SCORING A SOFTWARE DEVELOPMENT ORGANIZATION WITH A SINGLE NUMBER

BY

RYAN M. SWANSTROM

A dissertation submitted in partial fulfilment of the requirements for the
Doctor of Philosophy
Major in Computational Science and Statistics
South Dakota State University
2015

SCORING A SOFTWARE DEVELOPMENT ORGANIZATION
WITH A SINGLE NUMBER

This dissertation is approved as a creditable and independent investigation by a candidate for the Doctor of Philosophy degree and is acceptable for meeting the dissertation requirements for this degree. Acceptance of this does not imply that the conclusions reached by the candidates are necessarily the conclusions of the major department.

Dr. Gary Hatfield
Dissertation Advisor Date

Dr. Kurt Cogswell
Head, Math and Statistics Date

Dean, Graduate School Date

ACKNOWLEDGEMENTS

I would like to acknowledge the generous support I received from my family. Thank you to Emily for providing encouragement and the final push to get me to eventually finish. Thank you to Ainsley, Porter, Trey, and Ryker for always providing a smile.

I would also like to acknowledge numerous other people for providing guidance and support during the process. I will only use first names, but you know who you are. Thank you to: Jess, Chris, Clay, Bob, Todd, Leslie, Mike, and Ralph.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

ABSTRACT

SCORING A SOFTWARE DEVELOPMENT ORGANIZATION WITH A SINGLE NUMBER

RYAN M. SWANSTROM

2015

Nearly every large organization on Earth is involved in software development at some level. Some organizations specialize in software development while other organizations only participate in software development out of necessity. In both cases, the performance of the software development matters. Organizations collect vast amounts of data relating to software development. What do the organizations do with that data? That is the problem. Many organizations fail to do anything meaningful with the data.

Another problem is knowing what data to collect. There are many options, but certain data is more important than others. What data should a software development organization collect?

This paper plans to answer that question and present a framework to gather the right information and provide a score for an organization that produces software. The score is not to be comparative between organizations, but to be comparative for a specific organization over time.

The primary goal of this work is to provide a general framework for what a software development organization should measure and how to report on those measurements. The focus is providing a single number to represnet the entire organization and not just the development efforts. That single number is considered the MPI score. The secondary goal of this work is to provide a software implementation of that framework.

1 INTRODUCTION

Software is becoming a vital part of companies. In 2011 Marc Andreessen, co-founder of the venture capital firm Andreessen-Horowitz, famously claimed, "Software is Eating The World" [2]. His argument was for the ever increasing importance of software in all organizations big and small regardless of the industry. With this important declaration, the production of new software is going to be critical. Just as important will be the effective measurement of how this software is produced.

This work provides a technique for a software development organization to create a single number score which indicates the overall performance of the organization. The score is based upon data collected for 5 result indicators of a software development organization: quality, availability, satisfaction, schedule, and requirements. It is not meant to be comparative between organizations, but to form a historical baseline for a specific organization.

1.1 OVERVIEW

Software development organizations are no different than any other business or organization. There are: tasks to be completed, goals to achieve (or miss), and measurements to be analyzed. One difficulty with software development is the varied number and amount of measurements to be used. It can be difficult to determine the correct activities to measure and the appropriate mechanism to report the measure. This has led organizations to either collect too little information or to collect too much information. Another problem is the inconsistency of the reported measures. It is difficult to compare historical performance if the same measurements are not consistent throughout the recent history of an organization.

Software development organizations need a framework to define what

measurements should be tracked and how those measurements should be reported. The MPI framework provides a solid foundation for a consistent evaluation. MPI analyzes the historical performance of an SDO to create a baseline in order to provide a broad view of the overall organization. It is common for software development organizations to measure and focus soley on the source code being produced. However, a software development organization does more than just produce source code. There is documentation to be written, testcases to be created, systems to be deployed, and decisions to be made. The framework provides an evaluation of the overall software development organization, not just the source code.

The framework will produce a single number score for each of the five result indicators as well as a single overall score. It will be able to provide a quick evaluation of the organization. The scores will enable performance to be consistently measured and compared.

Other attempts at evaluating a software development organization have been presented, but none produce a single number score for the entire effort of the software development organization. The following are attempts to evaluate all or parts of software development.

1.2 CMMI

The Capability Maturity Model Integration (CMMI) is one of the most widely acknowledged models for process improvement in software development. CMMI offers a generic guideline and appraisal program for process improvement. It was created and is administered by the Software Engineering Institute at Carnegie Mellon University [16]. While the CMMI is not specific to software development, it is often applied in software development settings. CMMI certification is required for many United States Government and Department of Defense contracts.

CMMI-Dev is a modification of the CMMI specific to the development activities applied to products and services. The practices covered in CMMI-Dev include project management, systems engineering, hardware engineering, process management, software engineering, and other maintenance processes. Five maturity levels are specified, and they include the existence of a number of process areas. The 5 maturity levels and the process areas are specified as follows.

CMMI MATURITY LEVEL 1 - INITIAL A maturity level 1 organization consists of an adhoc and chaotic processes. While working products are still produced, the results are often over budget and behind schedule. A level 1 organization will also have difficulties repeating a process with the same degree of success. These organizations typically rely on the heroic efforts of certain individuals.

CMMI MATURITY LEVEL 2 - MANAGED A maturity level 2 organization has a policy for planning and executing processes. The processes are controlled, monitored, reviewed, and enforced. The practices are even maintained in times of stress. The following process areas should be present at maturity level 2.

- Configuration Management (CM)
- Measurement and Analysis (MA)
- Project Monitoring and Control (PMC)
- Project Planning (PP)
- Process and Product Quality Assurance (PPQA)
- Requirements Management (REQM)
- Supplier Agreement Management (SAM)

CMMI MATURITY LEVEL 3 - DEFINED A maturity level 3 organization has well-understood processes that are described in standards, tools, procedures, and methods. The organization has standard processes that are reviewed and improved over time. The major differentiators between level 2 and level 3 is the existence of standards and process descriptions. A level 2 organization will have processes that are inconsistent across projects. A level 3 organization will tailor a standard process for each project. Also, level 3 processes are described with much more rigor. In addition to the process areas found in level 2, the following process areas should be present at maturity level 3.

- Decision Analysis and Resolution (DAR)
- Integrated Project Management (IPM)
- Organizational Process Definition (OPD)
- Organizational Process Focus (OPF)
- Organizational Training (OT)
- Product Integration (PI)
- Requirements Development (RD)
- Risk Management (RSKM)
- Technical Solution (TS)
- Validation (VAL)
- Verification (VER)

CMMI MATURITY LEVEL 4 - QUANTITATIVELY MANAGED A maturity level 4 organization has quantitative measures for quality and process performance. The measures are based upon customer needs, end users, and process implementers. The quality and process performance are

understood mathematically and managed throughout the life of a project. Level 4 is characterized by the predictability of the process performance. In addition to the process areas found in level 2 and 3, the following additional process areas should be present at maturity level 4.

- Organizational Process Performance (OPP)
- Quantitative Project Management (QPM)

CMMI MATURITY LEVEL 5 - OPTIMIZING The final and pinnacle level of CMMI maturity is level 5. A maturity level 5 organization continually improves processes based upon quantitative measures. The major distinction from level 4 is the constant focus on improving and managing organizational performance. A maturity level 5 organization has well-documented standard processes that are tracked and enforced as well as a focus on continual improvement of the processes based upon quantitative measures. In addition to the process areas of the previous maturity levels, maturity level 5 should contain the following process areas.

- Causal Analysis and Resolution (CAR)
- Organizational Performance Management (OPM)

A visual description of the CMMI maturity levels can be seen in Figure 1. While CMMI-Dev does provide an excellent framework for improving a process, it is wholely focused on process improvement. It does not provide guidelines for evaluating the final product. Also, it does not provide a specify mechanism for evaluating or scoring the progression through the maturity levels. An indicator is still needed to quantify the overall performance of an organization, not just the compliance to standard processes.

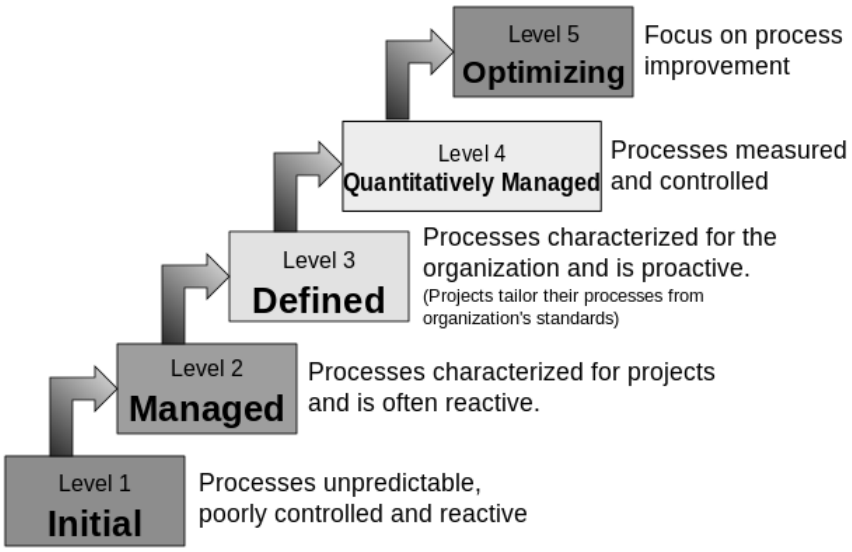

Figure 1: Characteristics of CMMI, image adapted from [26]

1.3 PREVIOUS SOFTWARE EVALUATION WORK

An example of scoring software development is presented by Jones [39]. The methodology looks for the presence of various techniques used in software engineering. The methodology provides a score based upon the productivity and quality increase of the technique being evaluated. Points are positive or negative based upon the presense of various techniques. A couple example techniques are: automated source code analysis and continuous integration. The end result is a score in range $[-10, 10]$. While the result is a single number score, it does not account for the entirety of the software development lifecycle.

Constructive Cost Model (COCOMO) is a software cost estimation model created by Boehm [11]. It combines future project characteristics with historical project data to create a regression model to estimate the cost of a software project. The original version developed in 1981 was focused on mainframe and batch processing. An unpdated version, named COCOMO II, was created by Boehm in 1995 to be more flexible for newer development practices such as desktop development, off the shelf components, and code reuse. COCOMO provides a nice

algorithm for making decisions regarding building or buying software products. It does not provide an algorithm to review and modify past performance based upon estimates. COCOMO II can be a useful tool for estimating the time and costs within an SDO, but it only provides an estimate and not an evaluation of the actual performance.

Sextant is a visualization tool for Java source code [76]. Sextant provides a graphical representation of the information related to a software system. The tool provides the capability to provide custom rules which are specific to the domain or application. However, Sextant only provides metrics and analysis of the software code. It provides no information regarding the rest of the software development lifecycle. Also, the primary output of Sextant is visual graphs. While these graphs do provide useful information, they do not provide a single number to determine the performance of the software.

Another promising research area is *process mining*. As stated by Wil van der Aalst [73], "The idea of process mining is to discover, monitor and improve real processes (i.e., not assumed processes) by extracting knowledge from event logs readily available in today's systems." Creating software involves a vast amount of processes. Numerous logs of raw data are collected. One application of process mining in the area of software development was an algorithm and information for measuring a software engineering process from the Software Configuration Management (SCM) [63]. The technique creates process models to understand the process of developing software and code. It is less focused on the output and results, but it is more focused on adherence to a specified process.

Process mining has also been applied to decision making regarding software upgrades [74]. Historic and current logs can be processed and evaluated. Then small pilot groups can be offered the upgrade, and the new logs will be processed and evaluated. Thus, process mining can be beneficial for new software implementation.

More research needs to be done applying process mining to the other processes involved with software development, such as other documentation, testing, and implementation.

Although process mining can be useful for analyzing software development data, it has not yet been applied to the entirety of a software development organization. It also does not focus on the results of the organization. It focuses on process conformance, which can be very important, so it is limited in the ability to evaluate the entire organization.

Much work has been done to determine metrics for source code, in fact entire books have been written on the topic of software metrics [37, 59]. Yet, organizations still struggle to measure the production of software. Little work exists for scoring the entire software development organization.

1.3.1 SEMAT

Software Engineering Method and Theory (SEMAT) is claiming to be the "new software engineering" [35]. The authors rightfully claim that software engineering lacks an underlying theoretical foundation found in other engineering disciplines. This lack of theory has led software engineering to not be engineering, but rather a craft. The goal of SEMAT is to merge the craftsmanship and engineering to provide a foundation for software engineering. The primary initiative of SEMAT has been the creation of a kernel for software engineering. The kernel is the minimal set of things common to all software development endeavours. The three parts to the kernel are:

Measurement - There must exist a means to determine the health and progress of an endeavour

Categorization - The activities must progress through categories during an endeavour

Competencies - Specific competencies will be required for completing activities

The kernel defines alphas, which are seven dimensions with specific states for measuring progress. The seven dimensions are:

1. Opportunity
2. Stakeholders
3. Requirements
4. Software Systems
5. Work
6. Team
7. Way of Working

Although SEMAT is very promising, the development is not yet complete. Adoption is limited so the technique has not been validated on many actual software engineering endeavours. Although SEMAT does include a part for measurement of progress, it does not specify how the measurement is to be performed.

1.3.2 SOFTWARE QUALITY

Software quality is one of the most well studied aspects of software development. Most of the work focuses on either the problems with the software or the source code. Quality and the number of problems with the software are inversely related, more problems means lower quality. Quality is easy to measure, but that measurement is usually very software specific. It is easy to find that some software has X number of problems, but it is nearly impossible to determine whether the quality of that software is better or worse than some other software with X defects.

One piece of software can be larger[1] or more complex. Thus, finding a value for quality is easier than interpreting that value. No matter the interpretation, the goal is to release the number of problems with the software. Top 10 lists have been created for techniques to remove problems from software [9]. A number of different techniques or best practices for preventing defects have been proposed [23]. These are all strategies to identify or remove the problems before the software is completed and released to users.

Another aspect of software quality is the complexity of the source code. More complex code results in more maintenance efforts and more chances for problems. A couple of numerical measures for the complexity of source code have been created. The most common examples are McCabe [51] and Halstead [29]. However, the measures on source code only explain part of the software development lifecycle.

Another measurement of quality can be the cost per defect, also known as the cost to fix a problem. As seen in [41], this measurement has problems because the lowest cost per defect will occur on software with the most problems. Therefore, the lowest cost per defect is actually the lowest quality as well. Due to this difficulty and others, a number of other models have been created for evaluating the quality of software [54]. While all of the models have merit in certain situations, the measures of quality must be combined with other measures in order to provide an overall evaluation of a software development organization.

1.3.3 SOFTWARE ANALYTICS

One area of research that is focusing on the evaluation of software development organizations is *software analytics*. Software analytics is less focused on evaluation and more so on all sorts of analysis of software data. The earliest variants of

[1]Saying a piece of software is larger can be a rather arbitrary statement. Does that mean the software requires more computing time, has more lines of code, more documentation, more hours spent on development, or some other arbitrary measure.

software analytics were disguised as applications of data mining techniques to software engineering data in the late 1990s [64, 21, 27]. Later the field began to emerge more heavily, but still remained primarily methods of data mining applied to software engineering [44, 77, 70, 28]. Finally, The term software intelligence was proposed for the field of study [31], but eventually the term software analytics became the dominant term for referring to the field of study [12, 78].

The goal of software analytics is to extract insights from software artifacts to aide practitioners in making better decisions regarding software development [79]. The three main focus areas of software analytics are:

1. **User Experience** - How can the software enable the user to more easily or quickly accomplish the task at hand?
2. **Quality** - How can the number of problems with the software be decreased?
3. **Development Productivity** - How can the processes or tools be modified to increase the rate at which software is produced?

Later, Martin Shepperd in [30] identified 3 important questions that software analytics must address:

1. "How much better is my model performing than a naive strategy, such as guessing [. . .]?"
2. "How practically significant are the results?"
3. "How sensitive are the results to small changes in one or more of the inputs?"

These are 3 important questions that should be addressed when presenting any results in software analytics. The research needs to demonstrate clear advantages for practitioners. The work presented in this work will address both the 3 main focus areas and the 3 important questions of software analytics.

One of the software analytic techniques focuses specifically on the source code; analyzing the complexity, size, and coupling [71]. Letier and Fitzgerald discuss how to choose the correct tools and techniques to analyze software data [48]. A goal model is produced that matches the data analysis methods with the goals of the software stakeholders. The method does not focus specifically on analysis of the development of software.

Software Development Analytics is a subfield of software analytics [53]. It focuses specifically on the analytics of the development of software, however not the overall performance of the software. Hassan points out in [30] that software analytics needs to go beyond just the developers. Everyone and everything involved in the development of software produces some data and that data can be meaningful. The insights from non-developer data has the potential to yield important results as well. Software development produces many valuable pieces of datum that can be analyzed [50]. Just a few of the pieces of datum are: email communication, bugs, fixes, source code, version control system histories, process information, and test data. Examples of this type of data can be found in the PROMISE Data Repository [52].

All of these techniques are of no use if the correct data is not available. Therefore, it is important to identify the information that is needed to properly perform software development analytics. Unfortunately, there are vast amounts of information that need to be collected to meet the analytic needs of developers and managers [13]. When the data and tools exists, the analytics should help an organization with the following tasks.

- Evaluate a project
- Determine what is and is not working
- Manage Risk
- Anticipate changes

- Evaluate prior decisions

In order to store the data, appropriate tools are needed. Microsoft is working on developing some tools for the analysis of the development of software [18, 79]. Microsoft has developed StackMine, a postmortem tool for performance debugging, and CODEMINE, a tool for collecting and analyzing software development process data. Both tools provide analytical insight for various aspects of the software development process, however neither tool covers all aspects of software development. These tools are currently early in development and the adoption of the tools by practitioners is still unknown. One of the reasons for the slow adoption of new tools is the inherent difficulty of producing new tools for the software development process [69]. A tool that works fantastic for one team might not automatically apply to another team. The people creating the tools need to be acutely aware of the needs of the technical practitioners that will be using the tools.

After the proper tools are in place to collect the necessary data on software development and software analytics are being properly implemented, an obvious next step is the application of gamification to software development. Gamification is "the process of making activities more game-like" [75]. Some of the benefits of gamification are higher productivity, competition, and greater enjoyment. Prior attempts at gamification of software development focus only on the computer programming phase [67]. Others focus on defining a framework for gamification within the software development process [36]. There are even some indications that gamification might help increase software quality [20]. While this work will not focus on gamification, it is important to note that an implementation of an evaluation technique for software development could be implemented simultaneously with a gamification strategy. Both will require new collections of data and new reporting.

Overall, there exist many attempts to evaluate portions of the a software development organization. None of the the attempts provide a single number score

for the entirety of the organization. Most of the techniques focus on specific portions of the software development lifecycle, namely the development portion. Plus, there are many tools that need to be created for software analytics to provide all the value that it promises.

1.4 ORGANIZATION OF THE WORK

The remainder of this dissertation is divided into 5 chapters. The next chapter provides an overview of software, software development lifecycles, software engineering, and software development organizations. Chapter 3 introduces what is meant by the term data-driven software engineering. Chapter 4 then provides an explanation of the Master Performance Indicator (MPI). It will present the essential elements for calculating the MPI, as well as the formulas, framework, and data necessary to produce the MPI. Chapter 5 provides a technological framework for generating and storing the current and historical MPI values. Chapter 6 demostrates how MPI can be implemented in a software development portion of a large financial institution. Chapter 7 concludes the dissertation with a summary of the results and some possible future directions for further enhancements.

2 A SOFTWARE DEVELOPMENT ORGANIZATION (SDO)

A *Software Development Organization* is any organization or subset of an organization that is responsible for the creation, deployment, and maintenance of software. Many times a software development organization is a company that produces software. Other times, a software development organization is contained within the Information Technology department of a larger organization. Some of the job roles with an SDO are: software engineer, system administrator, software quality analyst, programmer, database administrator, and documentation specialist.

2.1 WHAT IS SOFTWARE?

Numerous definitions can be found for the term *software*. Software is more than just computer programs. According to Ian Sommerville [68], "Software is not just the programs but also all associated documentation and configuration data which is needed to make these programs operate correctly." This is the definition used for the remainder of this work.

2.2 THE SOFTWARE DEVELOPMENT LIFE CYCLE

The discipline of software engineering has created a workflow for developing software. This workflow is called the *Software Development Life Cycle (SDLC)*. SDLC can be defined as [65]:

> [...] a conceptual framework or process that considers the structure of the stages involved in the development of an application from its initial feasibility study through to its deployment in the field and maintenance.

While the SDLC states what needs to be done, there are numerous models that formalize exactly how to perform the SDLC. The models contain steps that are commonly referred to as a phases. A few of the popular models are described below.

2.2.1 WATERFALL

The waterfall model is the oldest and most influential of the SDLC models. It was first presented at a Navy Mathematical Computing Advisory Panel in 1956 by Herb Benington [6]. Figure 2 shows the model Benington outlined for producing large software systems. In 1970, Benington's model was modified by Royce [62]. Royce produced an updated version of the diagram seen in Figure 3 which provides some loops to go back to a previous phase in the workflow.

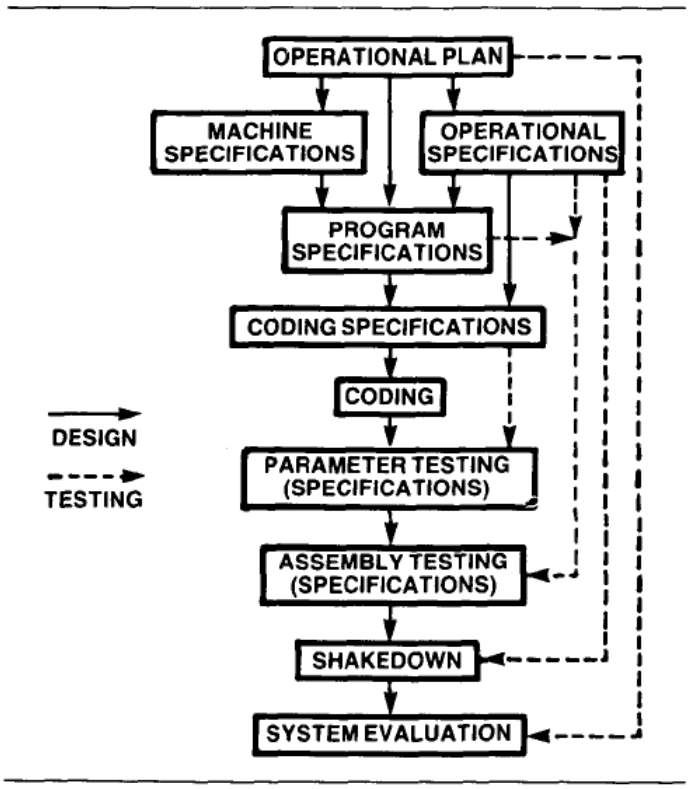

Figure 2: Benington's original diagram for producing large software systems, adapted from [6]

The modern version of the waterfall model specifies that each phase needs to be entirely completed before moving onto the next phase. Some small amount of overlap is permitted and looping occurs but both actions are discouraged and should be limited. A modern diagram of the waterfall model can be seen in Figure 4.

Waterfall has some excellent features such as: simple to understand, easy to plan, and well-defined phases. However, waterfall lacks the flexibility required of many software systems built today [49]. Due to the fact the phases are so sequential, it makes changes during the life cycle difficult and expensive if not impossible. Therefore, other models of SDLC have been created to address the lack of flexibility of the waterfall model. Notice, the other models are adaptations of waterfall.

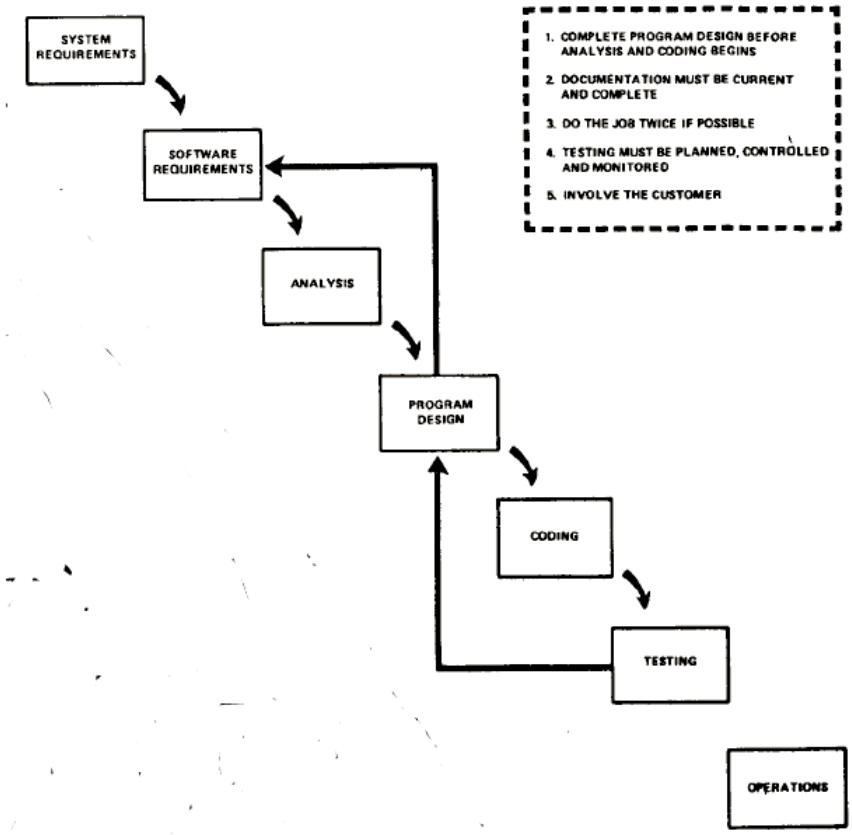

Figure 3: Royce's version of the waterfall model for producing software systems, adapted from [62]

2.2.2 SPIRAL

The spiral model for software development was presented by Boehm in 1986 [7, 8]. The goal of the spiral model of software development is very risk-driven. A software project will start with many small and quick iterations. Each iteration will cover the following 4 basic steps.

1. Determine Objectives
2. Identify Risks
3. Develop and Test
4. Plan Next Iteration

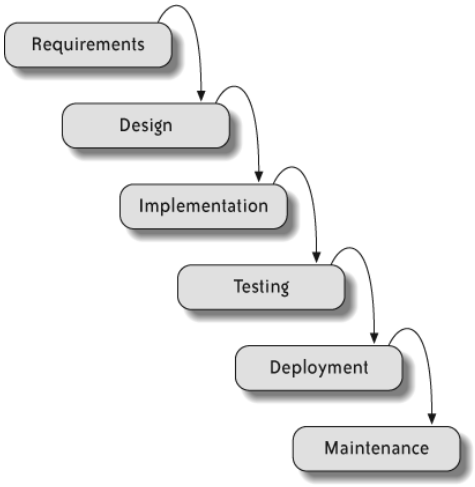

Figure 4: Modern Waterfall Diagram, adapted from [32]

This model allows software to be built over a series of iterations without risking too much time or effort in any single iteration. Sprial requires a very adaptive management approach as well as flexibility of the key stakeholders [65]. It can also be difficult to identify risks that will occur in future iterations. Figure 5 provides a bit more detail on the iterations and the overall process.

2.2.3 AGILE

Agile software development has arisen due to the inability of the waterfall and other models to adjust to changes during the development cycle. Agile software development is a group of SDLC models that operate under the influence of the following four key principles [5].

1. Individuals and interactions over processes and tools
2. Working software over comprehensive documentation
3. Customer collaboration over contract negotiation
4. Responding to change over following a plan

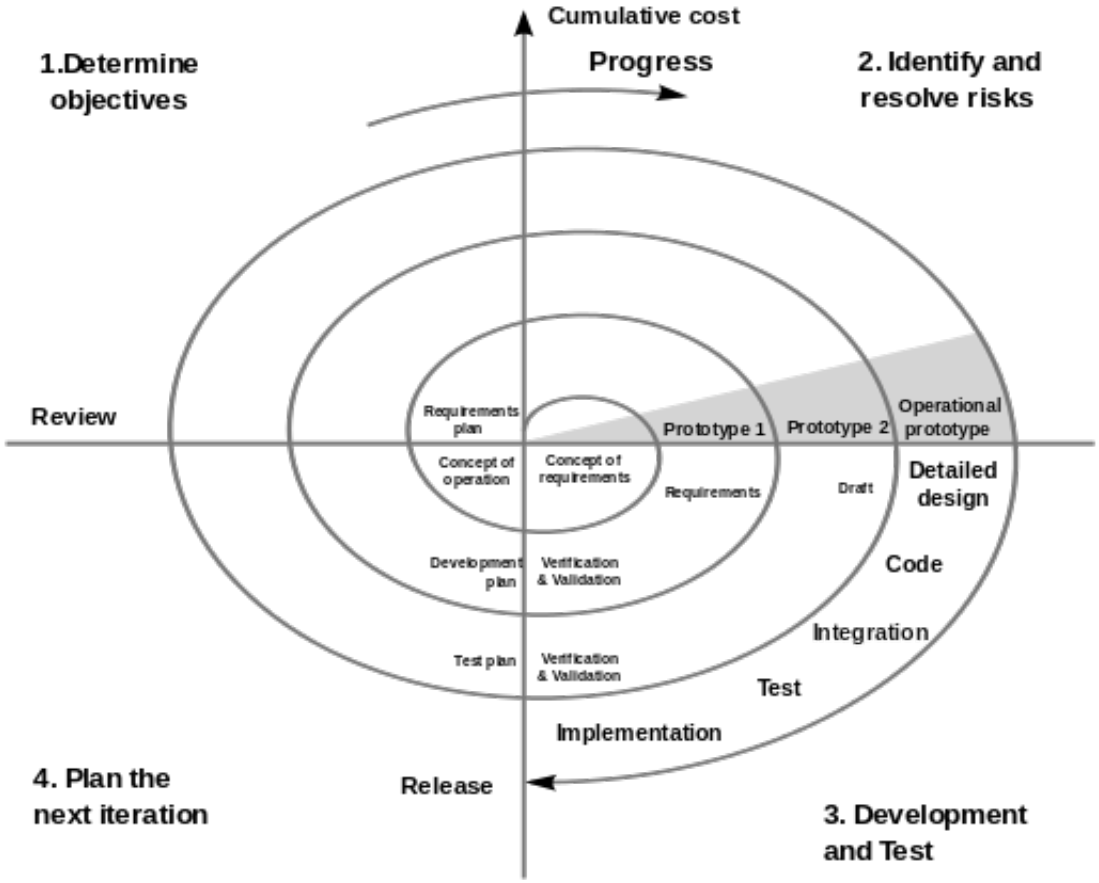

Figure 5: Spiral SDLC Model [10]

Agile does not specify an implementation, but some specific models of agile SDLC are: eXtreme Programming, Scrum, Lean, Kanban and others [55, 32]. Agile models are very popular in many of today's software development organizations because the models work well for dynamic quickly changing applications such as web-based applications. Startups have largely adopted the Lean methodology for its ability to identify a minimum viable product[2] and reduce the time to market [25].

2.2.4 SDLC COMMONALITIES

Even with the large number of SDLC models currently being used by different SDOs, many ommonalities exist among the models. The commonalities can be tied

[2]A *minimum viable product* is a reduced version of software that contains only the bare minimum functionality required to meet the requirements.

back to the steps of waterfall. All of the models exhibit, to some degree, the following phases. The only major difference is the scope, size, and duration of each phase. For example, the spiral model spends less time in each phase. The agile models produce less documentation and focus more on the Implemenation phase. Here are the common phases in nearly all SDLC models:

1. Requirements

The first phase is involved with defining what the software must do. Each piece of functionality is considered a requirement.

2. Design

Before writing any code, the necessary infrastructure and involved software systems must be identified. This phase can serve as a roadmap for the remaining phases. If done properly, this phase can greatly help the later phases.

3. Implementation

Often the only phase of the SDLC that is measured, this is the phase where the actual computer code is written.

4. Testing

This phase validates the expected functionality. Also, testing attempts to discover unexpected side affects of the software.

5. Deployment and Maintenance

All software must be correctly deployed and maintained. This phase is the most expensive and lengthy phase of the software development lifecycle.

The WHAT of SDLC - These are the steps of SDLC that need to be completed for each project.

- Identify the Work/Task/Project

  - Get Initial Idea
  - Obtain Details
- Estimate
  - Create an Estimate (What is included? What is the output? days/dollars/hours/reqs) Obtain Approval
    - Quit or Go Forward
- Document Requirements
  - Identify the Requirements
  - Detail the Requirements
- Design The Software
  - Find System Integrations
  - Identify Functional Specs
  - Detail the Functional Specs
- Development of all the tasks in Design and Requirements
  - Identify the Coding Tasks
  - Write the Code/Develop the solution
  - Write the Unit Tests
- Test
  - Create Test Plans and Cases
  - Run Test Plans and Cases
- Deployment
  - Create Deployment Steps
  - Run Deployment Steps

- Maintenance
  - Capture Bugs
  - Survey Users

2.3 WHAT IS SOFTWARE ENGINEERING?

Software Engineering as a term dates back to the 1968 North Atlantic Treaty Organization (NATO) conference [72, 56]. Over the years many definitions have been provided. IEEE provides a definition that encompasses many of the other definitions. IEEE ISO610.12 defines software engineering as, "The application of a systematic, disciplined, quantifiable approach to the development, operation, and maintenance of software" [1].

Software engineering has struggled to determine the correct projects to complete [19]. Software projects are commonly behind schedule and over budget [47, 15, 42, 34] with nearly 20% of projects in the United States still failing [22]. As of 2015, Software engineering is still awaiting the professional status of more established fields such as medicine, law, and general engineering [40]. Organizations need a better technique to understand the past performance so they can better predict the future performance. Proper measurement will be essential to solidifying software engineering as certified profession.

2.4 TERMINOLOGY

App (Application) - a software system or a collection of other apps, item[Project] - A body of work involving zero or more apps in preparation for a release, infrastructure upgrades can be a project but no specific app is involved

Release - A collection of projects being put in production on a specified date

SIT (Systems Integration Testing) - The initial step of testing after the

development phase of the SDLC. This is typically performed by members of the SDO. It is validation that all the software components function together as expected.

UAT (User Acceptance Testing) - The final step of testing when a select few members of the user group are invited to validate the software system. Once validation has occured for UAT, the software system is ready to proceed to production

PROD (Production) - The software has been released to the final audience.

defect "A software defect is a bug or error that causes software to either stop operating or to produce invalid or unacceptable results" as quoted from [38]. It is important to mention that even though defects are typically found in the computer code, a defect should not be isolated to just code. A poorly written requirement or missed test cases can both be considered a defect. Other common names for a defect are: bug, error, fault, or ticket.

Also in here is the 4 severity levels. Also, current defect removal is only about 85% and this value should be increased to about 95%.

Check these book [38, 37, 46].

3 MEASURING AN SDO

Anything can be measured [33]. An SDO how numerous metrics.

A metric can be defined as the science of indirect measurement. The following are some examples of metrics that can be collected about source code:

- SLOC - The number of Source Lines of Code
- NOM - The Number of Methods per class

- Complexity - A numerical measure of the code complexity, some common examples are McCabe [51] and Halstead [29]
- Design - The amount of coupling and cohesion present in the software code
- Source Code Analysis - Tools that determine whether code adheres to specified set of rules. Common examples are PMD[3] and FindBugs^{TM} [17, 4].

Data Driven Software Engineering not Data Driven Software Development

3.1 INDICATORS

Indicators can be crucial measurements within any business setting, and an SDO is no exception. Determining the correct indicators for an organization can be difficult. Software development organizations The differences between the indicators will be explored and possible measures for each indicator in an SDO will be presented. The 4 different types of indicators are:

RI (Result Indicator) - what has been done

KRI (Key Result Indicator) - how you have done

PI (Performance Indicator) - what to do

KPI (Key Performance Indicator) - what to do to dramatically increase performance

More information on the indicators can be found in chapter 1 of [58] as well as [24, 45].

3.1.1 RESULT INDICATORS FOR SDO

This section will cover the potential result indicators for an SDO.

[3]PMD is a source code analysis product. It is not an acronym.

3.1.2 KEY RESULT INDICATORS FOR SDO

3.1.3 PERFORMANCE INDICATORS FOR SDO

This section will cover the potential performance indicators for an SDO.

3.1.4 KEY PERFORMANCE INDICATORS FOR SDO

3.2 BALANCED SCORECARD

A common technique for organizations to measure themselves is a balanced scorecard. A *balanced scorecard* must contain the following six characteristics.

1. Financial
2. Customer Focus
3. Environment/Community
4. Internal Process
5. Employee Satisfaction
6. Learning and Growth

Balanced scorecards are great for displaying the important information about an organization. However, a balanced scorecard does not specify what exactly needs to be tracked. It is also not specific to an SDO and it does not produce a single number.

This info comes from [58].

3.3 PROJECT MANAGEMENT MEASUREMENT

According to Putnam and Myers in [59], the 5 core measurements for managing software projects are:

1. **Quantity of function** usually measured in terms of size (such as source lines of code or function points), that ultimately execute on the computer
2. **Productivity** as expressed in terms of the functionality produced for the time and effort expended
3. **Time** the duration of the project in calendar months
4. **Effort** the amount of work expended in person-months
5. **Reliability** as expressed in terms of defect rate (or its reciprocal, mean time to defect)

However, these measurements don't focus on the entire SDO. They say nothing about the availablity of the software infrastructure or the satisfaction of the users.

Time This measurement can be hours, days, weeks, or months. the time increment does not matter as long as it is consistent.

Effort This measurement is person-months, person-weeks, or person-days.

Quality This measurement is the defect rate (defects per time period).

Amount of Work This measurement has the same units as Effort above.

Process Productivity look into this a bit further

Function Points find a definition and a comparison with story points

Story Points Look in some agile book

Chapter 7 of [59] contains some equations to calculate and relate these measures. Those equations will be studied and evaluated.

Antolic in [3] demostrates some ways to measure KPIs for the software development process.

3.4 MEASURING OTHER ASPECTS OF AN SDO

It is important to note that software departments do not just develop software. The department has many other duties including: deploying software, installing server hardware/software, writing documentation, surveying users, and other things.

3.5 COMBINING THE MEASURES

How can the PIs, RIs, KRIs, and KPIs be combined to form a single value called an MPI (Master Performance Indicator)? The field of sports analytics also has numerous techniques for combining multiple indicators to produce a single number [14].

4 MASTER PERFORMANCE INDICATOR (MPI)

Software development organizations struggle to measure overall performance. The Master Performance Indicator (MPI) is an algorithm to provide a single number score to measure the performance of a software development organization. It works by statistically analyzing the past performance of the organization and using that information to score an organization on current performance. MPI focuses on the following elements of a software development organization: *quality*, *availability*, *satisfaction*, *schedule*, and *requirements*. A separate MPI score is calculated for each element and then aggregated together to form an overall MPI score. A new MPI score will be calculated based upon the selection of a given time period (weekly, monthly, quarterly). **MPI is not meant to be comparative between organizations, but to measure the amount of increase or decrease a single organization exhibits across elements**. MPI is made to be easily expandable to other elements if desired.

The scores for MPI will range from -1, indicating the worst performance, all

the way to +1, indicating perfection. A score of 0 is an indication of meeting the basic expectations. A negative score indicates under-performance and a positive score indicates over-performance. Here are some examples. An MPI score of 0.35 means the organization is performing 35% better than expected. Conversely, a score of -0.15 means an organization is performing 15% worse than expected.

Below are the attributes of the MPI scoring.

- The range of scores must have equal values above and below 0
- The minimum score must equate to the worst possible performance, however that is defined
- Similarily, the maximum score must equate to the best possible performance.
- A score of 0 must be average (or expected) performance
- All individual elements must have the same scoring range

As long as those 5 features are met, the range can be anything. The range of $[-1, 1]$ was chosen because it is easy to scale to a different range such as $[-10, 10]$ or $[-100, 100]$. It maybe makes sense to use variables for the range. The formulas are designed to fall in the range $[-100, 100]$. Thus scaling can be applied to obtain values in any appropriate range. The scaling is denoted with the variable k. The scale will be the same for all 5 elements.

4.1 ELEMENTS OF MPI

The MPI score consists of data collected from five different elements of an SDO.

1. Quality
2. Availability
3. Satisfaction

4. Schedule

5. Requirements

These five elements will be outline in the next sections.

4.1.1 QUALITY

Measuring quality is a crucial part of accessing software development results. Poor quality means time, money, and resources are spent fixing the problems. As a result, new features are not being created. One of the key indicators of software quality is defects. Thus, it is important to measure the number of defects associated with a software release. The following is the necessary data to collect and an algorithm to calculate the quality score of software being released.

QUALITY DATA In order to properly score the quality of an SDO, certain data needs to be obtained in order to measure performance. Table 1 identifies the columns of data that will be used to create a score for the quality element of MPI. Each column is classified as *required* or *optional*. That is to allow some flexibility in the model for organizations that collect varying amounts of data.

Column Name	**Data Type**	
Application ID	String (factor)	Required
Frequency Date	Date	Required
Development Effort	Integer	Required
Testing Effort	Integer	Optional
SIT Defects	Integer	Optional
UAT Defects	Integer	Optional
PROD Defects	Integer	Required

Table 1: QUALITY DATA NEEDED FOR MPI

The development and testing effort can come from any of the following choices for effort. It is possible that other measures will work for effort.

Actual Time This number is a representation of the total amount of time spent on a project. This number can be measured in any unit of time: hours, days, weeks, etc. Actual time can be applied to development or testing effort.

Estimated Time This number is a representation of the initial estimated amount of time spent on a project. This number can be measured in any unit of time: hours, days, weeks, etc. Estimated time can be applied to development or testing effort. It is common for the estimated and actual times to be diffent.

Source Lines Of Code (SLOC) This number is the count of the total number of lines of source code for a project. Obviously, this item only counts as a level of effort for development unless coding in used to generate automated test cases. [4]

Modified Lines Of Code This number is a count of the number of modified lines of source code. Modified lines is defined as the number of deleted, added, and modified lines of source code. This number is different from above since it does not include all the lines of source code. Similar to above, this number makes more sense for development effort.

Test Cases A test case is a step or series of steps followed to validate some expected outcome of software. Organizations will create a number of test cases to be validated for a software system. The number of such test cases could be used as a level of testing effort.

[4] Automated testing is the process of creating software to automatically run tests against other software. The adoption of automated testing is varied and it is not a solution in all cases [60].

QUALITY FORMULA The first step in creating a score for the quality element is analysis of the historical data. The historical data is all quality data collected before a given point in time. Some common historical cutoffs are the current date or the end of the previous fiscal year. Then a mathematical model to predict PROD Defects will be produced. In statistical terms, the response is *PROD Defects* and the predictors are: *UAT Defects*, *SIT Defects*, *Testing Effort*, and *Development Effort*. Some of the following strategies will be employed to find a reasonable model.

- Removal of outliers and/or influential points
- Linear Regression
- Stepwise Regression
- Ridge Regression for suspected multicolinearity

Once a model has been found, it will be labeled as f and it will not change. The function f can be the same for all Application IDs or it can be different for each Application ID or any combination of Application IDs. It serves as the quality baseline for MPI. All future quality scores will be dependent upon the original f. Once set, the model will not change.

After the model f has been determined, it is time to calculate the quality score for each application ID within the given time period. The quality score for each Application ID can be calculated as follows.

$$S_{1_i} = \left\{ \begin{array}{ll} \text{where } f_i \geq d_i & : \frac{f_i - d_i}{f_i} \times k \\ \text{where } d_i > f_i & : \frac{f_i - d_i}{6\sigma_i} \times k \end{array} \right] \text{, calculate quality score for each app } i$$

where

- S_{1_i} is the quality score for Application ID i

- n is the number of Application IDs
- d_i is the actual PROD defects
- f_i is the function to predict PROD defects for Application i based upon *UAT Defects*, *SIT Defects*, *Testing Effort*, and *Development Effort*
- 6σ 6 times an estimate of the standard deviation of the population using function f_i above

Then the overall quality score is calculated as below.

$$S_1 = \sum_{i=1}^{n} w_i' S_{1_i} \text{ where } \sum_{i=1}^{n} w_i' = 1$$

where

- S_1 is the combined quality score for all Application IDs, a weighted average

Then S_1 represents the quality score for that given time period.

4.1.2 AVAILABILITY

All the new requirements and great quality do not matter if the software is not available. All the new features and great quality do not matter if the software is not available. Thus it is essential to set an expected Service Level Agreement (SLA)[5] and measure performance against that SLA. The following section will outline to data needed to properly calculate an SLA and the data needed to calculate the MPI score for availability.

Special Note: The System ID for availability does not have to be the same as the Application ID for quality or any of the other elements. Some organizations have a one-to-one mapping between Applications being developed and systems being

[5]For an SDO, an SLA is a contract specifying the amount of time software will be available during a given time period.

deployed. Others have more complex sceranios that require multiple applications to be combined to form a system. Then the availability of the system is tracked.

AVAILABILITY DATA Table 2 identifies the necessary data to calculate the MPI element score for availability. Notice the three optional fields: *Uptime*, *Scheduled Downtime*, and *Unscheduled Downtime*; they are optional because they can be used to calculated the *Percent Uptime*. The *Percent Uptime* is the important value for the MPI schedule score. Here are the two common approaches for calculating percent uptime:

The preferred method

$$\text{Percent Uptime} = \frac{\text{Uptime}}{\text{Uptime} + \text{Scheduled Downtime} + \text{Unscheduled Downtime}}$$

and the alternative method

$$\text{Percent Uptime} = \frac{\text{Uptime}}{\text{Uptime} + \text{Unscheduled Downtime}}$$

The only difference is the removal of scheduled downtime from the calculation. The calculation approach is typically specified in the contract associated with the SLA. Thus, the Percent Uptime is important and it can either be supplied in the data or calculated from the optional fields.

Column Name	**Data Type**	
System ID	String	Required
Frequency Date	Date	Required
Uptime	Float	Optional
Scheduled Downtime	Float	Optional
Unscheduled Downtime	Float	Optional
Percent Uptime	Float	Required

Column Name	**Data Type**	
Expected Percent Uptime	Float	Required

Table 2: AVAILABILITY DATA NEEDED FOR MPI

AVAILABILITY FORMULA The formula for availability is more straightforward than the quality formula. It does not include any analysis of the historic data. That lack of historical analysis is avoided since the SLA provides an existing baseline to measure against. The following formula is simply a percentage the SLA was exceeded or missed.

$$S_{2_i} = \begin{cases} \text{where } A_{a_i} \leq A_{e_i} & : \left[\frac{A_{a_i} - A_{e_i}}{A_{e_i}} \times k\right] \\ \text{where } A_{a_i} > A_{e_i} & : \left[\frac{A_{a_i} - A_{e_i}}{1 - A_{e_i}} \times k\right] \end{cases}, \text{ calculate availability score for each app } i$$

where

- A_{a_i} actual availability for System ID i
- A_{e_i} expected availability for System ID i

Then the overall availability score is calculated as below.

$$S_2 = \sum_{i=1}^{n} w_i' S_{2_i} \text{ where } \sum_{i=1}^{n} w_i' = 1$$

4.1.3 SATISFACTION

The satisfaction of the customer is also very important. The following is an algorithm to calculate the satisfaction of software.

$$S_2 = \frac{\sum_{i=1}^{n} \left(\frac{\sum_{j=1}^{m} a_{ij} - \frac{min+max}{2}}{m} \right)}{n}$$

- a_{ij} answer to question j for app i
- m number of questions with equal weights
- n number of apps
- min minimum score for a question
- max maximum score for a question

Column Name	**Data Type**	
Question ID	String	Required
Question Text	String	Optional
Application ID	String	Required
Frequency Date	Date	Required
Response	Integer between min and max	Required
Response Date	Date	Optional

Table 3: SATISFACTION DATA NEEDED FOR MPI

4.1.4 SCHEDULE

Delivery of software in a timely manner is an essential part of being a successful SDO. Being able to meet scheduled deadlines is a sign of accurate estimation and planning. Drastically missing deadlines is a sign of an SDO with a process that needs refinement. Studies have shown that software projects exceed the estimates by an average of 30% [43]. Thus it is important to score SDOs on accurate schedule adherence. Without tracking and measuring schedule adherence, it will not improve.

The score is based upon the historical deviance of estimates for projects. The top score of 100 will be obtained when the schedule is met exactly. A historical

average of deviance of schedule will be used to determine an appropriate range of schedule adherence. A schedule that occurs within that range will be awarded a score of 0 or above, with less deviance being awarded more points. Schedules that do not fall within the range of deviance will be awarded negative scores, with a greater deviance resulting in a lower score all the way to -100 which means the package took twice as long as expected or more. For example, a package that takes more than twice as long to deliver than estimated will receive a score of -100.

Column Name	**Data Type**	
Application ID	String	Required
Frequency Date	Date	Required
Scheduled Start Date	Date	Required
Scheduled Finish Date	Date	Required
Actual Start Date	Date	Required
Actual Finish Date	Date	Required

Table 4: SCHEDULE DATA NEEDED FOR MPI

SCHEDULE DATA

SCHEDULE FORMULA The formula for schedule is based upon past performance. Of the 422 projects scheduled since June 2013, only 45 projects had an estimated start date, estimated finish date and an actual finish date. Of those 45 projects; 20 finished exactly on time, 9 finished early and 16 finished late.

The first step of the formula is finding the percentage the schedules were missed for historical projects. The calculation treats over- and under-estimating the schedule the same. The same penalty is applied in both cases. For example, being 15% late will result in the same score as being 15% early. Perform this calculation

only for projects that did not exactly meet the estimated finish date.

$$\Delta_i = \left| \frac{F_{a_i} - F_{s_i}}{F_{s_i} - B_{s_i} + 1} \right|$$

Find the average of the Δ_i's. This is the average percent of a missed schedule.

$$\bar{\Delta} = \frac{\sum_{i=1}^{n} \Delta_i}{n}$$

The formula for schedule is then a percentage above or below the Δ. The number is calculated for each project, and then averaged to form the schedule score.

After the $\bar{\Delta}$ is calculated, the following formulas are used to create the schedule scores for each project and then the averaged schedule score.

$$S_{4_i} = \left\{ \begin{array}{ll} \text{where } \Delta_i \geq 1 & : -1 \times 100 \\ \text{where } \Delta_i \leq \bar{\Delta} & : \frac{\bar{\Delta} - \Delta_i}{\bar{\Delta}} \times 100 \\ \text{where } \Delta_i > \bar{\Delta} & : \frac{\bar{\Delta} - \Delta_i}{1 - \bar{\Delta}} \times 100 \end{array} \right] \text{, calculate schedule score for each project } i$$

$$S_4 = \frac{\sum_{i=1}^{n} S_{4_i}}{n} \text{ , average the schedule scores}$$

- n is the number of projects
- F_{a_i} the actual finish date of project i
- F_{s_i} the scheduled finish date of project i
- B_{s_i} the scheduled beginning date of project i
- Δ_i the percent the schedule was missed
- $\bar{\Delta}$ is the average percent RT misses schedules, ≈ 26
- S_{4_i} is the schedule score for project i

ALTERNATE APPROACH The best possible score should be achieved when meeting the estimated date exactly. The maximum score should come from the best estimate. Then given historical release data, it is easy to determine an average Δ between the actual and the estimated. Finishing a project within that Δ should result in a positive score. Outside the Δ results in negative scores. For example, a project releasing one day early or one day late would receive the same score because in both cases the estimate was missed by one day.

4.1.5 REQUIREMENTS

Column Name	**Data Type**	
Application ID	String	Required
Frequency Date	Date	Required
Requirements Scheduled	Integer	Required
Actual Requirements Released	Integer	Required

Table 5: REQUIREMENTS DATA NEEDED FOR MPI

REQUIREMENTS DATA

REQUIREMENTS FORMULA Requirements are desired new features or enhancements to a software product. It is important to know how many requirements were scheduled to be completed versus how many actually got completed.

$$S_5 = \frac{\sum_{i=1}^{n} (R_{a_i} - R_{e_i})}{n}$$

- R_a actual requirements completed for application i
- R_e expected requirements completed for application i

4.1.6 OVERALL MPI SCORE

One complete overall score cannot be successful without including information from various aspects of the software development organization. Therefore, the previous results are combined to provide one overall score.

$$MPI = \sum_{i=1}^{n} w_i S_i \text{ where } \sum_{i=1}^{n} w_i = 1$$

4.2 SENSITIVITY OF MPI

Some simulations will be run with data from each given distribution. Then the MPI scores will be analyzed. For more on sensitivity analysis in statistical modeling, see [66].

4.3 IMPORTANCE OF MPI

Earlier, in the introduction section 1.3.3, 3 main focus areas and 3 important questions for software analytics were presented. Table 6 provides a explanation of how MPI addresses each focus area. As can be seen, MPI clearly addresses the 3 main focus areas of software analytics. MPI does provide any mechanisms for improving the focus areas, but it provides a consistent mechanism to measure the focus areas.

Focus Area	Why MPI?
User Experience	One of the 5 elements of MPI is satisfaction. While not all of the questions focus solely on the user experience, the entire purpose of the survey is to determine if the user is satisfied with the software product. Does it have the correct features? Are new features added in a timely manner? Of course, specific survey questions can be created to focus solely on a certain user experience.
Quality	Again, one of the 5 elements specifically focuses on quality. MPI provides a single number to measure quality. Therefore, it is easy to track changes in quality over time. MPI does not address how to improve the quality, but without a consistent measurement, it would be impossible to determine the change in quality.
Development Productivity	The combination of MPI elements, schedule and requirements, provide an indication of development productivity. The schedule element measures the productivity related to estimated schedule. Similarly, the requirement element measures the amount of work actually being completed.

Table 6: SOFTWARE ANALYTICS FOCUS AREAS AND MPI

Question	Why MPI?
How much better is my model performing than a naive strategy, such as guessing?	MPI provides consistency which may not exist without it. Therefore, MPI removes the guesswork of measuring a software development organization.

Question	Why MPI?
How practically significant are the results?	The MPI score is consistent and easy to comprehend. Thus comparison with past performance is quick and simple. This is a significant advantage for software development organizations.
How sensitive are the results to small changes in one or more of the inputs?	The question was extensively addressed in section 4.2. It appears MPI is not overly sensitive to small changes in the inputs.

Table 7: IMPORTANT QUESTIONS FOR SOFTWARE ANALYTICS AND MPI

5 SDLC ANALYTIC ENGINE

In order for an SDO to properly track the elements of MPI, a data storage system should be available to store the appropriate data. A consistent storage system should help to avoid the problem of inaccurate data caused by numerous manipulations of the existing data [57]. Plus, if the system is implemented correctly by allowing limited changes to existing data, it will be able to alliviate some of the dishonesty that is currently present in software projects [61]. This storage system will be named the SDLC Analytic Engine (SDLC-AE). Figure 6 provides an overview of the data that could potentially be stored in the SDLC-AE.

Once all the SDLC data is collected into a single place, there are many possible applications. Software analytics will be much easier to create and gamification will be much more easily attainable.

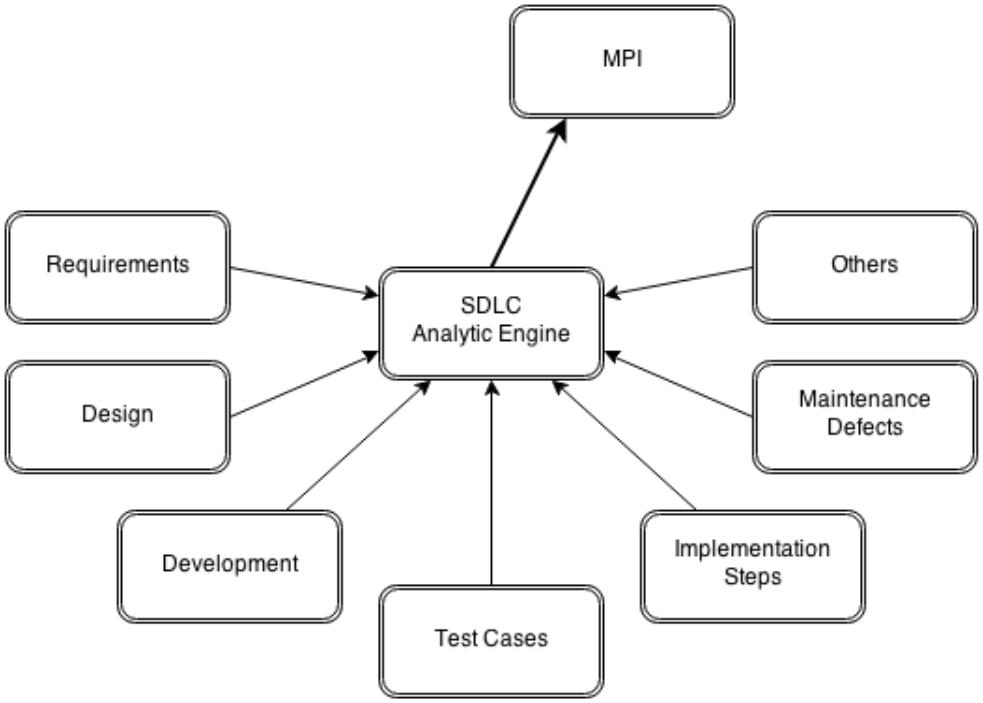

Figure 6: SDLC ANALYTIC ENGINE

5.1 DATABASE STRUCTURE

All the data necessary to compute MPI needs to be stored in a database. This section will lay out the structure of tables and relationships necessary to the store all the data within a relational database such as MySQL or PostgreSQL. Eventually, database diagrams and SQL scripts will be provided in the section.

5.2 QUERIES

The section will provide the necessary SQL queries to obtain the correct data needed to compute the individual elements of MPI.

5.3 COMPUTATIONS

This section will provide R scripts to calculate each individual score and the overall MPI score. The scripts will need to be run every time the data is updated and the MPI score needs to be recalculated.

5.4 ESTIMATION

- Change to Database Structure
- Modify Database Data
- Create a New Database, number of new databases
- Server Configuration Changes Required
- New Servers Required
- Number of Team Members Involved
- Number of (sub)Systems Involved
- Estimation Date
- Number of Days Allowed
- List of Other Attributes
- Number of Screens Involved
- Actual Value (hours, days, dollars)

Estimated Dev Hours - The number of development hours estimated for a project, this is just developer hours Estimated Doc Hours - The number of documentation hours estimated for a project Estimated Test Hours - The number of testing hours estimated for a project Estimated Deployment Hours - The number of estimated hours required to deploy the project

5.5 REQUIREMENTS

List of:

Title

Description

author

projectID

date

Comments list of

date

comment Text

author

5.6 MAINTENANCE (DEFECTS)

List of:

Project

Release

description

date Entered

Date fixed

Comments list of

date

comment Text

author

5.7 TESTING

List of:

- **Project**
- **Release**
- **title**
- **description**
- **author**
- **Date Started**
- **Date Executed**
- **Status (Pending, pass, fail)**
- **Comments** list of
  - **date**
  - **comment Text**
  - **author**

5.8 DEVELOPMENT

List of Coding Tasks:

- **Project**
- **Release**
- **List of Files**
- **author**

Date Started

Completion Date

Number of Unit Tests

Lines of Code

Others See [38, 39, 63, 67]

5.9 IMPLEMENTATION

List of:

Project

Release

date Entered

Date Scheduled

Date Executed

Ordering/Prerequisites

Comments list of

date

comment Text

author

6 CASE STUDY: SCORING AN SDO OF A LARGE FINANCIAL INSTITUTION

Data has been collected from the software development processes of an SDO within a large financial institution. The data collection was from 2008 to present. This section will provide the analysis performed and the MPI scores of that data. The data is incomplete but that is realistic of most oganizations. Currently, the data consists of availability, quality, and schedule. Satisfaction and Requirements are being obtained. An example of the initial analysis is included in appendix A.

6.1 WITHOUT HISTORICAL DATA

An MPI calculation with default values.

6.2 WITH HISTORICAL DATA

How to create better formulas with historical data.

7 CONCLUSION

There are many metrics that can be used to evaluate a Software Development Organization (SDO). Knowing which metrics to use and what they all mean can be a daunting task. MPI is a proposed solution to the difficulty of measuring an SDO.

Upon completion, this work shall identify:

- Define **What** characteristics should be measured for an SDO
- Define **How** to measure those characteristics
- Map the relationship or lack of relationship with a Balanced Scorecard
- Create a Framework to store the necessary data for MPI

- Outline a Process to generate the MPI score
- Provide an example MPI score with real data

An entire software development organization (SDO) needs to be measured and analyzed properly, not just the development portion. This document has provided an overview of what indicators need to be measured for an SDO, and how those indicators can be combined to form a single number indicating the performance score of a software development organization. Also, a framework to store this data was discussed.

APPENDIX

A CASE STUDY SOURCE CODE

A.1 R CODE - QUALITY

```
# Load the Quality data.
setAs("character","myDate", function(from) as.Date(from, format="%m/%d/%Y") )
setClass('myDate')

quality_data <- read.csv('quality_data_clean.csv',
        colClasses=c('factor','myDate','myDate','numeric','numeric',
        'numeric','numeric','numeric','numeric') )
str(quality_data)
head(quality_data)
summary(quality_data)

# Get data prior to 2013
old_quality_data = quality_data[quality_data$MONTH_DT <= as.Date('2013-12-31') ,]
str(old_quality_data)
pairs(old_quality_data[,c('TOTAL_TIX','EST_DEV_RES_HRS')] )

# Separate out large (>1000 hours) and small projects
large_quality_data = old_quality_data[old_quality_data$EST_DEV_RES_HRS >= 1000,]
str(large_quality_data)
largefit <- lm(TOTAL_TIX ~ EST_DEV_RES_HRS + DFTS_CNT_SIT + DFTS_CNT_UAT ,
        data=large_quality_data)
summary(largefit)
largefit_sigma = summary(largefit)$sigma
pairs(large_quality_data[,c('EST_DEV_RES_HRS','DFTS_CNT_SIT',
        'DFTS_CNT_UAT','TOTAL_TIX')])

small_quality_data =
        old_quality_data[old_quality_data$EST_DEV_RES_HRS < 1000
            && old_quality_data$APP != 'App␣86' ,]
str(small_quality_data)
smallfit <- lm(TOTAL_TIX ~ EST_DEV_RES_HRS + DFTS_CNT_SIT + DFTS_CNT_UAT ,
        data=small_quality_data)
summary(smallfit)
smallfit_sigma = summary(smallfit)$sigma
```

```
pairs(small_quality_data[,c('EST_DEV_RES_HRS','DFTS_CNT_SIT',
        'DFTS_CNT_UAT','TOTAL_TIX')])

# predict new data
new_large_quality_data =
        quality_data[quality_data$MONTH_DT > as.Date('2013-12-31')
            & quality_data$MONTH_DT < as.Date('2014-12-31')
            & quality_data$EST_DEV_RES_HRS >= 1000,]
new_small_quality_data =
        quality_data[quality_data$MONTH_DT > as.Date('2013-12-31')
        & quality_data$MONTH_DT < as.Date('2014-12-31')
        & quality_data$EST_DEV_RES_HRS < 1000,]
new_large_quality_data$PREDICTION =
        predict(largefit, newdata=new_large_quality_data)
new_large_quality_data$SCORE =
        (new_large_quality_data$PREDICTION - new_large_quality_data$TOTAL_TIX)
        / (6 * summary(largefit)$sigma)
new_large_quality_data[,c('APP','MONTH_DT','TOTAL_TIX','PREDICTION', 'SCORE')]

new_small_quality_data$PREDICTION = predict(smallfit, newdata=new_small_quality_data)
new_small_quality_data$SCORE =
        (new_small_quality_data$PREDICTION - new_small_quality_data$TOTAL_TIX)
        / (6 * summary(smallfit)$sigma)
new_small_quality_data[,c('APP','MONTH_DT','TOTAL_TIX','PREDICTION', 'SCORE')]

#Combine new data
new_quality_data = rbind(new_large_quality_data, new_small_quality_data)
str(new_quality_data)
summary(new_quality_data)

aggregate(SCORE ~ MONTH_DT, new_quality_data, mean)

# TODO include
# numerical summary stats, graphical summaries,
# dist of variables(hist used for guidance), associations among variables
```

A.2

REFERENCES

[1] IEEE Standard Glossary of Software Engineering Terminology. *IEEE Std 610.12-1990* (December 1990), 1–84.

[2] Andreessen, M. Why software is eating the world. *The Wall Street Journal* (August 2011). http://goo.gl/MXk4TS accessed 31-Mar-2014.

[3] Antolić, v. An example of using key performance indicators for software development process efficiency evaluation. In *MIPRO 2008: 31st International Convention on Information and Communication Technology, Electronics and Microelectronics, May 26-30, 2008, Opatija Croatia. Microelectronics, electronics and electronic technologies, MEET.. Grid and visualization systems, GVS* (2008), P. Biljanović, K. Skala, Grid, and V. Systems, Eds., vol. 1, MIPRO.

[4] Ayewah, N., Pugh, W., Morgenthaler, J. D., Penix, J., and Zhou, Y. Evaluating static analysis defect warnings on production software. In *Proceedings of the 7th ACM SIGPLAN-SIGSOFT Workshop on Program Analysis for Software Tools and Engineering* (New York, NY, USA, 2007), PASTE '07, ACM, pp. 1–8.

[5] Beck, K., Beedle, M., van Bennekum, A., Cockburn, A., Cunningham, W., Fowler, M., Grenning, J., Highsmith, J., Hunt, A., Jeffries, R., Kern, J., Marick, B., Martin, R. C., Mellor, S., Schwaber, K., Sutherland, J., and Thomas, D. Manifesto for agile software development, 2001.

[6] Benington, H. D. Production of large computer programs. In *Proceedings of the 9th International Conference on Software Engineering* (Los Alamitos, CA, USA, 1987), ICSE '87, IEEE Computer Society Press, pp. 299–310.

[7] Boehm, B. A spiral model of software development and enhancement. *SIGSOFT Software Engineering Notes 11*, 4 (August 1986), 14–24.

[8] Boehm, B. A spiral model of software development and enhancement. *Computer 21*, 5 (May 1988), 61–72.

[9] Boehm, B., and Basili, V. R. Software defect reduction top 10 list. *Computer 34*, 1 (January 2001), 135–137.

[10] Boehm, B., and Hansen, W. J. Spiral development: Experience, principles, and refinements. Tech. rep., Carnegie Mellon Software Engineering Institute, July 2000. Special Report CMU/SEI-2000-SR-008.

[11] Boehm, B. W. *Software Engineering Economics*, 1st ed. Prentice Hall PTR, Upper Saddle River, NJ, USA, 1981.

[12] Buse, R. P., and Zimmermann, T. Analytics for software development. In *Proceedings of the FSE/SDP Workshop on Future of Software Engineering Research* (New York, NY, USA, 2010), FoSER '10, ACM, pp. 77–80.

[13] Buse, R. P. L., and Zimmermann, T. Information needs for software development analytics. In *Proceedings of the 34th International Conference on Software Engineering* (Piscataway, NJ, USA, 2012), ICSE '12, IEEE Press, pp. 987–996.

[14] Cervone, D., D'Amour, A., Bornn, L., and Goldsberry, K. Pointwise: Predicting points and valuing decisions in real time with nba optical tracking data. MIT Sloan Sports Analytics Conference http://goo.gl/Rsbds1 accessed 26-Mar-2014, February 2014.

[15] Charette, R. Why software fails [software failure]. *IEEE Spectrum 42*, 9 (September 2005), 42–49.

[16] CMMI Product Team. CMMI for Development, Version 1.3. Tech. rep., Carnegie Mellon Software Engineering Institute (SEI), http://goo.gl/MBESq0, November 2010.

[17] Copeland, T. *PMD Applied: An Easy-to-use Guide for Developers*. An easy-to-use guide for developers. Centennial Books, 2005.

[18] Czerwonka, J., Nagappan, N., Schulte, W., and Murphy, B. Codemine: Building a software development data analytics platform at microsoft. *IEEE Software 30*, 4 (2013), 64–71.

[19] DeMarco, T. Software engineering: An idea whose time has come and gone? *IEEE Software 26*, 4 (July 2009), 96–96.

[20] Dubois, D. J., and Tamburrelli, G. Understanding gamification mechanisms for software development. In *Proceedings of the 2013 9th Joint Meeting on Foundations of Software Engineering* (2013), ACM, pp. 659–662.

[21] Ebert, C., Liedtke, T., and Baisch, E. Improving reliability of large software systems. *Annals of Software Engineering 8*, 1-4 (August 1999), 3–51.

[22] Emam, K. E., and Koru, A. G. A replicated survey of it software project failures. *IEEE Software 25*, 5 (September 2008), 84–90.

[23] Faizan, M., Khan, M. N. A., and Ulhaq, S. Contemporary trends in defect prevention: A survey report. *International Journal of Modern Education and Computer Science (IJMECS) 4*, 3 (2012), 14.

[24] Florac, W. A., and Carleton, A. D. *Measuring the Software Process*. Addison Wesley, Boston, 1999.

[25] Giardino, C., Unterkalmsteiner, M., Paternoster, N., Gorschek, T., and Abrahamsson, P. What do we know about software development in startups? *IEEE Software 31*, 5 (September 2014), 28–32.

[26] Godfrey, S. Characteristics of capability maturity model. via Wikimedia Commons http://goo.gl/MpNx9b accessed 12-22-2014, October 2011.

[27] Goel, A. L., and Shin, M. Software engineering data analysis techniques (tutorial). In *Proceedings of the 19th International Conference on Software Engineering* (New York, NY, USA, 1997), ICSE '97, ACM, pp. 667–668.

[28] Halkidi, M., Spinellis, D., Tsatsaronis, G., and Vazirgiannis, M. Data mining in software engineering. *Intelligent Data Analysis 15*, 3 (August 2011), 413–441.

[29] Halstead, M. H. *Elements of Software Science (Operating and Programming Systems Series)*. Elsevier Science Inc., New York, NY, USA, 1977.

[30] Hassan, A. E., Hindle, A., Runeson, P., Shepperd, M., Devanbu, P., and Kim, S. Roundtable: What's next in software analytics. *IEEE Software 30*, 4 (July 2013), 53–56.

[31] Hassan, A. E., and Xie, T. Software intelligence: The future of mining software engineering data. In *Proceedings of the FSE/SDP Workshop on Future of Software Engineering Research* (New York, NY, USA, 2010), FoSER '10, ACM, pp. 161–166.

[32] Hibbs, C., Jewett, S., and Sullivan, M. *The Art of Lean Software Development: A Practical and Incremental Approach*, 1st ed. O'Reilly Media, Inc., 2009.

[33] Hubbard, D. *How to Measure Anything: Finding the Value of Intangibles in Business*. Wiley, 2010.

[34] Ichu, E. A. *The Role of Quality Assurance in Software Development Projects: Software Project Failures and Business Performance*. LAP Lambert Academic Publishing, Germany, 2012.

[35] Jacobson, I., and Seidewitz, E. A new software engineering. *Communications of the ACM 57*, 12 (November 2014), 49–54.

[36] Jain, A., and Angadi, S. Gamifiying software development process. *Infosys Labs Briefings 11*, 3 (2013), 21–28. http://goo.gl/T9PB96 accessed 01-Jan-2015.

[37] Jones, C. *Applied Software Measurement: Assuring Productivity and Quality*, 2nd ed. McGraw-Hill, Inc., Hightstown, NJ, USA, 1997.

[38] Jones, C. *Software Engineering Best Practices*, 1 ed. McGraw-Hill, Inc., New York, NY, USA, 2010.

[39] Jones, C. Scoring and evaluating software methods, practices, and results. http://goo.gl/3i06pN, June 2012. Namecook Analytics Blog, accessed 22-Mar-2014.

[40] Jones, C. *The Technical and Social History of Software Engineering*, 1st ed. Addison-Wesley Professional, 2013, ch. 10.

[41] Jones, C. WHY "COST PER DEFECT" IS HARMFUL FOR SOFTWARE QUALITY. http://goo.gl/QUsvlw, July 2013. Namecook Analytics Blog, accessed 01-Jan-2015.

[42] Jørgensen, M. A strong focus on low price when selecting software providers increases the likelihood of failure in software outsourcing projects. In

Proceedings of the 17th International Conference on Evaluation and Assessment in Software Engineering (New York, NY, USA, 2013), EASE '13, ACM, pp. 220–227.

[43] Jorgensen, M. What we do and don't know about software development effort estimation. *IEEE Software 31*, 2 (March 2014), 37–40.

[44] Kaner, C., and Bond, W. P. Software engineering metrics: What do they measure and how do we know? In *METRICS 2004* (2004), IEEE CS Press.

[45] Kaplan, R. S., and Norton, D. P. The balanced scorecard: Measures that drive performance. *Harvard Business Review* (January-February 1992), 71–80.

[46] Lee, J. *Software Engineering with Computational Intelligence*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2003.

[47] Lehtinen, T. O. A., Mäntylä, M. V., Vanhanen, J., Itkonen, J., and Lassenius, C. Perceived causes of software project failures - an analysis of their relationships. *Journal Information and Software Technology 56*, 6 (June 2014), 623–643.

[48] Letier, E., and Fitzgerald, C. Measure what counts: An evaluation pattern for software data analysis. In *2013 1st International Workshop on Data Analysis Patterns in Software Engineering (DAPSE)* (2013), IEEE, pp. 20–22.

[49] Maheshwari, S., and Jain, D. C. A comparative analysis of different types of models in software development life cycle. *International Journal of Advanced Research in Computer Science and Software Engineering 2*, 5 (May 2012), 285–290.

[50] Marcus, A., and Menzies, T. Software is data too. In *Proceedings of the FSE/SDP Workshop on Future of Software Engineering Research* (New York, NY, USA, 2010), FoSER '10, ACM, pp. 229–232.

[51] McCabe, T. A complexity measure. *IEEE Transactions on Software Engineering SE-2*, 4 (December 1976), 308–320.

[52] Menzies, T., Caglayan, B., He, Z., Kocaguneli, E., Krall, J., Peters, F., and Turhan, B. The promise repository of empirical software engineering data, June 2012.

[53] Menzies, T., and Zimmermann, T. Goldfish bowl panel: Software development analytics. In *2012 34th International Conference on Software Engineering (ICSE)* (June 2012), pp. 1032–1033.

[54] Miguel, J. P., Mauricio, D., and Rodríguez, G. A review of software quality models for the evaluation of software products. *International Journal of Software Engineering & Applications (IJSEA) 5*, 6 (November 2014), 31–54. http://www.airccse.org/journal/ijsea/papers/5614ijsea03.pdf.

[55] Moniruzzaman, A. B. M., and Hossain, S. A. Comparative study on agile software development methodologies. *Global Journal of Computer Science and Technology 13*, 7 (2013).

[56] Naur, P., and Randell, B., Eds. *Software Engineering: Report of a Conference Sponsored by the NATO Science Committee, Garmisch, Germany, 7-11 Oct. 1968, Brussels, Scientific Affairs Division, NATO.* 1969. http://homepages.cs.ncl.ac.uk/brian.randell/NATO/nato1968.PDF.

[57] Olson, J. *Data Quality: The Accuracy Dimension.* The Morgan Kaufmann Series in Data Management Systems. Elsevier Science, 2003.

[58] Parmenter, D. *Key Performance Indicators: developing, implementing, and using winning KPIs*. John Wiley and Sons, Inc., Hoboken, New Jersey, 2010.

[59] Putnam, L. H., and Myers, W. *Five Core Metrics: the Intelligence Behind Successful Software Management*. Addison-Wesley Professional, New York, New York, 2013.

[60] Ramler, R., and Wolfmaier, K. Economic perspectives in test automation: Balancing automated and manual testing with opportunity cost. In *Proceedings of the 2006 International Workshop on Automation of Software Test* (New York, NY, USA, 2006), AST '06, ACM, pp. 85–91.

[61] Rost, J., and Glass, R. *The Dark Side of Software Engineering: Evil on Computing Projects*. Wiley, 2011, ch. 2 Lying.

[62] Royce, W. W. Managing the development of large software systems: Concepts and techniques. In *Proceedings of the 9th International Conference on Software Engineering* (Los Alamitos, CA, USA, 1987), ICSE '87, IEEE Computer Society Press, pp. 328–338.

[63] Rubin, V., Günther, C. W., Van Der Aalst, W. M. P., Kindler, E., Van Dongen, B. F., and Schäfer, W. Process mining framework for software processes. In *Proceedings of the 2007 International Conference on Software Process* (Berlin, Heidelberg, 2007), ICSP'07, Springer-Verlag, pp. 169–181.

[64] Ruhe, G., and Gesellschaft, F. Knowledge discovery from software engineering data: Rough set analysis and its interaction with goal-oriented measurement. In *Principles of Data Mining and Knowledge Discovery*, J. Komorowski and J. Zytkow, Eds., vol. 1263 of *Lecture Notes in Computer Science*. Springer Berlin Heidelberg, 1997, pp. 167–177.

[65] Ruparelia, N. B. Software development lifecycle models. *SIGSOFT Softw. Eng. Notes 35*, 3 (May 2010), 8–13.

[66] Saltelli, A., Tarantola, S., and Campolongo, F. Sensitivity analysis as an ingredient of modeling. *Statistical Software 15*, 4 (2000), 377–395.

[67] Snipes, W. B. Evaluating developer responses to gamification of software development practices. Master's thesis, North Carolina State University, 2013.

[68] Sommerville, I. *Software Engineering*, 6 ed. Addison-Wesley, Harlow, England, 2001.

[69] Spraragen, S. L. The challenges in creating tools for improving the software development lifecycle. In *Proceedings of the 2005 Workshop on Human and Social Factors of Software Engineering* (New York, NY, USA, 2005), HSSE '05, ACM, pp. 1–3.

[70] Taylor, Q., and Giraud-Carrier, C. Applications of data mining in software engineering. *International Journal of Data Analysis Techniques and Strategies 2*, 3 (July 2010), 243–257.

[71] Tosi, D., Lavazza, L., Morasca, S., and Taibi, D. On the definition of dynamic software measures. In *Proceedings of the ACM-IEEE International Symposium on Empirical Software Engineering and Measurement* (New York, NY, USA, 2012), ESEM '12, ACM, pp. 39–48.

[72] Tsui, F. F. *Essentials of software engineering*, 3rd ed. Jones & Bartlett Publishers, 2013.

[73] van der Aalst, W. *Process Mining : Discovery, Conformance and Enhancement of Business Processes*. Springer, Heidelberg, 2011.

[74] van Genuchten, M., Mans, R., Reijers, H., and Wismeijer, D. Is your upgrade worth it? process mining can tell. *IEEE Software 31*, 5 (September 2014), 94–100.

[75] Werbach, K. (re)defining gamification: A process approach. In *Persuasive Technology*, A. Spagnolli, L. Chittaro, and L. Gamberini, Eds., vol. 8462 of *Lecture Notes in Computer Science*. Springer International Publishing, 2014, pp. 266–272.

[76] Winter, V., Reinke, C., and Guerrero, J. Sextant: A tool to specify and visualize software metrics for java source-code. In *Emerging Trends in Software Metrics (WETSoM), 2013 4th International Workshop on* (May 2013), pp. 49–55.

[77] Xie, T., Thummalapenta, S., Lo, D., and Liu, C. Data mining for software engineering. *Computer 42*, 8 (2009), 55–62.

[78] Zhang, D., Dang, Y., Lou, J.-G., Han, S., Zhang, H., and Xie, T. Software analytics as a learning case in practice: Approaches and experiences. In *Proceedings of the International Workshop on Machine Learning Technologies in Software Engineering* (New York, NY, USA, 2011), MALETS '11, ACM, pp. 55–58.

[79] Zhang, D., Han, S., Dang, Y., Lou, J.-G., Zhang, H., and Xie, T. Software analytics in practice. *IEEE Software 30*, 5 (September 2013), 30–37.