of 10: Rules for Developing Safety-Critical Code," Computer, vol. 39, no. 6, pp. 95-97, June, 2006.
7. Green, Roedy (2015). Unmaintainable Code. Canadian Mind Products. (Reviewed online 11/02/15 at http://mindprod.com/jgloss/unmain.html).
8. IEEE Computer Society (2015). Software Development Associate Engineer Certification. (Reviewed online 12/01/15 at http://www.computer.org/web/education/software-development-associate).
9. IEEE Computer Society (2015). Software Engineering Body of Knowledge (SWEBOK) V3. (Reviewed online 11/16/15 at http://www.computer.org/web/swebok/v3).
10. International Software Architecture Qualification Board (iSAQB) (2015). Certified Professional for Software Architecture (CPSA). (Reviewed online 11/15/15 at http://www.isaqb.org/certifications/?lang=en).
11. Jet Propulsion Laboratory (JPL) (2009). JPL Institutional Coding Standard for the C Programming Language. (Reviewed online 11/02/15 at http://lars-lab.jpl.nasa.gov/).
12. Microsoft Developer Network (MSDN) (2015). Coding Techniques. (Reviewed online 11/02/15 at https://msdn.microsoft.com/en-us/library/aa291593 (v=vs.71).aspx).
13. MindTools (2015). Pareto Analysis – Using the 80/20 Rule to Prioritize. (Reviewed online 12/03/15 at https://www.mindtools.com/pages/article/newTED_01.htm).
14. National Aeronautics and Space Administration (NASA) (2015). Software Engineering Handbook. (Reviewed online 11/02/15 at https://swehb.nasa.gov).
15. Oracle Corporation (1999). Code Conventions for the Java™ Programming Language. (Reviewed online 11/16/15 at http://www.oracle.com/technetwork/java/codeconvtoc-136057.html).
16. Twine, James R (2003). The importance of a coding standard and the benefits of adhering to it. JRTwine Software, LLC. (Reviewed online 11/01/15 at http://www.jrtwine.com/Articles/CodingStyles/CStyle1.htm)
17. US Department of Defense (1994). MIL-STD-498: Software Development and Documentation. (Reviewed online 11/24/15 at www.abelia.com).

Appendix I: Interview Questionnaire

Interviewee Data

Use this questionnaire to capture data about the interviewee, including professional and biographical data.

Name:	________________
Company:	________________
Number of years with company:	______
Position:	________________
Number of years in position:	______
Number of years in current career:	______
Professional Interests:	________________
Education:	________________
Reason for responding to interview request:	________________

Interviewee Responses

Use this questionnaire to capture the interviewee's responses to the questions.

In your opinion, what is the most critical issue facing the software industry at large? Please explain.

What is the most critical issue facing your organization? Please explain.

What suggestions would you have for any professional attempting to devise a training and/or documentation solution to these issues?

Which types of issues are the most difficult to solve: purely technical issues or those involving human factors? Why?

Rank the following IT trends in order of importance, either in terms of impact or degree of difficulty to solve. If you believe that another trend or issue is more important, rank **Other** as number one and write the trend or issue in the blank provided.

______ Cybersecurity:
______ Gamification:
______ Cloud technology:
______ Development Cooperation:
______ Platform Cocreation:
______ Platform Adoption:
______ Innovation Measurement:
______ Open-Source Software:
______ Crowdsourcing:
______ Other: ____

Please explain why/how you chose your number one response.

____
____
____
____

What resources would you recommend to any professional attempting to find a solution to these issues (books, web sites, courses, etc.)?

____
____
____
____

Who else should I speak with, either within your organization or within your professional network? What is your contact's area of expertise?

____
____
____
____

Appendix II: Interview Schedule

Table 2 – Software Engineer Participants

Name	Position Title	Interview Date	Interview Location	Project Review Date
Katherine Theresa Walker	Software Engineer, Alstom Signaling Inc.	06/15/14	Rochester, NY	11/28/15
Saravana Kumar	CTO, CausBuzz Inc.	08/03/14	Henrietta, NY	No Review
Jeremy Morgan	Software Engineer, Dynetics Inc.	09/07/14	Remote - Madison, AL	11/18/15
Harshavardhan Srinivasan	Software Engineer, Intuit, Inc.	10/05/14	Remote - San Diego, CA	No Review
Carlos Osorio	President, LJ Solutions, LLC	11/06/14	Randolph, NJ	No Review
John Freund	Technical Lead, Freund Associates Technical Communications	11/15/14	Mahwah, NJ	No Review
Mark Mikolajczyk	Software Engineer, Exelis (Formerly ITT)	12/11/14	Mahwah, NJ	No Review
Martin Pain	Software Developer, IBM	03/20/15	Remote - Romsey, UK	11/17/15
Leon Vaks	Software Engineer, SBS Design	03/26/15	Paramus, NJ	No Review
Allen George	Software Engineer, ActionIQ	05/16/15	New York, NY	No Review
Apurv Kulkarni	Software Engineer, Mobiquity, Inc.	08/22/15	Philadelphia, PA	No Review
Walter Dearing	Software Engineer, Kunversion, LTD	08/25/15	Beacon, NY	No Review
Adam Torres	Software Engineer, Live Technologies	08/30/15	Warwick, NY	No Review

Name	Position Title	Interview Date	Interview Location	Project Review Date
Lakshman Easwaran	Software Engineer, Stryker	09/10/15	Mahwah, NJ	No Review
Chris Sharp	Software Engineer, IBM	09/23/15	Remote - Romsey, UK	No Review
David Merillat	Software Engineer, Allworx, Inc.	09/23/15	Remote - Rochester, NY	No Review
Mark Reynolds	Software Engineer, (Company Confidential)	09/24/15	Remote - Rochester, NY	11/18/15
Brandon Flaherty	Software Engineer, The New Office	09/24/15	Remote - Austin, TX	No Review
Ken Bolton	Software Engineer, BScientific	10/07/15	Beacon, NY	No Review
Srinivas Seeram	Software Engineer, Mobiquity, Inc.	10/12/15	Remote - Philadelphia, PA	No Review