ENTERPRISE SOFTWARE METRICS: HOW TO ADD BUSINESS VALUE

A thesis submitted
to Kent State University in partial
fulfillment of the requirements for the
degree of Masters of Science

by
Binamra Dutta
May 2009
Thesis written by
Binamra Dutta
B.E, Amravati University, 2001
M.S, Kent State University, 2009

Approved by

Dr. Austin Melton, Advisor

Dr. Robert A. Walker, Chair, Department of Computer Science

Dr. John R. D. Stalvey, Dean, College of Arts and Sciences
TABLE OF CONTENTS

LIST OF FIGURES ...................................................................................................................... vi
LIST OF TABLES ....................................................................................................................... vii
CHAPTER 1 .................................................................................................................................. 1
INTRODUCTION ........................................................................................................................... 1
  1.1. Overview .......................................................................................................................... 1
  1.2. Thesis Contribution ......................................................................................................... 3
  1.3. Thesis Organization ......................................................................................................... 6
CHAPTER 2 .................................................................................................................................. 7
BENEFITS OF METRICS-DRIVEN SOFTWARE DEVELOPMENT ............................................. 7
  2.1. Project Estimation and Progress Monitoring .................................................................. 7
  2.2. Evaluation of Work Products .......................................................................................... 8
  2.3. Process Improvement through Software Quality Analysis .......................................... 8
  2.4. Experimental Validation of best practices ...................................................................... 8
CHAPTER 3 .................................................................................................................................. 10
SOFTWARE MEASUREMENT FOUNDATION OVERVIEW ................................................ 10
  3.1 ISO 9126 Standard ........................................................................................................... 10
    3.1.1 Functionality .............................................................................................................. 10
    3.1.2 Reliability .................................................................................................................. 11
    3.1.3 Usability .................................................................................................................... 11
    3.1.4 Efficiency .................................................................................................................. 11
    3.1.5 Maintainability ......................................................................................................... 11
    3.1.6 Portability ................................................................................................................. 12
  3.2 Measurement Theory ......................................................................................................... 12
    3.2.1 Nominal Scale ........................................................................................................... 13
    3.2.2 Ordinal Scale ............................................................................................................. 13
    3.2.3 Interval Scale ............................................................................................................ 13
    3.2.4 Ratio Scale ................................................................................................................. 14
    3.2.5 Absolute Scale ......................................................................................................... 14
CHAPTER 4 .................................................................................................................................. 15
AGILE DEVELOPMENT METHODOLOGY ........................................................................... 15
  4.1 Software Development Methods ....................................................................................... 16
    4.1.1 Agile methods: adaptive rather than predictive ......................................................... 17
    4.1.2 Agile methods: people-oriented rather than process-oriented .................................. 17
LIST OF FIGURES

FIGURE 1: GQM Paradigm .......................................................... 39
FIGURE 2: Checklist for Measuring Software Defects .................. 51
FIGURE 3: Cumulative Flow Diagram for ‘Features Completed vs Features Remaining’ .......................................................... 55
FIGURE 4: Employee Performance Assessment .......................... 71
LIST OF TABLES

TABLE 1: Metric Categories vs. Data Collected.................................................24
TABLE 2: Entities and Attributes for Example Question 1.................................45
TABLE 3: Entities and Attributes for Example Question 2................................45
TABLE 4: Entities and Attributes for Example Question 3................................46
TABLE 5: Data Ownership Examples..............................................................57
CHAPTER 1

INTRODUCTION

1.1 Overview

Metrology, the ‘science of measurement’ has a long history. It has advanced for some sciences such as chemistry and physics for over 200 years. Since information technology and computer science are fairly new sciences, they have not undergone the same level of metrological scrutiny as other sciences. Because software engineering is such a new discipline, we are struggling not only with the question of what to measure but also how to measure. A widespread definition used in the field is that measurement is ‘a process whereby numbers or symbols are assigned to an attribute of entities in such a manner as to describe the attribute in a meaningful way’ [1]. An entity may be a thing or an event, i.e. a person, a play, a developed program or the development process. An attribute is a trait of the entity, such as the height of a person, the cost of a play, or the length of the development process. In general, the amount of an attribute is what we measure. Software measures are used to measure specific attributes of a software product, process or resource during the software development process. Software metrics are used for determining if we have achieved desired state of quality assurance, for deriving project estimates and timelines, to analyze defects, to determine complexity and to validate the best practices being followed. In short, software metrics help us in making better decisions.
In recent years, software measurement has evolved into a key software engineering discipline. It was common practice in the industry to look at measurement initiatives as ‘extra work’ and something that can only be afforded by bigger companies. In the last few decades however, things have changed quite a bit and measurement is now considered to be a basic software engineering practice, as evidenced by its inclusion in the Level 2 maturity requirements of the Software Engineering Institute’s Capability Maturity Model Integration (CMMI) products and related commercial software process standards [9]. Although there are many generic claimed benefits of software metrics [2] [3] [4] [5], we support the ideology that the most significant of these benefits is that they are supposed to provide information to support quantitative managerial decision-making during the software lifecycle [6].

Traditional metric approaches, often driven by regression-based models for cost estimation and defects prediction, provide little support for managers wishing to use software metrics to analyze and minimize risk [29]. We believe that the future for software metrics lies in using relatively simple goal oriented metrics better suited to build management decision-support tools that combine different aspects of software development and software quality assurance techniques, enabling managers to make different kinds of predictions, assessments and trade-offs during the software life-cycle.

In the present day globally competitive, knowledge based, economic paradigm it is the people working in an organization that are its most important asset. As the economy has transitioned from being physical production oriented to service provision oriented it has become important to develop human capital performance metrics that not only judge individual performance, but also correlate it with the achievement of broader corporate objectives. In current times, human capital has become more important than physical capital that has dominated the industry in the past. As there is no one trait to assess an individual’s performance, a lot of factors
like job specific skills, individual goals, teamwork skills, organizational compatibility, motivation and future growth potential have to be kept in mind while developing a performance assessment. One of the major industries where assessment techniques have been greatly used is the education industry. The general idea behind standardized assessment techniques in schools was to make the schools accountable to the public, as well as to develop a standard for assessing a student’s knowledge and skill. We believe that the challenges in developing and applying human capital performance assessment techniques in corporate industry are in a lot of aspects similar to the challenges of applying ‘Assessment for Learning’ techniques in education industry.

1.2 Thesis Contribution

Although there is no consensus on what metrics work best for a specific organization, it is still a widely accepted belief that software metrics can help organizations in achieving better productivity. In the course of developing software, it is common for software teams to focus more on the deliverables. For them to be sure that they are on the right path of achieving their business goal, some kind of measurement program is needed to measure and monitor the progress correctly and efficiently. A well planned measurement program tailored for a specific organization can help provide specific information for managing the multitude of concurrent projects, tracking roadblocks, and suggesting improvements in the software development process thereby leading to fulfillment of the short term as well as long term project and business goals.

In this research we have tried to collect and review the existing methodologies for establishing a successful metrics program in software development industry. Due to the various models of software development being followed in the industry, it is difficult to propose best
practices for all the approaches in the scope of this research. We have reviewed existing information for establishing a metrics program in an organization [15] [18] [19] [21] [25] and propose ideas to further refine them for step by step practical application in organizations which follow agile development practices. We also review some of the measurement techniques that have been successfully implemented in the past in Motorola [20] and can be of help to any team working on designing a metrics program.

Assessment techniques have also been used successfully in various fields in the past. Assessment techniques in schools and standardized testing techniques in academics are one of the very popular ones. We believe that the human anxiety that accompanies any new metrics program being instituted in an organization is similar to anxiety connected with classroom assessment. Traditionally, schools have used grade assessments with the belief that maximizing anxiety will fuel growth in learning. In recent years a shift has occurred in schools to motivate students to become more competent and using assessment as a power tool as opposed to just a ‘ranking scale’. To better understand the people management aspect of measurement and the motivational intricacies involved, we have outlined the efforts made in classroom assessment in the past five decades to connect assessment with learning improvement. We believe that some of those techniques can be successfully applied for motivating people and creating a positive measurement culture within an organization.

This research is intended to be used by a software engineering process group of an organization or a working group tasked to implement a measurement program in their organization. But, because measurement is used to help quantify the software process for making decisions, virtually anyone involved with software at various levels in an organization can benefit from this research.

Our objectives for this thesis are:
• Review and discuss the current usage of measurement theory in software engineering.
• Outline some steps to help an organization to start and sustain a measurement program.
• Provide guidelines that can be used to design and implement a measurement process within an organization that:
  - ties measurement to organizational goals and objectives
  - provides support for managerial decision making
  - defines measurement consistently, clearly and accurately
  - collects and analyzes data to measure progress towards goals
  - evolves and improves as the process matures
• Review and discuss the recent paradigm shift in academic assessment techniques and how software metrics can benefit from those
• Review and discuss successful metrics programs in the industry.

Software has been developed since the 1950’s, and different methods, paradigms and process models have been invented to handle the complexity of software development. Some of the development methods have become heavily document oriented or expect the developers to rigorously follow certain processes. These are generally referred to as heavy or traditional software development methods, for example the waterfall model of development [22], and more recently the spiral model or the Rational Unified Process [23]. In the turn of the millennium, new software development ideas were presented in the form of the Agile Manifesto as a counteraction to rigorous, plan-driven software development [24]. Agile methods are less document-oriented than traditional methods. Instead, agile methods are more code-oriented and they emphasize working code over documentation. Besides, agile methods are more people-oriented that process-oriented [31]. Keeping in mind the wide adoption of Agile Development methodology in the
industry, we make agile development the chosen methodology for application of the suggested guidelines for enterprise metrics. The thesis has been structured as follows.

1.3 Thesis Organization

In chapter two, we discuss the most significant uses of metrics-driven software development. In chapter three, we present an overview of ISO 9126 international standard for the evaluation of software quality and measurement theory so that the reader who is not acquainted with it can understand the remainder of the thesis. In chapter four, we outline the basic ideas behind agile development. In chapter five, we discuss the traditional software metrics approaches still popular in the industry and their need for extension. In chapter six, we propose best practices and implementation strategies for applying software metrics during agile development. In chapter seven, we analyze and discuss the ‘assessment for learning’ aspect of school assessment, how progress has been made by using various initiatives in the last five decades and how some of those techniques can be successfully applied in the industry to motivate employees, resulting in increased individual and organizational performance. In chapter eight, we discuss one of the few well documented successful metrics program in the industry.
Measurement is introduced by information technology organizations to better understand, evaluate, control and predict their decision making and software processes. There are many practical uses of software metrics. Four of the more significant ones that have been explored by both academics and industry are as follows:

2.1 Project Estimation and Progress Monitoring

There has been a lot of work on estimation, with limited success. Today, the software industry knows that there are many aspects of software development that influence estimates. The recent emphasis on process improvement gives us some pointers here, for much of our estimation inaccuracy is a result of long development cycle times and process variability. It is not unusual for the project requirements to change after inception of the project. Managers should be responsible to put adequate systems in place to measure project progress. They need to use metrics data procured from previous projects for project estimation and efficiently handle process and requirement variability.
2.2 Evaluation of Work Products

Analysis of code is the single largest area of software metrics research. All software development ultimately results in a deliverable of software code. Code is expressed in an industry standard language that can be processed automatically. Each language is well defined, and code is the complete form of the desired result, unlike any intermediate work product. This makes the end product - software code - a convenient vehicle to apply software metrics.

2.3 Process Improvement through Software Quality Analysis

Software quality is one of the most promising and widely sought after areas of today’s development processes. Defect analysis, tracking and removal offers a great potential for companies to cut down their development and maintenance costs. Software Quality covers a host of attributes: integrity, interoperability, flexibility, maintainability, portability, expendability, reusability, resilience and usability. These attributes should be kept in mind while designing and implementing the software development process. Then their achievement and adherence may be confirmed using defect analysis metrics and quality assurance tests.

2.4 Experimental Validation of Software Development Best Practices

Software Engineering is a rapidly changing field. It is critical for companies to continue to use metrics data to examine the sometimes unrealistic claims that are made for new software development methods and tools in software engineering. Software professionals and researchers
have validated some important engineering practices using software metrics leading to quicker, widespread acceptance of these practices by industry.
CHAPTER 3

SOFTWARE MEASUREMENT FOUNDATION OVERVIEW

3.1 ISO 9126 Standard

We start by looking at the ISO 9126 standard [32]. The standard is divided into four parts which address, respectively, the following subjects: quality model; external metrics; internal metrics; and quality in use metrics.

The quality model established in the first part of the standard, ISO 9126-1, classifies software quality in a structured set of characteristics and sub-characteristics as follows:

3.1.1 Functionality - A set of attributes that bear on the existence of a set of functions and their specified properties. The functions are those that satisfy stated or implied needs.

- Suitability
- Accuracy
- Interoperability
- Compliance
- Security
3.1.2 Reliability - A set of attributes that bear on the capability of software to maintain its level of performance under stated conditions for a stated period of time.

- Maturity
- Recoverability
- Fault Tolerance

3.1.3 Usability - A set of attributes that bear on the effort needed for use, and on the individual assessment of such use, by a stated or implied set of users.

- Learnability
- Understandability
- Operability

3.1.4 Efficiency - A set of attributes that bear on the relationship between the level of performance of the software and the amount of resources used, under stated conditions.

- Time Behavior
- Resource Behavior

3.1.5 Maintainability - A set of attributes that bear on the effort needed to make specified modifications.

- Stability
- Analyzability
- Changeability
- Testability
3.1.6 Portability - *A set of attributes that bear on the ability of software to be transferred from one environment to another.*

- Installability
- Replaceability
- Adaptability

Each quality sub-characteristic (as adaptability) can be further divided into more product specific attributes. An attribute is a property which can be verified or measured in the software product. Attributes are not defined in the standard, as they vary between different software products. The challenge for software measurement is to characterize the qualitative attributes of the ISO 9126 characteristics with software measures. Another major challenge is the problem of uniqueness which pertains to: How unique is the resulting measure or scale? For example the complexity of a software program can be measured by dozens of different measures. Adequate thought needs to be given on the kind of comparisons and mathematical manipulations which are possible with a new measure when compared to other measures.

3.2 Measurement Theory

Measurement is the process whereby numbers or symbols are assigned to attributes of entities in such a manner as to describe the attribute in a meaningful way. A particular way of assigning numbers or symbols to measure something is called a *Scale* of measurement. The scales of measurement can be transformed using *Permissible Transformations* that preserve the relevant relationships of the measurement process. The scale types are distinguished into five main types with each having its own set of permissible transformations.
3.2.1 Nominal Scale
- Two entities are assigned the same value if they have the same value of the attribute.
- No meaningful relationship between instances of the attribute.
- Information is more qualitative than quantitative.
- Examples: numbering of football players, numbering of television channels, numbering of race cars etc.

3.2.2 Ordinal Scale
- Entities are assigned numbers such that the order of the numbers reflects an order relation applicable on the attribute. Two entities x and y with attribute values \( a(x) \) and \( a(y) \) are assigned numbers \( m(x) \) and \( m(y) \) such that if \( m(x) > m(y) \) then \( a(x) > a(y) \)
- Intervals between numbers may not be equal.
- Examples: Numbers assigned for grading of gasoline, grades assigned for academic performance.

3.2.3 Interval Scale
- Entities are assigned numbers such that the differences between numbers reflect differences between the attribute being measured. Four entities x, y, u and v with attribute values \( a(x), a(y), a(u) \) and \( a(v) \) are assigned numbers \( m(x), m(y), m(u) \) and \( m(v) \) such that if \( m(x) - m(y) > m(u) - m(v) \) then \( a(x) - a(y) > a(u) - a(v) \)
- Permissible transformations can be of the type \( g(x) = ax + b \)
- Examples: temperature in degrees Fahrenheit or Celsius, calendar date.
3.2.4 Ratio Scale

- Entities are assigned numbers such that the differences and the ratios between the numbers reflect differences and ratios of the attributes.
- Permissible transformations can be of the type \( g(x) = ax, a > 0 \) which implies that the unit of measurement may be arbitrary.
- Examples: Length in centimeters, duration in seconds, temperature in degrees Kelvin.

3.2.5 Absolute Scale

- Entities are assigned numbers such that properties of the numbers are analogous to the attribute.
- Needs exactly one measure.
- The only permissible transformation is the identity transformation \( g(x) = ax, a > 0 \).
- Example: number of children in a family.

These measurement scales mentioned above form a linear order from being weaker to stronger on the sets of permissible transformations. In real life a scale of measurement may not conform precisely to any of these scales. User questionnaires can be an example where there are several non response categories making it conform to both the nominal and the ordinal scales.
Agile methods are relatively new, light weighted software development methods and processes, which attempt to reduce the bureaucracy of the software development process and to minimize the time from requirement commitment to delivery. Agile methods can be looked as a compromise between using no process in software development and using a strict and usually very document oriented development process.

In recent years a number of processes claiming to be ‘agile’ have been proposed in the literature. To avoid confusion over what it means for a process to be ‘agile’, seventeen agile process methodologists came to an agreement on what ‘agility’ means during a 2001 meeting where they discussed future trends in software development processes. One result of the meeting was the formation of the ‘Agile Alliance’ and the publication of the Agile Manifesto [24]. The manifesto of the ‘Agile Alliance’ is a condensed definition of the values and goals of ‘Agile Software Development’. This manifesto is detailed and contains a number of common principles for agile processes. The principles are listed below.

- “Our highest priority is to satisfy the customer through early and continuous delivery of valuable software.”
- “Welcome changing requirements, even late in development.”
- “Deliver working software frequently.”
• “Business people and developers must work together daily throughout the project.”
• “Build projects around motivated individuals. Give them the environment and support they need, and trust them to get the job done.”
• “The most efficient and effective method of conveying information to and within a development team is face-to-face conversation.”
• “Working software is the primary measure of progress.”
• “Agile processes promote sustainable development. The sponsors, developers and users should be able to maintain a constant pace indefinitely.”
• “Continuous attention to technical excellence and good design enhances agility.”
• “Simplicity – the art of maximizing the amount of work not done - is essential.”
• “The best architectures, requirements, and designs emerge from self-organizing teams.”
• “At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly.”

4.1 Software Development Methods

Software development methods are used in general to efficiently manage and organize the ever growing code base within an organization. Over the years various software development methodologies have been introduced which are based on similar ideas. The drive behind these various engineering methods is to make a good development plan and control the day to day work. Using these methodologies makes both the development process and the outcome more predictable. Agile methods differ from the other software development methods in two ways
4.1.1 Agile methods: adaptive rather than predictive

The roadmaps used by various engineering methods do not adapt well to changing requirements. In most development methods their nature is to resist change. The agile methods however, welcome change. In agile methods long term plans are very fluid, and the only stable plans are short term plan that are made for a single iteration. Iterative development gives a firm foundation in each iteration that we can base our later plans around [31].

4.1.2 Agile methods: people-oriented rather than process-oriented

Across all agile development methodologies, the one common factor is the importance assigned to people and the team work needed to complete a project. Brooks [17] acknowledges the same

“The quality of people on a project, and their organization and management, are more important factors in success than are the tools they use or the technical approaches they take.”

4.2 Flavors of Agile Development

The term ‘agile’ is more like a philosophy of software development. Various approaches like Extreme Programming, Scrum, Lean Development etc all fall under Agile Development Methodologies. Each of these approaches has its own leaders and followers forming a community of its own. The one common factor among all these approaches is the adherence to the Agile Manifesto [24].
4.2.1 Extreme Programming (XP)

Extreme Programming begins with five values (Communication, Feedback, Simplicity, Courage and Respect). It then elaborates these into fourteen principles which are again divided into twenty four practices. The idea is that practices are concrete things that a team can do day-to-day, while values are the fundamental knowledge and understanding that underpin the approach. XP puts a very strong emphasis on software testing. While all processes mention software testing, most do so with a pretty low emphasis. However, XP puts testing at the foundation of development, with every programmer writing tests as they write their production code [31]. Continuous integration processes are then used to yield a stable code base. For this reason XP is also referred to as Test Driven Development.

4.2.2 Scrum

Scrum was developed in the 80’s and 90’s primarily with Object Oriented circles as a highly iterative development methodology. The term comes from the game ‘Rugby’ in which eight players from one team interlock with members of the other team to win possession of the ball. Scrum concentrates on the management aspects of software development, dividing development into thirty day iterations (called ‘sprints’) and applying closer monitoring and control with daily scrum meetings. It places much less emphasis on engineering practices and many teams usually combine its project management approach with Extreme Programming engineering practices [31].
4.2.3 Crystal

The Crystal family of software development methods is a group of approaches tailored to different size teams. Despite their variations, all crystal approaches share common features. All crystal methods have three priorities: project safety, development efficiency and process habitability (team communication, feedback etc). They also share common properties, of which the three most important are: Frequent Delivery, Reflective Improvement and Close Communication. The process habitability priority is an important part of the crystal mind-set. Crystal methods aim for less discipline than extreme programming, trading off less efficiency for greater process habitability and reducing chances of failure [31].

4.2.4 Rational Unified Process (RUP)

Another well known process to have come out of the object-oriented community is the Rational Unified Process, sometimes just referred to as the Unified Process. The original idea was that like the unified modeling languages the rational unified process could unify software processes. RUP is a very large collection of practices and is really a process framework rather than a process. Rather than give a single process for software development, it seeks to provide a common set of practices for teams to choose from for an individual project. As a result, for a team using RUP the first step is to define their individual processes, or in RUP terms a, ‘development case’. The key common to aspects of RUP is that it is Use Case Driven (development is driven through user-visible features), iterative, and architecture centric (there’s a priority to building a architecture early on that will last the project through) [31].
Despite the different flavors of agile development there are some principles that are common for most of them

- Development is done in iterations
- Each iteration delivers a version with more customer value than the previous release
- The development is planned at a high level in the beginning, and then the plan is updated for each iteration
- Interaction with the customer is continuous.
CHAPTER 5

METRICS IN INDUSTRY: DISCUSSION, NEED FOR EXTENSION, GUIDELINES

Software Metrics is used as a collective term to define all the different activities pertaining to measurement in software engineering. The field of software measurement provides approaches for measurement of specific attributes of a software product. Software metrics are used for various reasons in the industry like deriving project estimates, determining program complexity, determining software quality, defect analysis or to experimentally validate software development best practices. Although software metrics have been in use from the 1960’s when Lines of Code metric was first introduced as a quantitative way to measure programmer productivity, we believe that there is still widespread confusion regarding definition of software metrics and how best to apply them in industry. we agree with the statement [8].

“What theory is doing with respect to measurement of software work and what practice is doing are on two different planes, planes that are shifting in different directions”

The recent industrial software metrics activity doesn’t necessarily have to do with companies realizing the ‘value-added’ proposition of software metrics. Most of the times software metric programs are introduced with the sole purpose of identifying the roadblocks within specific projects. The other most common use of instituting software metrics programs in companies is for them to acquire higher levels of CMM [9] since evidence of metrics usage is intrinsic to attaining the same.
There are large numbers of different types of metrics that are used currently in the software industry for the software development process. During the study we found that a lot of these metrics still fall under categories given by McConnell [7].
<table>
<thead>
<tr>
<th>Metrics Category</th>
<th>Data Collected by Metric</th>
</tr>
</thead>
<tbody>
<tr>
<td>Size Metrics</td>
<td>- Total lines of code written</td>
</tr>
<tr>
<td></td>
<td>- Total comment lines</td>
</tr>
<tr>
<td></td>
<td>- Total data declarations</td>
</tr>
<tr>
<td></td>
<td>- Total blank lines</td>
</tr>
<tr>
<td>Productivity Metrics</td>
<td>- Work-hours spent on the project,</td>
</tr>
<tr>
<td></td>
<td>- Work-hours spent on each routine changed</td>
</tr>
<tr>
<td></td>
<td>- Dollars spent on project</td>
</tr>
<tr>
<td></td>
<td>- Dollars spent per line of code</td>
</tr>
<tr>
<td></td>
<td>- Dollars spent per defect</td>
</tr>
<tr>
<td>Defect Tracking Metrics</td>
<td>- Severity of each defect</td>
</tr>
<tr>
<td></td>
<td>- Location of each defect</td>
</tr>
<tr>
<td></td>
<td>- Way in which each defect is corrected</td>
</tr>
<tr>
<td></td>
<td>- Person responsible for each defect</td>
</tr>
<tr>
<td></td>
<td>- Number of lines affected by each defect correction</td>
</tr>
<tr>
<td></td>
<td>- Work hours spent in correcting each defect</td>
</tr>
<tr>
<td></td>
<td>- Average time required to find a defect</td>
</tr>
<tr>
<td></td>
<td>- Average time required to fix a defect</td>
</tr>
<tr>
<td></td>
<td>- Attempts made to correct each defect</td>
</tr>
<tr>
<td></td>
<td>- Number of new errors resulting from defect correction</td>
</tr>
<tr>
<td>Overall Quality Metrics</td>
<td>- Total number of defects</td>
</tr>
<tr>
<td>Metric Categories</td>
<td>Data Collected</td>
</tr>
<tr>
<td>---------------------------</td>
<td>---------------------------------------------------------</td>
</tr>
<tr>
<td></td>
<td>- Number of defects in each routine</td>
</tr>
<tr>
<td></td>
<td>- Average defects per thousand lines of code</td>
</tr>
<tr>
<td></td>
<td>- Mean time between failures</td>
</tr>
<tr>
<td></td>
<td>- Compiler-detected errors</td>
</tr>
<tr>
<td>Maintainability Metrics</td>
<td>- Number of parameters passed to each routine</td>
</tr>
<tr>
<td></td>
<td>- Number of local variables used by each routine</td>
</tr>
<tr>
<td></td>
<td>- Number of routines called by each routine</td>
</tr>
<tr>
<td></td>
<td>- Number of decision points in each routine</td>
</tr>
<tr>
<td></td>
<td>- Control-flow complexity in each routine</td>
</tr>
<tr>
<td></td>
<td>- Lines of code in each routine</td>
</tr>
<tr>
<td></td>
<td>- Lines of comments in each routine</td>
</tr>
<tr>
<td></td>
<td>- Number of data declarations in each routine</td>
</tr>
<tr>
<td></td>
<td>- Number of blank lines in each routine</td>
</tr>
<tr>
<td></td>
<td>- Number of jumps in each routine</td>
</tr>
<tr>
<td></td>
<td>- Number of input/output statements in each routine</td>
</tr>
<tr>
<td>Code Coverage Metrics</td>
<td>- Percentage of methods under test</td>
</tr>
<tr>
<td></td>
<td>- Percentage of classes under test</td>
</tr>
</tbody>
</table>

Table 1: Metric Categories vs. Data Collected
As this list suggests, there are a large number of metrics which can be used for software measurement. The software engineers and the software managers cannot usually even agree on what is important to measure, how to measure it or why to measure it. The biggest challenge for a software organization is to find the right goal oriented metric which would help in gaining quantitative insight into the development process and the developed products.

Lot of metrics like lines of code or hours worked can potentially have very counterproductive implications. The use of these metrics rewards the wrong behaviors; the commonly used phrase in metrics community, ‘you get what you measure’ highlights the problem. By tracking lines of code written, visible and unconscious incentives to generate lots of code are established. On the surface this may seem attractive. From the viewpoint of project manager it is gratifying to see lots of software code being written, but what should be measured is the functionality completed, business value generated and customers satisfied. In contrast, the more code that is generated the more code there is to maintain and extend in the future. Reducing the lines of code to a lower value may appear as if the project is going backwards on the surface but may end up adding more business-value to the organization. This phenomenon is what is popularly known as the ‘Hawthorn Effect’.

During the 1920’s Elton Mayo, a psychologist and sociologist [25], conducted a number of worker productivity experiments at the Hawthorne plant of Western Electric Company in Chicago. In these experiments, worker productivity was tested under varied lighting conditions. Although at first it looked like worker productivity went up in brighter conditions on further testing it was found that the productivity had risen due the workers awareness of their work being monitored and measured. This gave rise to what is commonly referred to as the ‘Hawthorn Effect’, which concludes that the process of measuring may actually influence the entity being measured thereby leading to false data. Because we will likely be influenced by what we measure, we
should be very careful about what we visibly measure. Lot of the traditional metrics models in use may suffer from the Hawthorne Effect. This phenomenon is also commonly referred to as ‘Metric Dysfunction’.

Some other key limitations of the software metrics in use in industry can be summarized as follows

- Manual – Due to manual data entry the data entry may be slow, corrupt and prone to errors.
- Static – Provides a snapshot in time, but doesn’t scale well for dynamic project requirements.
- Subjective – Data input is based upon individual interpretations, assumptions and biases
- Coarse – Provides high-level information, but limited detail or context at the process level
- Incomparable – Measured data varies from project to project, which limits comparability

The great challenge for the software metrics community is to produce effective metrics for the software development and testing process which take into account the crucial concepts missing from traditional regression-based approaches [10, 11, 12, 13]. Traditional regression based approaches use metrics for future predictions based on the metrics data from the past. While they might work in some cases they still don’t account for important factors like cause and effect relationships, expert judgement and constantly changing environments. Specifically for any collection of metrics to be effective, adequate thought needs to be given to these essentials
5.1 Goal Oriented

As pointed out by Einstein “Not everything that counts can be counted, and not every-
thing that can be counted counts”. This quote should be the fundamental idea behind any new
metrics program being developed. Lot of thought should be given to decide on what needs to be
measured, being sure that it is meaningful and doesn’t create an artificial incentive system within
the organization. Too often, organizations measure the wrong things – either metrics that lack
meaning to the business or metrics that result in unintended consequences and perverse incentives
leading to the Hawthorne Effect.

As pointed out by Lynch and Cross, business goals vary across an organization depend-
ing on the individual’s role [16]. The focus of people working in a specific department may be
different than the focus in the business units they report to. Nevertheless, the goals in hierarchical
organizations are related, and they all come down to competing on the basis of productivity,
flexibility and customer satisfaction. The measurement process should be designed to be initiated
at any organizational level where goals can reasonably be identified. Entering at a high level with
goals such as ‘Reduce cycle time’ or ‘Improve customer satisfaction’ has the advantage of ensur-
ing traceability of the resulting measures back to the primary business goals at that level. The
process can be started with looking at the goals of the person whose involvement will be most
important to implementing and sustaining the resulting measurement efforts. It can be initiated at
any level where quantitative information about products, processes, or resources would improve
the software development process. Activity based models [14] can also be used during the goal
setting process. In his book *Good to Great* [15], Collins discusses the attributes and behavior that
are common and unique to companies that have gone from a long history of mediocre results to a
long run of great results. One identified aspect is referred to as the ‘Hedgehog Principle’. In this principle, three questions are answered:

- "What can we be the best in the world at?"
- "What can we be passionate about?"
- "What is our one economic driver?"

This economic driver then serves as the one key metric by which all aspects of the organization can be measured. For example, at Walgreens, their driver is profit per customer visit. Other large institutions have chosen slightly different metrics. But Collins’ research concludes that all good-to-great companies have chosen a single economic driver metric that guides their entire decision making.

For example, every software company is interested in finding out the ‘time spent’ on various stages from software development to a complete deliverable. Having an exact measure for time helps them in better channelising the investment in various activities like development, testing, integration, customer support, bug removal etc. Having an exact measure for time spent on each of these activities enables the management to better plan for future initiatives. It is also of tremendous help in the risk assessment phase of the project. Having a standard across the company, for the kind of measures mentioned above enables better transparency across projects and teams. Measures like ‘time spent’ can then also be combined with other project level measures like project velocity, project bugs etc. to develop a metrics suite providing better decision making capabilities to the management.

The goal should be to select a few meaningful metrics that stand as a valid proxy for something that needs to be tracked to achieve visibility into software projects. The metric selected should enable the software project to be steered in the direction led by business goals. Care must be taken to not select too many metrics as it can lead to information overload, analysis paralysis.
and hunting phantom correlations which may result in strange or unpredicted behaviors. At the same time more than one or two metrics should be selected to ensure a balanced perspective and to mitigate the risk of behaviors aligning to measurement.

5.2 Data Collection

The challenge of collecting software engineering data is to make sure that the collected data can provide useful information for project, process and quality management and, at the same time, that the data collection process will not be a burden on development teams [28]. Adequate thought must be given on what data needs to be collected, at what intervals for the metrics. The goals of data collection should be established and the questions of interest should be defined before any data is collected. Care must be taken to define strategies for data collection so as not to over burden the available resources. The data extracted should be focused and accurate. Data should only be collected once the decision has been made to measure specific attributes.

Traditional methods for gathering software development metrics data have relied heavily on manual systems for entering and collecting data. As with any manual process, this makes data collection prone to human errors and speed limitations. The data collected may also get influenced by an individual’s preference or bias rather than what’s needed by the metric. This might also lead to over collection of information or unrelated data. To alleviate these issues, effort should be made to automate the data collection process. The automated data collection process for a metric should be seamless. The data should be gathered in the background without imposing any new processes or requiring developers to manually enter data.
To ensure proper oversight, many organizations commit all or part of a business day to check, validate and report on the progress of a software development process. Not only is this more work for the developers, it might also distract them from their primary responsibility of developing software. The metrics program should enable an organization to analyze the data that was collected after measurements in order to tune the organization's software process itself. Over a period of time the data collection procedures may become stale and inefficient for the organization. Feedback mechanisms involving people responsible for the collection of data as well as people consuming the data should also be designed. This helps in detecting inefficiencies in the data collection process early on, and takes necessary action to fix them.

5.3 Data Validation

One of the biggest concerns management in an organization has about metrics programs is validation of collected data as bad data automatically translates into bad business decisions. Neglecting data validation checks while designing metrics programs can result in incomplete, inaccurate and inconsistent information. Inaccurate data will inevitably lead to bad decisions, which can introduce risk into the software development process. In a lot of organizations software development projects are managed with traditional project management techniques which rely heavily on manually entered data like spreadsheets. The metrics based on manually entered data can be spurious. Studies similar to one done by Basili and Perricone [10] found that software data are error-prone and that special validation provisions are generally needed. A lot of factors can compromise manually entered data.
• Perception – People working in different teams may approach data entry with different mindsets and enter data based on their perception of what needs to be measured.

• Accountability – Sometimes people may only report data the way they want it to be perceived for the fear of being judged.

• Culture – A lot of projects may have globally distributed teams resulting in work cultural differences and communication gaps while entering data.

• Time – For data entry schedules which are typically very long and intensive, the sheer monotony associated with these tasks may lead to some human errors.

While developing automated data collection tools, basic validation checks like range checks, conditional checks, constraint checks, etc. and test cases to validate those checks should be an integral part of the design.

5.4 Actionable

Different approaches have been used in the software industry to implement software measurement. In a lot of these cases, measurement takes the form of ‘extra work’ and does not materially help the organization in achieving its objectives. Some organizations seek data for future audit purposes or collect metrics data simply because they can. Although it’s easier sometimes to collect more data now than to make changes in the future, collecting data without a clear perspective of where and how it will be used may result in high overhead and low value to the organization. The basic idea behind selecting a metric should always be to find out what actions can be taken using the data generated from the metric. Effort should be made to develop an action plan to react to values higher or lower than the allowable limits for the metric. The metric data
should have enough information for the team to understand the cause and respond to it accordingly. If later on, some cases are found where adequate action needs to be taken, then the action plan should be revisited and updated to include those actions. The basic questions to be kept in mind for generating actionable metrics should be

- Does the metric help fulfill the goals for the project?
- Is the data generated by the metric actionable?
- Can the metric data be aggregated without losing information?
- Does the metric add business value for the management?
- Does the metric have any limiting factors (human data entry, data accuracy etc.) and if so, does the system account for those limitations?

As the project progresses and the information needs change, so do the applied measures. The measurement process must be flexible enough to support existing software and technical and management processes and environments already in place, as well as adapt to changes needed for an ongoing project. The measurement process must be iterative, continually focusing measurement efforts on the most critical and actionable issues. It should be able to support the measurement of evolving process and product attributes as the project information needs and related objectives and issues change.

5.5 Measurement Culture

A lot of metrics programs fail during execution due to the high overhead nature of metrics programs and the burden they place on the developers and other participants in the software
development cycle. One of the practices to avoid while establishing a new metrics program is introducing process inefficiency.

To ensure proper oversight, many organizations will commit all or part of a business day to check, validate and report on the progress of a software development process. For example let’s assume that an organization has a time tracking system for each project and all the team members are supposed to enter time in it once everyday. If the new metrics program will require the same team members to enter time hourly compared to once everyday then that metrics program will most likely get a lot of pushback from the participants. Similarly, if instead of using an existing project time tracking system the team is supposed to enter time in a completely different, more complex time tracking system then it may face opposition from the team members also due to usability issues. Effort should be made to make the data collection program for the metrics as automated and unobtrusive as possible. If existing systems being used in an organization can be leveraged then people will be more supportive of the initiative and more productive as well. Before creating new systems for data collection it’s always beneficial to look at existing systems and come up with ideas to extend them to support newer requirements. Not only does this make people more receptive of the change but also re-uses the expertise of the team working on the existing system.

Another factor to consider when developing metrics is the challenge of collecting ‘invisible’ data. This refers to the human aspect of metrics program and the various attributes of a human being which are always difficult to collect. Robert Austin [18] points out that when measuring performance you have to get all the important factors under measurement. He says that the good thing about performance measurements is that ‘you get what you measure’. Then again the challenge with performance measurements is that ‘you get only what you measure, nothing else’. You tend to loose the human characteristics like insight, collaboration, creativity and dedication
to customer satisfaction. Establishing an employee rewards program based on performance metrics missing data on the aforementioned human attributes may result in partial failure of the program. In such cases, Austin recommends aggregating individual performance measurements into higher level informational measures that hide individual results in favor of group results. Some metrics may also result in ‘measurement dysfunction’, in which participants alter their behavior to optimize something that is being measured instead of focusing on the real organizational goals. As an example, if an organization is measuring productivity based on lines of code produced by a programmer as opposed to quality of code, it is expected that some developers may change their programming style to visually expand the volume of code they produce or code quickly without regard for defects. Austin concludes that organizations should choose between measurement-based management or delegatory style management. In measurement-based management the managers use objective data collected from metrics to make the decisions. In delegatory style management the task doers decide how measurement should be done and make the decisions based on measurement. Measurement based measurement works better in situations which involve simple repetitive tasks. Delegatory style management works better where each individual is responsible for solving complex problem/s using some unique expertise, which is usually the case in software development.

This is also where agile software development differs from traditional methods. Traditional methods have the inherent belief that measurement based management is the most efficient way of managing. The agile management on the other hand is a minimal, lightweight approach to management [29]. Agile management is essentially a hands-off style of management. It is highly delegated and provides empowerment at all levels. It works because the agile manager is empowered with making decisions, but so are the team leads and above all the developers and testers who do the direct work have direct input in those decisions. All the individuals in an agile team
take responsibility for delivery of value-added output. That responsibility involves knowing what is needed to complete a task but at the same time working with others to devise the most efficient way of doing that task. For example the software developers may know the best way to program a particular feature but may need guidance from the business analysts to understand customer requirements for that feature. The managers then have the task of devising the best way for the developers to interact with the business analysts to better understand customer requirements before the coding phase is initiated.
Establishing a metrics program to gain quantitative insight into the processes being followed in an organization and to take decisions based on the results of that program can be a challenging task in itself. The management may be concerned about the financial implications of starting a new measurement program. The project managers may be concerned about the procedural changes needed for measurement at different stages of the product and how it will affect the project timelines. The programmers may be concerned if it means extra work on their part for reevaluating and changing their day to day coding methodology. What is needed then is for the key stakeholders who will be part of the metrics program to take a structured approach for designing a formal process by which metrics will be collected. We feel that most successful measurement programs are based on a few manageable concepts. In this chapter we define what we think are the basic steps needed for selecting, designing and implementing a successful value-adding metrics program.
6.1 Identifying Metrics Consumers

The consumer of the metric/s is the person/s who will be making decisions or taking action based upon the metric; the person/s who needs and plans to use the information supplied by the metric.

There can be many different type of consumers for a metrics program within an organization. Each consumer may have a different work profile arising in completely different information requirements or in some scenarios a requirement for analyzing different set of attributes for the same entity. Metrics consumers in an agile team of a modern day Information Technology company, can be broadly categorized as

- **Line of Business Managers**: This group of consumers is interested in defining roadmaps for the software product and aligning them with business goals. They are interested in metrics to better predict the return on investment, rationalizing process efforts for maximizing profit and optimizing the process for better risk management.

- **Software Project Managers**: This group of consumers is interested in being able to better predict the scope of the project. Being involved in the day to day management of the development teams they are interested in clearly understanding the project status, identifying roadblocks and prompt defect resolutions.

- **Software Engineers / Programmers**: This group of consumers does the actual software development. They are interested in metrics that can help them in architecting better solutions, analyzing current code base and resolving defects in software.

- **DataBase Administrators**: This group of consumers is responsible for managing the data systems in use for the organization. They are interested in metrics to better under-
stand the system usage, predict system requirements, identify infrastructure bottlenecks and resolving defects in data systems.

- **Business Analysts**: This group of consumers is responsible for analyzing the business needs of technology initiative and making sure that customer requirements are being met. They are interested in metrics to quantify business value needed to transform customer requirements into product features, accessing current project status and validating product features.

- **Quality Assurance Managers/ Testers**: This group of consumers is responsible for developing test cases and identifying the defects in software. They are interested in metrics for setting up performance benchmarks, testing quality assurance entities and testing the overall quality of every product component.

- **System Administrators**: The task of a system administrator is to configure, to manage and to troubleshoot the servers, networks of workstations and development and testing environments used by the project team. Also, the system administrator may be involved in the productionizing of the system being developed.

The general rule for approaching the viability of any metric should be: ‘No Consumer, No Metric’. Business goals can only be achieved if the data from the metrics is acted upon by the consumers. Having no consumers to act upon and take ownership for the metric data can result in wasted effort.
6.2 Business Goals

The next step should be to identify the business goals which the software project and the organization as a whole needs to achieve. Basili and Rombach [21] have defined a Goal/Question/Metric paradigm which can be used to refine goals and design measures based on them. The GQM paradigm represents a systematic approach to tailoring and integrating goals with: models of the software processes, software products, and with particular quality perspectives of interest. GQM focuses on the specific needs of the software project and of the development organization. Measurement goals are defined on the basis of high-level management goals, goals are then refined into questions and questions are then refined into metrics. The GQM paradigm provides a method for top-down metric definition and bottom-up data interpretation.

![Figure 1: GQM Paradigm](image)

Application of GQM divides measurements into two parts. First, there is the definition process during which goals, questions, metrics and extra procedures are defined. Second, there is
the interpretation process, during which the collected data is analyzed, improvements are identi-
fied and experiences are described.

The definition process for GQM measurement consists of the following activities

- **Prestudy**: This is an important preparation activity for measurement. At this stage the project entities are characterized and goals for measurement are made explicit.

- **Measurement Goal Selection**: During this activity informal improvement goals are de-
scribed, refined and ranked. Priorities are assigned and it is decided which goals will be used and transformed into GQM goals. The goals that are selected to be used in GQM will vary depending on the level within the organization. Sometimes the same business goals may mean different things for different people relative to their work area. Any goal setting meet-
ing should have members from different teams to identify the pain-points and bottlenecks.

- **GQM Planning**: This activity involves actual design of the measurements. At this stage inter-
views are held with key project members to retrieve relevant information. GQM paradigm is then applied to the information to generate a detailed tree of goals, questions and metrics.

- **Measurement Planning**: At this stage various data collection procedures are developed and various automated tools for data collection are also introduced to optimize the process.

6.3 Identify Quantifiable Questions

Once the business goals behind developing a metric have been identified the next step is to define the quantifiable questions that need to be answered in order to ensure that each goal is being covered by the metric. When identifying questions it is important to keep in mind the goal(s) that are being addressed and how the desired measurement results will be used. The an-
swers to these questions will help identify the metrics, that’ll be used to measure the progress, being made towards achieving the business goals. For example, if in an agile work session the goal was to focus on defect tracking and fixing defects in the upcoming monthly iteration, we might select these quantifiable questions:

- Are the issues really defects or new requirements by customers?
- How critical are the defects reported?
- What are the defects being addressed in this monthly release patch?
- Did the defects arise due to improper integration of code?
- Are there existing test cases which need to be updated?
- Are there manual test cases which can be automated?
- Is regression testing needed before the release?
- Does the project roadmap account for time needed to fix and regression test these defects?
- Were there specific modules where defect density was higher than the rest?

### 6.4 Identify Metrics

The next step is to select metrics that provide the information needed to answer the questions decided upon in the goal setting meetings. For agile projects the questions should have been made into story cards for the upcoming iterations in the goal setting meetings and prioritized based on the business requirements. These story cards can now be revisited at this stage to refine them into metrics. Some examples of questions which can evolve into metrics are as follows
1) “Are we on-track for the project roadmap?”
Metric: Features Completed / Features Remaining

2) “What is the average bug removal time for this iteration?”
Metric: Man hours spent in selected iteration / Total number of bugs removed in selected iteration.

3) “How many issues were completed in last iteration?”
Metric: Total count of story cards * Issues per story card

4) “How can we increase customer satisfaction?”
Metric: “Satisfaction rating 1 to 10, 10 being very satisfied”

5) “How many visitors on the website are converting into customers?”
Metric: Number of Sales / Number of Visitors * 100

6) “How much revenue are we making from each visitor?”
Metric: Product Sales / Number of Visitors

7) “How many website visitors are signing up for the newsletter?”
Metric: Number of Newsletter subscribers / Number of website visitors * 100

Final metrics should be discussed as a group. Use cases should be discussed to explain how these metrics will lead to achievement of goals by a metrics consumer. During these discus-
sions, if additional metrics are discovered which elaborate on specific attributes of entities being used by a metrics consumer, then those metrics can also be included.

Comparisons should also be made between the answers collected during the development phase and the post delivery phase to get a perspective of how much progress has been made. These comparisons can be assimilated into reports as discussed later to be relayed to executive management to display improvements in achievement of goals using measurement.

6.5 Define Attributes and Entities

After the requirements statement has been documented for each metric in the previous step the next step should be to extend the requirements and agree upon some standard definitions for the entities and their measured attributes for each metric. We should examine each agreed upon metric and identify the entities that are implicit in it, then list the pertinent attributes associated with each entity. Pertinent attributes are usually cited in the questions either implicitly or explicitly. Identifying the entities may also give leads to the team to think of other questions and attributes. The list of entities and attributes for each entity should be the principal outputs of this step. The attributes will become candidates for the things that need to be measured.

Effort should also be made to clearly define and develop an organization-wide glossary for the entities and their attributes that will be measured. Even general terms like bugs, defects, change control process, ticket, project scope etc. may have a different meaning for people in different teams. An unclear understanding of the entities involved in measurement may lead to collection of wrong or irrelevant data. These differences may be even more prevalent across different organizations. For example, the terms defect report, bug report, incident report, action items
report, fault report, issue tracking report or customer call report may be used by various organizations to mean the same thing, but unfortunately depending on the specific project they may also refer to different entities. One application support engineer may use customer call report to refer to specific customer complaint and fault report as the description of the defect in software, while another application support agent in another company may just use the fault report to document both the initial customer complaint as well as the description of the defect in software. Not having an organization wide standard glossary, makes cross-organizational metrics program even more challenging. The existing definitions [18] within the industry can be used as a starting point to develop an organization wide glossary. Definitions that match the organizational objectives may be used as-is or as a basis for creating organization specific definition.

When listing attributes, it is useful to keep in mind the distinction between attributes (the characteristics of an entity) and measures (the scales and rules used to assign values to attributes). Care must be taken not to get overly specific when building this list of attributes. A few examples of entities and their attributes are as follows

**Examples of Entities and Attributes for Improving Performance of a Change Management Process**

<table>
<thead>
<tr>
<th>Business Goal</th>
<th>• Improve the Change Management Process for the Organization</th>
</tr>
</thead>
<tbody>
<tr>
<td>Question 1</td>
<td>• Do change requests received from customers contain the information needed to generate timely and effective changes?</td>
</tr>
<tr>
<td>Entity</td>
<td>• The set of change requests received from customers</td>
</tr>
</tbody>
</table>
| Attributes | • Size (count of requests for change received)  
|           | • Adequacy (the percentage of change requests with all the required fields filled in correctly)  
|           | • Defect distribution (for each required field, the percentage of requests)  
|           |   - with that field omitted  
|           |   - with that field filled in incorrectly |

**Table 2: Entities and Attributes for Example Question 1**

| Business Goal | • Improve the Change Management Process for the Organization |
| Question2     | • How large is our backlog of our customer change requests? |
| Entity        | • The backlog of customer change requests |
| Attributes    | • Size of backlog (change requests received – change requests completed)  
|               | • Size of queues awaiting action at each stage of the change management process  
|               | • Total effort (estimated) required for clearing the backlog |

**Table 3: Entities and Attributes for Example Question 2**

| Business Goal | • Improve the Change Management Process for the Organization |
| Question3 | • Is the response time for fixing defects compatible with customer constraints? |
| Entity     | • The change management process |
| Attributes | • Customer’s expectation for cycle time  
|            | • Frequency distribution of time from receipt of a change request until it is implemented  
|            | • Frequency distribution of time from receipt of a change request until it is installed at the customer’s site.  
|            | • Average amount of time that requests spend at each step of the change management process |

Table 4: Entities And Attributes for Example Question 3

6.6 Choosing a Measurement Model

Software Measurement for a metric has been broadly categorized into Fundamental and Derived Measurement. Fundamental measurement deals with the measurement process that takes place at an early stage of the measurement model development. Derived measurement takes place later, when new measures are defined in terms of others previously developed. Fundamental or base measures are measured directly and their measurement function typically consists of a single variable. Examples of fundamental measures include the Lines Code Metric for a specific iteration, the number of man hours spent on a user story point or the number of ideal days spent on a
user story point. Derived metrics are more complex and are modeled using mathematical combinations (eg. equations or algorithms) of fundamental measures or other derived measures. Examples of a derived measure would be Measure of Defect Density for a module after three iterations, an inspection’s preparation rate modeled as the number of lines of code reviewed divided by the number of preparation hours, the velocity of a Agile project modeled as the number of story points completed divided by the number of ideal work days. At this stage decisions should be made on what kind of measurement model will be used for the metrics. For using derived measurement existing metrics being collected across the organization should be reviewed and included as part of the model. In general the model should be kept concise and focused to include only the attributes and entities necessary to help make better decisions.

Once the model has been designed it should be validated based on whether it can be used to improve the overall quality of the product. The following questions may be of help in deciding the practicality of a model

- Can the model use existing metrics or will it need process changes across the organization?
- Can we predict the degree of maintainability needed for this model?
- Does it help in predicting the error-proneness of the system?
- Can we predict the amount of effort needed to extract the quantifiable attributes and entities needed by this model?
- What are the criteria for validation of this model?
- Is collective ownership possible?
- Is the model easy to update at a later date?

For example a common task in agile development is determining the workload size of a project. Measure of size is important because knowing the amount of work to do can help greatly
in deriving the project costs and release estimates. For this type of model the organization can use existing metrics like Ideal Days or User Story Points. Ideal Days are the imaginable amount of time that a developer needs if there were no interruptions while working full time and any resource needed by the individual is readily available. User Story Points are also very common in agile projects. They are very high level definitions of a business requirement, broken down into smaller tasks so that developers can easily estimate the amount of effort needed to complete those tasks in number of ideal days. Story points are therefore derived metrics. The team assigns a count of 1 or 2 ideal days to the relatively simple standard tasks like configuring build scripts or writing unit tests and estimates the size of other tasks by comparing them to these standard tasks. A count of Ideal Days or User Story Points can then be displayed to the whole team for estimation of the workload size of the project.

6.7 Choosing a Measurement Method

At this stage of measurement program planning, the exact process needed for measuring the metric needs to be defined. The measurement method should define the mapping system which will used by the organization to assign numbers to the attributes of the entity being measured.

Let’s assume that an organization decides on using the Lines of Code metric as its measurement model for counting the software size. The situation gets complicated because multiple definitions exist for the basic unit of Line of Code (LOC) and its most common multiple Kilo Line of Code (KLOC) for thousand lines of code. Line of code can be counted by different organizations in different ways like with or without comments, including or excluding declarations,
including or excluding deleted code etc. Varying definitions may even exist across different divisions within an organization. The definitions for size measurements being used like Line of Code should be established across the company and adhered to for every project. The company specific definitions for such methods also make it difficult to adopt an existing size measurement tool as-is without some customization.

To keep the measurement methods clear and concise, standard measurement checklists as suggested by the Software Engineering Institute[33] can be used. Checklists are commonly used in the industry to communicate effectively what a set of measurement represents. For the management, checklists are similar to business requirements specification that describes what measurement entities and attributes are required for the measurement program. Such measurement checklists should include the following columns

- Include: Choice of whether the specific attribute is being included in the count
- Exclude: Choice of whether the specific attribute is being excluded from the count
- Total Count: A comprehensive view of the included and excluded attributes of the entity being measured
- Value Count: Count of a single attribute of the entity being measured which provides a one dimensional view.
- Array Count: A multidimensional view of two or more major attributes of the entity being measured.

To avoid any confusion or overlap the checklist should be accompanied with a definition of each item. This information can also be appended to a company wide data dictionary. The information should at a minimum include a definition of the entity to be measured, definition of the attributes of the entity being measured, any counting rules that apply and any company specific process or information relevant to correct interpretation of the checklist.
As an example the figure below shows a checklist [33] that can be used for defect-tracking in an organization. Formats for checklists like this one should be tailored to the defect-tracking process used within an organization, and sent out to all the respective teams concerned with defect resolution.
<table>
<thead>
<tr>
<th>Problem Status</th>
<th>Include</th>
<th>Exclude</th>
<th>Value Count</th>
<th>Array Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>Open</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Recognized</td>
<td></td>
<td></td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td>Evauated</td>
<td></td>
<td></td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td>Resolved</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Closed</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Problem Type</th>
<th>Include</th>
<th>Exclude</th>
<th>Value Count</th>
<th>Array Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>Software defect</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Requirements defect</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Design defect</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Code defect</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Operational document defect</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Test case defect</td>
<td></td>
<td></td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td>Other work product defect</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

| Other problems        |         |         |             |             |
| Hardware problem      | ✓       |         |             |             |
| Operating system problem |     |         |             |             |
| User mistake          |         |         |             | ✓           |
| Operations mistake    |         |         |             | ✓           |
| New requirement/enhancement |       |         |             |             |

| Undetermined          |         |         |             |             |
| Not repeatable/Cause unknown |     |         |             | ✓           |
| Value not identified  |         |         |             | ✓           |

<table>
<thead>
<tr>
<th>Uniqueness</th>
<th>Include</th>
<th>Exclude</th>
<th>Value Count</th>
<th>Array Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>Original</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Duplicate</td>
<td></td>
<td></td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td>Value not identified</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Criticality</th>
<th>Include</th>
<th>Exclude</th>
<th>Value Count</th>
<th>Array Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>1st level (most critical)</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2nd level</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3rd level</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4th level</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5th level</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Value not identified</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Urgency</th>
<th>Include</th>
<th>Exclude</th>
<th>Value Count</th>
<th>Array Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>1st (most urgent)</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2nd</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3rd</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4th</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Value not identified</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Figure 2: Checklist Example of measurement method for measuring Defects**
<table>
<thead>
<tr>
<th>Finding Activity</th>
<th>Include</th>
<th>Exclude</th>
<th>Value Count</th>
<th>Array Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>52</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Figure 2: Checklist Example of measurement method for measuring Defects</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Synthesis of</th>
<th>Include</th>
<th>Exclude</th>
<th>Value Count</th>
<th>Array Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>Design</td>
<td></td>
<td>✔</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Code</td>
<td>✔</td>
<td>✔</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Test procedure</td>
<td>✔</td>
<td>✔</td>
<td></td>
<td></td>
</tr>
<tr>
<td>User publications</td>
<td>✔</td>
<td>✔</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Inspections of</th>
<th>Include</th>
<th>Exclude</th>
<th>Value Count</th>
<th>Array Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>Requirements</td>
<td>✔</td>
<td>✔</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Preliminary design</td>
<td>✔</td>
<td>✔</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Detailed design</td>
<td>✔</td>
<td>✔</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Code</td>
<td>✔</td>
<td>✔</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Operational documentation</td>
<td>✔</td>
<td>✔</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Test procedures</td>
<td>✔</td>
<td>✔</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Formal reviews of</th>
<th>Include</th>
<th>Exclude</th>
<th>Value Count</th>
<th>Array Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>Plans</td>
<td>✔</td>
<td>✔</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Requirements</td>
<td>✔</td>
<td>✔</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Preliminary design</td>
<td>✔</td>
<td>✔</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Critical design</td>
<td>✔</td>
<td>✔</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Test readiness</td>
<td>✔</td>
<td>✔</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Formal qualification</td>
<td>✔</td>
<td>✔</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Testing</th>
<th>Include</th>
<th>Exclude</th>
<th>Value Count</th>
<th>Array Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>Planning</td>
<td>✔</td>
<td>✔</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Module (CSU)</td>
<td>✔</td>
<td>✔</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Component (CSC)</td>
<td>✔</td>
<td>✔</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Configuration item (CSCI)</td>
<td>✔</td>
<td>✔</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Integrate and test</td>
<td>✔</td>
<td>✔</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Independent test, and valid.</td>
<td>✔</td>
<td>✔</td>
<td></td>
<td></td>
</tr>
<tr>
<td>System</td>
<td>✔</td>
<td>✔</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Test and evaluate</td>
<td>✔</td>
<td>✔</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Acceptance</td>
<td>✔</td>
<td>✔</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Customer support</th>
<th>Include</th>
<th>Exclude</th>
<th>Value Count</th>
<th>Array Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>Production/deployment</td>
<td>✔</td>
<td>✔</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Installation</td>
<td>✔</td>
<td>✔</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Operation</td>
<td>✔</td>
<td>✔</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Undetermined</td>
<td>✔</td>
<td>✔</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Finding Mode</th>
<th>Include</th>
<th>Exclude</th>
<th>Value Count</th>
<th>Array Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>Static (non-operational)</td>
<td>✔</td>
<td>✔</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Dynamic (operational)</td>
<td>✔</td>
<td>✔</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Value not identified</td>
<td>✔</td>
<td>✔</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
6.8 Reporting Structure and Mechanism

The next step is to define and develop the reporting of the selected metrics. This involves defining the report audience, deciding upon the reporting format to be used for displaying the reports and the reporting mechanism to be used for distribution of reports to the respective audience.

Within an organization various report audiences may have different goals and requirements for reporting of metrics. Not all requirements for the various audiences can be fulfilled in one report. Once the target audience has been decided, they should be involved in reviewing the proposed reporting structure and mechanism to make sure that their goals and requirements are being met by customized reports. For deciding on the report audience the general rule should be that project teams and individuals who are responsible for data collection for the metrics should benefit from reporting. Specific project teams should receive the analysis and reporting of their projects which the metrics may have covered. These reports would help them to further streamline the process and identify improvements for upcoming projects. Being the report audience also enables the project teams to suggest any necessary changes as well as be more committed to implementing those changes.

The reporting format defines the physical look and presentation of the reports in the most effective way possible for the end user. This may involve brainstorming on questions as discussed below:

- Is the newest measurement included in the report with measurement values from previous agile iterations?
- Should the values be structured as a trend chart to track values over multiple agile iterations?
• What type of chart (for e.g. bar, line, pie, area) should be used to display the data?

• Should detailed analysis be included with the report?

• Should goals and control values be also included in the report?

• Are there existing industry standards that can be used for this report?

• What multimedia features should the report contain?

Although the format and layout of the reports can be defined at this stage, they may have to be revisited at a later date when actual data is available to determine the best visual representation of the data. Data on the reports should be accompanied with its context and explanation. This is necessary to make sure that the information is interpreted and used appropriately by the audiences. As an example let’s consider a report which needs to communicate the project cost till date, in contrast to the the total allotted budget for the project. If the report says that x dollars have been spent so far and the total budget is y dollars the report consumers might not have enough information to judge if it is a good or bad situation. The better way to report would be to change the reporting structure to state that x dollars have been spent to date and that as per the estimate only y dollars should have been spent. If the trend continued then the estimate for the final cost of the project will go up by z dollars. Reporting with this context may help the report consumers better understand what the numbers are saying.

As an example for agile projects let’s consider a vital metric: ‘Features Completed vs Features Remaining’. Cumulative Flow Diagrams are great ways to report features completed versus features remaining.
The sample diagram shows the features completed versus the features remaining for a project that is still in progress. The blue area represents all the planned features to be built. This number has risen from 400 to 420 in June and then to 450 in August as additional features were added to the project. The yellow area plots the work in progress, and the green area shows the total number of features completed.

Adequate mechanism should be decided upon for distribution of the reports at a desired frequency. Depending on the needs of the audience, different reports may be needed at different time intervals. Some project managers prefer to have visibility at a very granular level and may need weekly reports for each task which is in progress or completed. Others might prefer to receive reports solely for the metrics which are measuring high-risk data. For example reports pertaining to finance may need to be distributed quarterly for line of business managers, while defect tracking metrics reports may need to be distributed at the end of each bi-weekly iteration for project managers. At this stage, access rules should also be put in place for the reports for data security purposes. This should clearly define the users or groups which will have access to the reports and what rights they have (for e.g. create, read, update, delete) on the report data.
6.9 Data Collection and Repository

The step involves defining who collects the data needed by the metrics program, what data needs to be collected and how to collect it. The data should be collected only at points necessary to support the selected metrics. If the organization has plans to benchmark the metrics against industry data, then the data collection entities should be kept in line with those collected by the industry for easier and more accurate comparisons.

For example, if the data collection plan is being implemented for the defect tracking metrics program being used in an organization, then the metrics implementation team can collect data on the following entities

- Where did the defect occur?
- When did the defect occur (development phase, testing phase or post production)?
- What was the observed behavior?
- What were the consequences of the defect?
- How did the defect occur?
- Why did the defect occur?
- How were the users affected by the defect?
- How much did it cost (for example man hours) in fixing the defect?

This kind of data collection can be easily integrated with the development life cycle, so that it becomes a part of the process and is not perceived as something extra.

Another important part of the data collection process is to define a list of data ownership roles and the corresponding data entity owned by the role. This can be a list similar to defining metrics consumers as discussed in section 6.1 and should assist in the most appropriate assignment of personnel for data collection. An example data ownership list is defined below.
<table>
<thead>
<tr>
<th>Data Owners</th>
<th>Data Owned</th>
</tr>
</thead>
<tbody>
<tr>
<td>Project Managers</td>
<td>• Project roadmap</td>
</tr>
<tr>
<td></td>
<td>• Planned Iterations</td>
</tr>
<tr>
<td></td>
<td>• Project Budgets</td>
</tr>
<tr>
<td>Business Analysts</td>
<td>• Project business requirement analysis</td>
</tr>
<tr>
<td></td>
<td>• Project priority</td>
</tr>
<tr>
<td>Software Engineers</td>
<td>• Time spent on each story point</td>
</tr>
<tr>
<td></td>
<td>• Defects resolved per iteration</td>
</tr>
<tr>
<td></td>
<td>• Assigned defects</td>
</tr>
<tr>
<td></td>
<td>• Cause of a specific defect</td>
</tr>
<tr>
<td>Quality Assurance Analysts</td>
<td>• Test plan for a specific module</td>
</tr>
<tr>
<td></td>
<td>• Problem report after testing</td>
</tr>
<tr>
<td></td>
<td>• Test coverage reports</td>
</tr>
<tr>
<td>Change Control Specialists</td>
<td>• Change control plan</td>
</tr>
<tr>
<td></td>
<td>• Modules elevated in monthly release</td>
</tr>
<tr>
<td></td>
<td>• Live support issues</td>
</tr>
<tr>
<td>End Users</td>
<td>• Problem reports for specific features</td>
</tr>
<tr>
<td></td>
<td>• Feature enhancement requests</td>
</tr>
<tr>
<td></td>
<td>Observed behavior of defects</td>
</tr>
</tbody>
</table>

**Table 5: Data Ownership Examples**
For actual data collection, existing data collection forms/templates should be used. In-house mechanisms like checklists and spreadsheets are very commonly used. Automated tools can also be purchased for performing some of the data collection. Automating the data collection process reduces the additional overhead and improves data integrity. The data collection process selected should be easy to use and not taxing for the organizational technical capabilities and resources. If third party tools are purchased for automating the data collection process, then a training program for the employees should also be devised. The training program should be focused on ensuring correct capture and reporting of the measurement data through proper use of the tools, as well as correct interpretation of the collected data.

A considerable amount of effort is expended in collecting valuable metrics data so it is advisable at some point to import all the measured data into a central metrics repository. Depending on the complexity of the architecture needed, a decision should be made on whether to develop an in-house metrics repository or to buy a commercial off the shelf database management system. The repository of choice should fit well with the existing infrastructure and be extensible to accommodate more metrics in the future. A system that is too complex and too expensive to implement may do more to disable the metrics program than to enable it. Once the central metrics repository is functional the project managers of each team should coordinate efforts to disable use of various data collection vehicles for example custom checklists, software tools etc. being used by different teams and secure commitment from them on using the central metrics repository. Organizational policies, processes and procedures should also be put in place for regular maintenance and updates to the central metrics repository. A well-designed and updated central metrics repository can add to the organization’s capabilities to make fact-based decisions for ongoing and future projects.
6.10 Managing People

It is people working in the organization that can make or break a metrics program. Many people may have difficulty in accepting that procedures that have been successful and are currently being used need to be monitored using metrics. Their defense of the existing processes and procedures may limit the acceptance of the metrics program. In such cases, the metrics implementation team may have to ‘sell’ the metrics program to people working at different levels within the organization as part of the implementation.

The executive management may readily accept a metrics program if it translates into a positive return-on-investment for upcoming initiatives. The software developers may buy into the metrics program if it translates into better coding practices and development standards while cutting down on their effort and time. Project managers may welcome the idea if it means data to better streamline and organize their projects. Acceptance of the metrics program by people within the organization is necessary as it leads to increased levels of cooperation and overall process improvement.

The metrics program can also be coupled with an awards program to recognize outstanding contributions made by teams. Care must to be taken not to measure individuals and review their performance while looking at measurement data. As an example, if we measure productivity in lines of code per hour, then people may concentrate on their own work more to the detriment of the whole team and project. Even worse, they may come up with unique ways of programming the same functionality padded with extra lines of code to appear more productive. To help the team overcome the fear, they should be educated about the importance of the metrics program. They should understand why measurement is important and how the intended data will be put to use.
It’s harder to abuse the metrics data if managers are unaware of which individual the data came from. To ensure inter-organization metrics data privacy, measures similar to ones suggested by Grady [19] can be put in place. This involves classifying each base measure collected into one of these three privacy levels:

- **Individual Privacy**: Only the individual who collected the data knows about the data. For a broader project perspective data collected by various individuals may be pooled together.

- **Project Team Privacy**: Data collected by individuals in a project team is visible only to project team members and not shared across the organization. For a broader organizational perspective data collected by various project teams may be combined to make decisions.

- **Organization Privacy**: Data can be shared throughout the organization by individuals working in different teams.

Implementing a successful metrics program may involve a lot of incremental changes to the processes and procedures being followed within the organization. This may be time consuming and the gains may only be noticeable after a period of time. As the metrics program implementation may span a couple of years, it may also suffer from a loss of momentum. To avoid such weak spots management must continuously reinforce and convey the benefits of implementing the metrics program to employees. A positive work culture focused on discipline, patience and team work should be promoted by the management.
CHAPTER 7

CLASSROOM ASSESSMENT INITIATIVES AND SOFTWARE METRICS

Academic assessment techniques have been in use for the past 100 years or so for gathering information on students learning and skill level. The most common assessment technique in use is to give standardized tests to students at the end of an instructional unit or semester. The general idea behind standardized testing in schools was to make the schools accountable to the public as well as to make sure that student’s knowledge and skill was measurable on a standard scale. A vast amount of research has been done in this area which has since suggested that if applied incorrectly there can be a lot negative consequences of classroom assessment. We believe that the challenges in applying successful assessment techniques in academics and promoting motivational learning in students are similar to the ones in applying a successful employee performance assessment program in a company and motivating the people involved. For these reasons we study the recent initiatives that have been undertaken in classroom assessment to motivate students for learning, and discuss why and how some of those techniques are applicable for human resource performance and skills assessment in software development industry.
7.1 Classroom Assessment Motivation

In our review of the existing literature on classroom assessment [30, 34, 35, 36] we found that there are a few basic reasons for performing classroom assessment which also has some parallels with the software development industry.

7.1.1 Pre-Assessment and Curriculum Planning

Pre-assessment techniques in education are used for measuring the current skill level of students as well as for assessing learning in progress. For example, a language teacher may create pre-assessments tools like questionnaires or peer group discussions to find out the current skill level of students in a class. Then after analyzing the assessment results, the teacher can then create a tailored curriculum focused on honing specific language skills like grammar, vocabulary etc. which need further improvement.

This is similar to the measurement activities performed by companies using software metrics to evaluate software quality or to identify problem areas in specific software modules that need further improvement. While designing and implementing a software development process various software quality attributes like integrity, interoperability, flexibility, maintainability, portability, expendability, reusability, resilience and usability have to be kept in mind. Various quality assurance tests, assessments and measurements can then be used to assess the achievement and adherence of these attributes in projects.
7.1.2 Assessment for Accountability

This form of assessment is usually done at the end of a specified period of time to evaluate how the acquired skills after taking a class stack up against standardized educational assessments. For example an end of unit test, final course grades, state board examinations are all standardized assessment techniques used for external accountability. Course grades help in accounting student achievement for those beyond the classroom. Similarly, state exams help in accounting student achievement for guardians, policymakers and college admission officials.

Similar measurement activities are also common in software metrics with companies instituting software metrics programs for acquiring higher levels of CMM [9]. Another common usage of standardized assessments in software industry is Software Certification. Software certifications not controlled by any central certification authority, are usually offered by vendors and are designed for testing the individual’s skill level on specific vendor products. With companies investing substantial capital over time on specific products, there is demand for people with specific software product skills. Some software consulting companies often use the number of certified professionals working in their organization as credentials of how fast and effectively they can and will be able to deliver solutions to their clients. Software certifications thus translate into greater marketability in the job market for individuals as well as organizations. Although there are several efforts under way to create a standardized certification process in the software industry no single consensus has yet emerged. It is therefore hard to design one exam or one particular set of exams which can clearly measure the level of competence of software professionals.
7.1.3 Selection and Ranking

Another common use of assessment techniques in academics is for selection and ranking of students among peers or across local and national institutions. Tests are given to students to judge how they differ from each other. These tests generally referred to as Norm Referenced Measures lay more emphasis on reliability of data rather than meeting the curriculum standards of a grade. The underlying assumption for doing this is that some students are smarter than others and grades should reflect differences in ability. Reliability is of primary importance as based on the test data decisions are taken on whether specific students need special programs or whether they should be awarded merit based scholarships etc. The Graduate Record Examination (GRE) scores used by graduate schools to award admissions is an example of such Norm Referenced Measures. The GRE questions are formulated by specialists from various fields. All the questions are first pretested in standard testing conditions. The first time questions are then further analyzed by experts for their usefulness and weaknesses. All the ones that are found satisfactory after the analysis then become part of a pool of questions from which future tests are developed. Similarly, lot of companies are using human resource assessment techniques like software skills tests, in person interviews, standardized performance assessment forms and scorecards for ranking, selection and retention of employees.

7.2 Classroom Assessment Initiatives

The traditional belief has been to motivate students for learning using the pressure and anxiety of attaining higher test scores. These ‘Assessment Of Learning’ techniques commonly
known as summative assessment while beneficial for decision makers at the school, district, state and national level has also resulted in discouraging a section of learners who could not handle the anxiety associated with standardized testing and assessment programs. Due to this drawback in recent years more emphasis has been laid on ‘Assessment For Learning’ techniques commonly known as Formative Assessment which promotes student achievement and learning rather than higher test scores.

### 7.2.1 Summative Assessment

This involves activities which result in a mark or a grade for judging a student’s performance. The student grades are also used to determine the awards at the end of the academic year. The grades achieved by students of an institution on standardized tests at a state or national level are also used for accountability of educational institutions and comparison among them. These factors thus translate into stress for both students and staff when dealing with assessment. Pellegrino[36] summarizes the concerns about the effectiveness of summative assessment

- Summative assessment doesn’t provide tools to effectively capture the complex knowledge and skills acquired by students.
- Summative assessment doesn’t provide the full scope of a student’s understanding and doesn’t help teachers with any pointers on the type of intervention needed to improve a student’s deficiencies.
- Summative assessment only provides a snapshot of a student’s skills and achievement at the time of assessment but doesn’t provide tools for continuous monitoring of a student’s progress over time.
7.2.2 Formative assessment

This involves activities which rather than judging the learning expertise of a student using grades, are more aligned towards creating a feedback mechanism for students about their learning. Rather than ranking and grading of students, formative assessment and feedback is more focused on learning activities in which both teachers and students can get involved. The feedback from students is also used to adapt the teaching to better suit the needs of the students.

Although over the decades a lot of effort has been put into designing and implementing better grading schemes in higher education, recent shift in higher education has been towards designing better formative feedback mechanisms to promote ‘Assessment for Learning’. Black [30] suggests the following as the basic principles of formative feedback:

- The learning objective should be made clear to students
- Formative Feedback should be designed such as to measure the student’s current learning state
- Formative Feedback mechanisms should be used as a means for closing the gap between the student’s learning state and the learning objectives
- Formative feedback should be of high quality to be effective.

A combination of various tools like anecdotal records, checklists, rating scales, brainstorming sessions with students, conferences, journals and homework is used by teachers to implement ‘Assessment for Learning’. By using appropriate tools and daily observation of student’s progress, teachers can get a sense of whether students understand the tasks and whether they are
developing valued skills. If problems arise then they can quickly step in to provide direction. The information gathered about student’s progress on valued performances such as research reports or projects is then shared with students during conferences or using written feedback mechanisms. By using these techniques effectively a positive learning atmosphere is created for the students where they feel like they are a part of the learning process and don’t fall prey to a negative outlook of just beating other students in the grade race.

7.3 Individual and Organizational Performance Assessment

In the present day, globally-competitive, knowledge-based, economic paradigm, people working in a company are the intellectual assets which have come to define how a company performs. The Human Resources department is responsible for managing this intellectual asset of a company. This responsibility includes having an oversight of a range of activities that should span the entire lifecycle of an individual’s employment with the company such as recruitment of new hires, creation of new roles, tracking of applicants, new hire orientations, management of employee benefit programs, employee performance tracking, tracking of compensation, employee satisfaction surveys and employee exit processing. Traditionally, such personnel departments have been highly process oriented and involved for the most part in creation and management of documentation. Every step from hiring an employee to termination and everything in between had to be carefully documented. With the advent of technology this kind of cataloging and tracking can be easily accomplished using automated tools, allowing them to focus more on developing ways to better and more effective utilize the human capital. Using employee performance assessment metrics has been one such initiative.
Skill assessment tests, in person interviews, human resource scorecards, human resource programs to enhance attitudes and skills, performance reviews and employee satisfaction surveys are just some of the assessment tools that have now become commonplace. In a knowledge based economy, the skill level and knowledge requirement for the staff of an organization has to continuously evolve. In the past few decades, the Human Resource departments have started utilizing metrics to gauge the performance of employees and to make sure that the human capital is aligned properly with the business goals as well the knowledge and skill requirements for their respective roles. While beneficial if applied correctly, human resource metrics focused on accessing employee performance and skills also have the potential for creating a stressful and unproductive environment for employees.

Similar to higher education, a paradigm shift needs to occur in human resource performance and skills assessment techniques being used in the industry. The most important thing to be kept in mind before implementing any individual or company-wide performance initiative is to lay the framework based on the missions and objectives of the organization. The goal of the performance assessment initiative should be ‘Assessment for Individual and Organizational Performance Improvement’ rather than just ‘Assessment of Individual Performance’.

When employees see the assessment of their performance unrelated to the job requirements and role expectations, it just becomes an external judgment which doesn’t help and might even do a lot of damage to their productivity. This is also the case if they have little or no say in the whole evaluation process and are unclear about the performance criteria. Too often such initiatives and assessments are just seen by employees as another tool for management to control them. The threat and anxiety of an evaluation can prevent an employee from becoming engaged in a project and can dampen the enthusiasm needed for successful projects. In contrast a successful employee assessment program can supplement employee achievement and enhance productiv-
ity. If employees can see value in each aspect of their performance assessment and they know that they will receive useful and constructive feedback, helping them grow within the organization then they are less likely to fear such performance evaluation.

Although the dynamics of the human capital will keep evolving, employee performance and organizational success will always be linked. The ability to maximize the performance and productivity of the human assets of a company relies on the ability to accurately measure their skills and examine their weaknesses. The challenge in developing a successful performance assessment program is to collect the ‘invisible data’ meaning the hundreds of discrete human attributes and behaviors that are called in for fulfilling a specific role in a company. For comparing and contrasting, behaviors may be organized into ranges of effectiveness or proficiency and ranked in terms of importance to job performance. For identifying the attributes and traits needed to efficiently perform a job a multi method approach involving various tools should be undertaken. This should include job observation, reviewing existing documentation regarding job requirements, job training materials, structured questionnaires and interviews with the immediate managers to understand the business requirements of a particular job. Once the job requirements, attributes and behaviors have been identified a sound measurement system can then provide the data needed to explore and determine the relationship between these values and the achievement of business goals. This provides a standardized method for rating on the job behaviors like quality, skills, knowledge, accountability and flexibility. Figure 4 gives an example of such a standardized assessment that can be used for rating an employee on attributes like work quality, dependability, initiative, flexibility, skill building, job knowledge, punctuality and supervisory ability. Such assessments should be designed to accurately collect as much data on behaviors and job outcomes as possible. Ultimately the assessment should reflect the breadth of the business requirements from a specific role. Rather than having a standard assessment for every employee they should be
tailored for specific categories of employees such as part time, full time, exempt, non exempt, new hires, interns, temporary consultants and contractors. Self and peer assessments can also be used to promote formative activities among employees allowing them to become part of the performance evaluation process. Care should be taken to keep the peer assessment activities constructive and promote increased participation within the team, resulting in constructive and valuable feedback.

Formative feedback should also be gathered from employees and managers at various levels on the effectiveness of such forms and necessary adjustments which might be needed over-time should be made. Continuous monitoring of the outcome data should also be done to determine the effects of the intervention. It may be desirable to couple the performance metrics program to an awards program for recognizing excellence. Identifying the attributes and behaviors needed to perform a job can also help a company streamline the recruitment process by evaluating and selecting only those candidates which reflect those attributes.
Employee name: __________________________

Employee title: __________________________

Period covered: __________________________

Date of evaluation: ________________________

<table>
<thead>
<tr>
<th></th>
<th>Excellent</th>
<th>Very good</th>
<th>Good</th>
<th>Fair</th>
<th>Unsatisfactory</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>Work quality</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Dependability</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Initiative</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Flexibility</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Skill building</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Job knowledge</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Punctuality</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Supervisory ability</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

General comments on employee’s performance:

Employee’s goals for the coming year:

Reviewed by: __________________________ Signature: __________________________

Figure 4: Employee Performance Assessment Sample
To offset concerns regarding any metrics initiative, continuous feedback should be collected at various levels within the organization. The leadership vision and strategy for implementing the metrics program should also be communicated clearly at all levels within the organization. This would also permit a more accurate analysis of the correlation between each individual performance and overall corporate performance. Using proper feedback mechanisms all employees should be able to recognize how they can directly influence the measurement results. With proper employee buy-in into the assessment metrics program the employees will push to achieve the desired measurement goals.

Encouraging feedback can also fail to motivate if the employees feel that post-assessment, there is no opportunity for meaningful growth and improvement. If after the performance evaluation certain knowledge areas needing improvement are identified in an individual then adequate steps should be taken for developing those skills. This can include conference registration, formal training, funding for advanced education, self directed training or on the job training. Feedback mechanisms should also be put in place to gather a consensus on the benefits of such activities from the participating individuals. If they are not deemed as beneficial then the organization should stop investing in that development activity. As knowledge and skill development activities do not produce immediate and visible return on investment, there might be a pushback from the executive management for supporting such an initiative. On the other hand job specific training might be accompanied with expectations for immediate performance improvement. The human resource department should make sure that executive management has reasonable expectations for any specific skill development initiative. Applied correctly, such skill development initiatives will in time increase the competence and expertise of employees leading to improved individual and organizational performance.
CHAPTER 8

SUCCESSFUL METRICS PROGRAMS IN THE INDUSTRY

This section provides as an example, the successful metrics program implemented in Motorola using the GQM paradigm as described in section 5. Motorola’s software metrics program is well documented by Daskalantonakis [20]. By following the Goal/Question/Metric paradigm of Basili and Rombach [21], goals were identified, questions were formulated in quantifiable terms and metrics were established. The goals and measurement areas identified by the Motorola Quality Policy for Software Development (QPSD) are listed in the following.

Goals

- Goal 1: Improve project planning
- Goal 2: Increase defect containment
- Goal 3: Increase software reliability
- Goal 4: Decrease software defect density
- Goal 5: Improve customer service
- Goal 6: Reduce the cost of nonconformance
- Goal 7: Increase software productivity

Measurement Areas

- Delivered defects and delivered defects per size
- Total effectiveness throughout the process
- Adherence to schedule
• Accuracy to schedule
• Accuracy to estimates
• Number of open customer problems
• Time that problems remain open
• Cost of nonconformance
• Software reliability

For each goal the questions to be asked and the corresponding metrics were also formulated. In the following, we list the questions and metrics for each goal

8.1 Goal 1: Improve Project Planning

Question 1.1: What was the accuracy of estimating the actual value of project schedule?

Metric 1.1: Schedule Estimation Accuracy (SEA)

\[ SEA = \frac{\text{Actual project duration}}{\text{Estimated project duration}} \]

Question 1.2: What was the accuracy of estimating the actual value of project effort?

Metric 1.2: Effort Estimation Accuracy (EEA)

\[ EEA = \frac{\text{Actual project effort}}{\text{Estimated project effort}} \]

8.2 Goal 2: Increase Defect Containment

Question 2.1: What is the currently know effectiveness of the defect detection process prior to release?

Metric 2.1: Total Defect Containment Effectiveness (TDCE)
TDCE = (Number of prerelease defects) / (Number of prelease defects + Number of postrelease defects)

Question 2.2: What is the currently known containment effectiveness of faults introduced during each constructive phase of software development for a particular software product?
Metric 2.2: Phase Containment Effectiveness for phase i (PCE)
PCEi = (Number of phase i errors) / (Number of phase i errors + Number of phase i defects)

8.3 Goal 3: Increase Software Reliability
Question 3.1: What is the rate of software failures, and does it change over time?
Metric 3.1: Failure Rate (FR)
FR = (Number of failures) / (Execution time)

8.4 Goal 4: Decrease Software Defect Density
Question 4.1: What is the normalized number of in-process faults, and how does it compare with the number of in-process defects?
Metric 4.1: In-process Faults (IPF)
IPF = (In-process faults caused by incremental software development) / (Assembly-equivalent delta source size)

Question 4.2: What is the currently known defect content of software delivered to customers, normalized by Assembly-equivalent size?
Metric 4.2a: Total Released Defects (TRD) total
TRD Total = (Number of released defects) / (Assembly-equivalent total source size)
Metric 4.2b: Total Released Defects (TRD) delta

TRD Delta = (Number of released defects caused by incremental software development) / (Assembly-equivalent total source size)

Metric 4.2c: Customer-Found Defects (CFD) total

CFD total = (Number of customer-found defects) / (Assembly-equivalent total source size)

Metric 4.2d: Customer-Found Defects (CFD) delta

CFD delta = (Number of customers – found defects caused by incremental software development) / (Assembly-equivalent total source size)

8.5 Goal 5: Improve Customer Service

Question 5.1 What is the number of new problems opened during the month?

Metric 5.1: New Open Problems (NOP)

NOP = Total new post release problems opened during the month

Question 5.2: What is the total number of open problems at the end of the month?

Metric 5.2: Total Open Problems (TOP)

TOP = Total post release problems that remain open at the end of the month

Question 5.3: What is the mean age of open problems at the end of month?

Metric 5.3: Mean Age of Open Problems (AOP)

AOP = (Total time post release problems remaining open at the end of the month have been open) / (Number of open post release problems remaining open at the end of the month)
Question 5.4: What is the mean age of the problems that were closed during the month?

Metric 5.4: Mean Age of Closed Problems (ACP)

\[ ACP = \frac{\text{Total time post release problems closed within the month were open}}{\text{Number of open post release problems closed within the month}} \]

8.6 Goal 6: Reduce the cost of NonConformance

Question 6.1: What was the cost to fix postrelease problems during the month?

Metric 6.1: Cost of Fixing Problems (CFP)

\[ CFP = \text{Dollar cost associated with fixing post release problems within the month} \]

8.7 Goal 7: Increase Software Productivity

Question 7.1: What was the productivity of software development projects (based on source size)?

Metric 7.1a: Software Productivity total (SP total)

\[ SP \text{ total} = \frac{\text{Assembly-equivalent total source size}}{\text{Software development effort}} \]

Metric 7.1b: Software Productivity delta (SP delta)

\[ SP \text{ delta} = \frac{\text{Assembly-equivalent delta source size}}{\text{Software development effort}} \]

8.8 Conclusion and Future Work

Establishing a successful measurement program in an organization is no small endeavor. Development and implementation of any metrics program involves paying close attention to sev-
eral categories of critical success factors as discussed in the previous sections. Furthermore, many psychological and personnel issues have to be addressed to increase chances of success from such a program.

For this reason, a business-goal oriented measurement plan as discussed in this research can greatly help structure and provide rigor to the processes being followed in an organization. In the ever-changing, globally competitive, knowledge based software industry, a well designed metrics program will provide an organization with information to work productively, effectively and with a high level of quality.

In this paper, we have provided practical guidelines to the steps we believe are crucial for achieving such a measurement process. We have also discussed recent initiatives in the field of classroom assessment and how those techniques can be successfully applied to motivate employees resulting in increased individual and organizational performance. Future work involves formalizing better structure and content of the measurement plan so that better automated support can be provided. This will reduce the complexity involved in metrics collection, resulting in more effective metrics programs.
BIBLIOGRAPHY


14. Deissenboeck, Wagner, Pizka, Girard: An Activity-Based Quality Model for Maintainability


22. W. W. Royce: Managing the development of large software systems.


