A Framework for Instituting Software Metrics in Small Software Organizations

Hisham M. Haddad, Nancy C. Ross, and Donald E. Meredith

Computer Science Department, Kennesaw State University (USA)
E-mail: hhaddad@kennesaw.edu

ABSTRACT

The role of metrics in software quality is well-recognized; however, software metrics are yet to be standardized and integrated into development practices across the software industry. Literature reports indicate that software companies with less than 50 employees may represent up to 85% of the software organizations in several countries, including the United States. While process, project, and product metrics share a common goal of contributing to software quality and reliability, utilization of these metrics has been minimal. While the well-known process models may not be ideal in a relatively small setting, some of the large process models are being scaled down while, simultaneously, new models are being developed specifically for smaller software organizations. Software metrics have been studied for years, and there are many options available for metrics utilization, either within or outside of a process improvement framework, regardless of the size of the setting. Under this work, case studies and industrial reports on the use of metrics in software organizations were gathered and analyzed. This work examines the practices of metrics in the software industry with emphasis on small organizations, explores the challenges and benefits of using software metrics in small organizations, and outlines a practical framework based on the Goal/Question/Metric paradigm for instituting metrics programs in small software organizations.

Keywords: Cost of Defects, Framework for Software Metrics, Software Metrics Challenges, Software Metrics in Small Organizations.

1- INTRODUCTION

It is yet to be widely recognized that metrics are valuable tools for a software organization. They provide measurement about schedule, work effort, and product size among many other indicators. The more they are utilized, the more effective and productive the organization becomes. They also provide better control over projects and a better reputation for the organization and its business practices. Software metrics are utilized during the entire software development life cycle. Gathered data is analyzed and evaluated by project managers and software developers. The practice of metrics involves Measures, Metrics, and Indicators. A *Measure* is a way to appraise or determine by comparing to a standard or unit of measurement, such as the extent, dimensions, and capacity, as data points. The act or process of measuring is re-

ferred to as Measurement. A *Metric* is a quantitative measure of the degree to which a component, system, or process possesses a given characteristic or attribute, and an *Indicator* represents useful information about processes and process improvement activities that result from applying metrics, thus, describing areas of improvement.

The literature indicates that, throughout the world, most software applications are developed by small organizations ranging from one to fifty employees. Such organizations represent about 85% of software organizations in several countries, including the United States [1,2,3,4]. Since small software organizations make up such a large percentage of all software companies in the world, it follows that they have had a large influence in creating the software crisis that has existed since software development began. The software crisis states that software projects are characteristically behind schedule and over-budget, the software produced is difficult to maintain and is of low quality and efficiency, and most large software products are never used or are only partially used because customers' requirements are not fully met. Software process improvement and a program of metrics would help to lessen the effects of the software crisis by guiding a small organization to produce higher quality software.

The major objective for deploying a software metrics program is to improve the software development process. Historical data gained from the collection of metrics from all the projects developed by an organization is a wealth of information that will aid in the estimation and planning of future projects. Although small organizations have many resource challenges when attempting to implement a software metrics program, it is possible for an effective measurement program to be developed on a smaller scale. Managers and developers must determine what to measure, how to measure it, and how to transform this raw measurement data into information that can assist the organization in achieving its goal of producing better software [5].

Small software organizations face a wide range of challenges as far as collecting data and utilizing software metrics. Initial upfront investment is the most challenging of all as the cost of deployment and personnel are great; however, managers and developers alike are well aware of the necessity to strive to achieve quality standards that eventually lead to lower project cost and desired benefits that may be gained in the long term [6]. Another difficulty is staff participation. Managers and developers understand the high-level goals of software process improvement and metrics programs but often are unwilling to expend the necessary effort for various reasons, such as the fear of consequences, concerns regarding assessment of personal performance, the perception that metrics do not accurately represent their development methodology, and misrepresentation of the amount of effort they put into their work [7].

Can the small software organization overcome these challenges? How can an efficient process be created that will produce valid, actionable information

while at the same time getting all of the “regular” work done, and how can management be convinced that this is a long-term win for everyone? Why should the small software organization take the time and effort to implement a software metrics program? This work builds on previous work [8,9,10]. It explores these questions and presents a four-step framework to help the small organization take the first steps toward implementing a productive metrics program. Available options and opportunities for each step are also presented.

2- STATE OF METRICS

The area of software measurement has been highly active for several decades. As a result, there are many commercial metrics tools available on the market. Such affordable metrics can be the starting point for small organizations; however, much more work is needed to standardize, validate, and integrate metrics into software practices. This work is motivated by the cost of defects, measurable characteristics of software, and common metrics available for small organizations.

2-1 COST OF DEFECTS

To present a convincing argument regarding the benefits of using metrics, one needs to highlight the incentives and payoff. William T. Ward [11] describes Hewlett-Packard’s (HP’s) “*10 x software quality improvement*” initiative. The author uses data from a software metrics database and an industry profit-loss model to develop a method to compute the actual cost of software defects. The database is an important element of HP’s software quality activities and is a valuable source for different tasks such as quality status reporting, resource planning, scheduling, and calculation of cost defects. Sources of data include product comparisons, analysis of source code size and complexity, defect logging, project post-mortem studies, and project schedule and resource plans.

The Software Quality Engineering Group follows definite steps to discover, correct, and retest a defect during testing activities (integration, system, and/or acceptance). The estimated effort here is about 20 hours, and it represents the average effort for discovering and fixing a defect. This effort is calculated using data points from multiple projects that were tracked with the software quality database. Defect cost can also be determined per project or phase, and cost can be weighted based on programmer productivity or product code size. For instance, the following formula shows the cost per defect that is discovered and fixed during the integration through the release phases of a project.

Software Development Cost = SDRC + PL
where:

SDRC (Software Defect Rework Cost) is determined by the amount of effort and expense required to find and fix defects during the integration through release phases and
PL (Profit-Loss) is the revenue loss caused by lower product sales throughout the entire post release lifetime.

To illustrate, a product has about 110 software defects found and fixed during testing. Each defect requires 20 engineering hours to identify and fix. The total work effort is 2200 hours. At $75/hour, SDRC is $165,000, and the rework cost per defect is $1500. These expenses could be saved had metrics been used to mitigate those defects. In addition, it should be noted that the other calculation for defect cost is product profit-loss. Here, missed market-window opportunities result in loss of sales, profits, and competitiveness. This illustrates typical losses that result from the lack of metrics utilization.

2-2 MEASURABLE CHARACTERISTICS

Unlike software engineering, other disciplines capitalize on the power of quantitative methods to measure their processes and activities. Based on this statement from Tom DeMarco [12] - "*You can't control what you can't measure*", these disciplines apply measurements to gain better control of their projects and quality of products. Although software engineering is a new and evolving discipline, experts have proposed quantitative methods applicable to all aspects of software projects with the goal of achieving high quality products. These methods are related to different measurable characteristics of software applications. Measurement-related activities include the following:

Cost and effort estimation: Estimation models [13] help to better plan and execute software projects. One factor that plays into the success of applying estimation models is the experience of the organization in predicting the effort and cost of new software systems. Mathematical models, such Boehm's COCOMO [14], Putnam's SLIM [15], and Albrecht's Function Points [16], can be used.

Productivity measures: Productivity models focus on the human side of a project. A key factor in accurately determining productivity is gathering sufficient information about the productivity of an individual (or a team) in different scenarios, such as the type of project, team structure, skills and backgrounds, tools, and environment. Measures and metrics for assessing the human side of the project are more challenging to develop and apply than other measures and metrics [5].

Data collection: Data collection is an important discipline, requiring diligence and careful implementation. Although it has obvious benefits for developing measures and metrics, team members often dislike it. The common perception among some team members is that data collection leads to an uneasy feeling of being "under pressure" and "at risk" as collected data can be nega-

tively used in performance evaluations. The real risk here is that inaccurate data can result in metrics that provide false assessments.

Quality assessment: This activity covers different measures including efficiency, reliability, flexibility, portability, usability, and correctness as well as many other measures. Standards that define quality in terms of specific project goals are needed. Here and with historical data, objectives (in terms of measures) should be achieved or exceeded in order to meet desired quality standards. Although quality assessment is often applied early in the life cycle, it covers, along with "umbrella activities", the entire life cycle [5].

Reliability models: Even though reliability is seen as a quality attribute, reliability assessment models are more related to software failures and are mostly applied during testing. The models work well when it is possible to monitor and trace failures during a test or operation. Many quality models use reliability as a factor, and the concept of reliability weighs much as far as the perception of quality.

Other activities include Performance evaluation for optimal solutions, Structure and complexity, Capability maturity assessment, Management by metrics, and Evaluation of methods and tools. These activities are becoming an important part of Software Engineering as each activity leads to the development of software metrics, some of which evolve into assessment models.

2-3 COMMON METRICS

Many commercially-available software metrics products have been developed over the years. As they vary in complexity and sophistication, many products are "affordable" for resource-constrained organizations. Process, Project, and Product are three common categories that offer a variety of metrics for small organizations.

Process Metrics: These metrics focus on software development and maintenance. They are used to assess people's productivity (known as private metrics), productivity of the entire organization (known as public metrics), and software process improvement. Process assessment is achieved by measuring specific attributes of the process, developing a set of metrics based on the identified attributes, and, finally, using the metrics to provide indicators that lead to the development of process improvement strategies. Private metrics are designed to help individual team members in self-assessment allowing an individual to track work tasks and evaluate self-productivity. Public metrics, on the other hand, help evaluate the organization (or a team) as a whole, allowing teams to track their work and evaluate performance and productivity of the process. A good example is a team's effectiveness in eliminating defects through development, detecting defects through testing, and improving response time for fixes.

Project Metrics: Project metrics are tactical and related to project characteristics and execution. They often contribute to the development of process metrics. The indicators derived from project metrics are utilized by project managers and software developers to adjust project workflow and technical activities. The first application of project metrics often occurs during the cost and effort estimation activity. Metrics collected from past projects are used as the basis from which effort and time estimates are made for new projects. During the project, measured efforts and expended time are compared to original estimates to help track how accurate the project estimates were. When the technical work starts, other project metrics begin to have significance for different measures, such as production rates in terms of models created, review hours, function points, and delivered source code lines. Common software project metrics include:

- Order of growth: This metric is a simple characterization of an algorithm‘s efficiency allowing comparison of the relative performance of alternative algorithms without being focused on the implementation details.
- Lines of code: The physical type is a count of lines including comment and blank lines (not to exceed 25% of all lines of code). The logical type is a count of the number of "statements" tied to a specific programming language.
- Cyclomatic complexity: This metric measures the application complexity and describes its flow of control.
- Function points: This metric reflects functionalities relevant to (and recognized by) the end user. It is independent of implementation technology.
- Code coverage: This metric determines the statements in a body of code that have been executed through a test run and those statements that have not been executed through a test run [17].

Other project metrics include coupling, cohesion, requirements size, application size, cost, schedule, productivity, and the number of software developers.

Product Metrics: These metrics focus on measuring key characteristics of the software product. There are many product metrics applicable to analysis, design, coding, and testing. Commonly-used product metrics include:

- Specification metrics: These metrics provide an indication of the level of specificity and completeness of requirements.
- Size metrics: These metrics measure the system size based on information available during the requirements analysis phase.
- Architectural metrics: These metrics provide an assessment of the quality of the architectural design of the system.
- Length metrics: These metrics measure the system size based on lines of code during the implementation phase.
- Complexity metrics: These metrics measure the complexity of developed source code.

- Testing metrics: These metrics measure the effectiveness of conducted tests and test cases.

Other product metrics focus on design features, quality attributes, code complexity, maintainability, performance characteristics, and code testability among others.

3- THE CHALLENGES

"In the late 1980s, two-thirds of all [software process improvement] SPI programs faltered or failed after the initial assessment due to flawed strategy, lack of commitment, lack of follow-through, not measuring improvements, and lack of crisp SPI objectives tied to business objectives" [18]. Small software organizations face many challenges when implementing software metrics. Among those issues are implementation cost, reluctant cooperation, fear of individual consequences, dishonesty in reporting metrics, differences in reporting and use of process and product metrics, lack of experience in the SPI process, lack of a champion for SPI, lack of organizational commitment, and inexperienced management. This section highlights most common challenges that small organizations face.

3-1 IMPLEMENTATION COST

Throughout the world, most software applications, including outsourced applications, are developed by small software organizations. Most of these organizations have a small number of employees with a high level of specialized training who usually take on many roles and work on multiple projects concurrently. Often, these organizations are owner-financed with a small amount of capital. They find it difficult to bring in other investors and, therefore, have only limited financial resources [19]. Most of the resources are invested in business requirements, and few resources are available to devote to a robust measurement program. Such businesses are not able to incur the cost of software process improvement models and standards such as CMM, SPICE, and ISO, because deployment and personnel costs are great. Due to lack of monetary resources, it is not possible for a small organization to hire a consultant to suggest software quality improvements; however, managers are aware of the necessity to strive to achieve quality standards that eventually lead to lower project cost. Costs to the organization are immediate when implementing a metrics program; however, it is inevitable that benefits will be gained in the long term.

3-2 RELUCTANT COOPERATION

When developers and managers feel that they will gain nothing from participation in the measurement process, great resistance to provide accurate raw data is encountered. Lack of communication of the objectives of the mea-

surement program as well as a choice of metrics that are too complex can contribute to this opinion. Developers also feel that the implementation of a measurement process increases the burden of documentation and keeps them from accomplishing their tasks [20]. Many times, developers do not feel that metrics data is precise. They are not motivated to collect accurate data because they think that the information will be changed by their managers. Also, personal opinions about the measurement program (perhaps developed through previous experience with such programs in the same or other companies) circulate through "the grapevine," and this influences programmers' thoughts about the newly-implemented plan [21].

Some developers and managers feel that a measurement process is unnecessary, but they are forced to collect data by their superiors. Because of a negative attitude toward the metrics program, they never realize how much their own work might improve from the analysis of the information gathered, nor do they suggest other measurements that might be necessary in order to develop a true picture of their work. In this case, the quality of the metrics data is poor, and the data collected is incomplete.

The standardization of metrics across the organization may generate the feeling that the measurement data does not fairly represent the effort that went into the product and, thus, unfairly depicts the amount of work that was done. Personnel may lose confidence in the metrics program if a measure does not indicate the difference in effort put forth by various groups such as development and maintenance.

Managers feel that developers are not receptive to the collection of metrics, and this attitude affects the perception of the developers. A manager's attitude toward the metrics process has a big influence on the acceptance of the program by developers, and if management does not realize that the organization can reap benefits from the collection of metrics, then the developers are not likely to realize this either.

Business managers tend to be skeptical about the measurement process. They require proof that they will see a quick return on investment before they are completely convinced of the value and usefulness of such a program. Because of this, it is imperative that improvements are implemented in key process areas that will provide a visible payback quickly. Measurable benefits should be seen within a year from the start of the program or business management will not support software process improvement programs [20,22].

From experience gained in a study of a start-up software company, Kajko-Mattsson and Nikitina found that the most challenging obstacle to implementing software process improvement was to encourage employees to change their habits. Developers did not use the change management tool when the study began; however, they implemented the tool as the study went on. At this same company, management changed requirements and assignments spontaneously without thought to the effect on project milestones, and this beha-

vior never changed. Many of the process improvement actions that Kajko-Mattsson and Nikitina attempted to implement could not be easily completed due to employees' lack of motivation to improve the process [23].

3-3 FEAR OF INDIVIDUAL CONSEQUENCES

According to Umarji [7], managers and developers are willing to collect and report metrics when the data shows them to be successful, but when metrics data identifies a problem, those same managers and developers will argue that the measurement does not accurately represent their development methodology. People tend to believe that their work is good and are heavily invested in the projects to which they are assigned, so it offends them to see metrics information misrepresent the amount of effort they put into their work. These employees want to ensure that their reputation is not adversely affected by the results of the metrics program for several reasons:

- Developers are proud of their code. The identification of bugs through peer reviews is embarrassing. Developers are afraid that the metrics they provide will be used to assess their personal performance; therefore, they are sensitive about the information that is reported to the organization, especially errors and defects found in their own code.
- Not only are developers concerned about their own well-being in the organization, but they are also concerned about the well-being of their friends and team members. When asked to do peer reviews, managers and developers sometimes report only the most obvious errors. Because software development requires such close personal interaction at times, the reporting of metrics data could make it difficult for peers to work together, and this may have a negative impact on the quality of the final product.

3-4 DISHONESTY IN REPORTING METRICS

Dishonesty in reporting metrics is a common practice. Employees may not mind reporting metrics, but they may resist the process of standardization of metrics. The standardization of processes and metrics may cause the project team to think that the metrics group is going to make comparisons between various projects. Because of this, they are hesitant to share accurate data because they are concerned about unfair comparisons. There must be a common definition of metrics if comparisons are going to be made, but in the world of software metrics, it is very difficult to develop a standard definition.

In some cases, scripts are written to enter data into metrics reports in order to present the appearance of collecting metrics without actually having to report them manually. The developers complete the required process including unit testing but write scripts to fill out metrics reports because the manual collection of the measurement data is too complex. Scripts can also be used to enter incorrect information intentionally. The intentional reporting of false metrics

data is a common problem, and if it happens on a small scale, it may never be noticed [7].

3-5 LACK OF EXPERIENCE IN PROCESS IMPROVEMENT

Organizations that have never implemented process improvement programs require the help of a consultant when defining their first effort into the improvement program; however, the hiring of a consultant is often difficult because of financial reasons. Software engineering and project management are the common areas of the process in which small organizations are deficient. In order to foster the development of this knowledge in employees, consultants or more experienced employees in the organization may mentor other employees to emphasize the benefits of improvement programs and the necessity of commitment to the organization's improvement goals [22]. As the process improvement program progresses, the employees of the organization absorb the skills needed to define the next increment of process improvement which may be closer to the organization's needs. As the process becomes more and more established, the organization's commitment to process improvement increases.

3-6 LACK OF A CHAMPION FOR PROCESS IMPROVEMENT

If an organization is to change its performance, then senior management must support the recommended changes. In most small organizations, although the business founder is the main role model, a respected developer or technical leader may have enough influence with upper management to convince everyone that a measurement program is necessary. In this case, the "champion" is responsible for making the organization aware of process improvement and for encouraging the sharing of knowledge among employees. Such an effort is necessary in order for improvement to take place [23,24].

3-7 LACK OF ORGANIZATIONAL COMMITMENT

Organizational commitment is imperative at all management levels and at all stages of the process improvement program. In some organizations, it is difficult to obtain commitment to process improvement because only the costs of the improvement program are recognized. The benefits of the program must also be realized, and in order for this to occur, measures of time, cost, quality, and customer satisfaction must be communicated to upper management throughout the process [22].

3.7 LACK OF PROJECT MANAGEMENT SKILLS

Often, a respected technical person is thrust into the project management role. In such a case, the technical expert has little, if any, project management experience or training. Good software process improvement and management practices are not known, so they cannot be applied [21].

4- PROCESS IMPROVEMENT FRAMEWORKS

Metrics programs are tied to process improvement practices. Although small organizations are aware of existing process assessment models, they often assume that such assessments can be expensive and time-consuming, and therefore, difficult to perform in such organizations. Furthermore, it is believed that assessment models and standards, including documentation and process standardization, are specifically for large organizations. Such procedures are seen as inappropriate for small settings which normally have informal processes and are focused on getting their product to market in order to survive [4].

The features of a process improvement framework are: 1) the definition of a set of characteristics that must be present if an effective software process is to be implemented, 2) a way to determine whether those characteristics are already in place, 3) a report of the results of the assessment, and 4) a plan for assisting the software organization to put those process features that are weak or missing into place.

Assessment-based process improvement models such as CMM, CMMI, SPICE ISO/IEC 15504, and ISO 9001 are based on formal frameworks which support the use of systematic processes and management practices for software engineering [18]. Statements of purpose for these formal models are outlined in Table 1.

Table 1 Formal Models for Software Process Improvement.

Model	Statement of Purpose
CMM	The Software Engineering Institute's (SEI) Capability Maturity Model (CMM) is a method of software process improvement. Organizations implement this methodology to help assessment teams find strengths and weaknesses in the software improvement process and to help managers and technical staff determine the best ways to improve that process. The organization must study itself using the concepts, goals, and activities defined by the CMM. A maturity scale provides a picture of process quality that can be used by developers and managers as a measure from which improvement strategies can be planned.
CMMI	CMM evolved into CMMI (Capability Maturity Model Integration) in 2002 as a framework to recognize and improve processes within an organization. Its maturity levels are steps that an organization can achieve in order to increase its value in the marketplace. CMMI is a complete process framework that is based on system and software engineering capabilities that should be present as organizations reach different levels of process capability and maturity. Specific goals and practices are assessed in each process area and rated according to pre-defined capability levels (Incomplete (level 0), Performed, Managed, Defined, Quantitatively Managed, and Optimized (level 5)). CMMI defines a complete software process in great detail covering twenty-two key process areas.
ISO/IEC	ISO/IEC 15504 is an international standard that presents a framework

15504 (SPICE)	for the assessment of process improvement and capability determination. This standard has five parts: Concepts and Vocabulary, Performing an Assessment, Guidance on Performing an Assessment, Guidance on Use for Process Improvement and Process Capability Determination, and An Exemplar Process Assessment Model. Process assessment is based on a capability dimension and a process dimension. The Capability Dimension Scale has six levels (Incomplete (level 0), Performed, Managed, Established, Predictable, and Optimizing (level 5)). The process dimension identifies a group of processes on which the assessment is performed and is based on one or more external process reference models which define a set of universal processes. By choosing the particular processes that are most important to an organization, the standard can be flexibly applied.
ISO 9001	ISO 9001 emphasizes the importance of identification, implementation, management, and continual improvement of the processes that are necessary for a quality management system. It also stresses the importance of the management of process interaction in achieving the organization's goals. Process effectiveness and efficiency can be assessed internally or externally and be measured on a maturity scale.

Over the years, researchers have done much work in creating informal process improvement approaches that make the formal models more feasible for implementation in small organizations. As a result, a number of informal methods have been developed for small organizations including PSP, TSP, PRISMS, ADEPT, BOOTSTRAP, MARES, RAPID, SPINI, FAME, TOPS, PROCESSUS, and GQM. Statements of purpose for these methods are described in Table 2.

Table 2 Informal Methods for Software Process Improvement.

Method	Statement of Purpose
PSP/TSP	PSP was designed to help software engineers continually do better work based on personal performance data, and TSP was designed to build effective, self-directed, self-improving teams.
PRISMS	The PRISMS model emphasizes the use of business goals for the selection of key process areas for assessment, the involvement of both upper management and developers, and the use of quantitative measurement.
ADEPT	ADEPT uses both plan-driven and agile methods to develop an improvement-based SPI program focused on a company's business goals.
BOOTSTRAP	The purpose of BOOTSTRAP is to identify process strengths and weaknesses, to support improvement planning with suitable and reliable results, to plan improvement steps that will meet the organization's goals, and to implement standard requirements that will increase process effectiveness.
MARES	MARES is a method that provides guidance in identifying target process profiles and selecting high-priority processes to assess on the basis of an organization's business goals and model.
RAPID	The RAPID assessment is limited to one day and must be performed by ISO/IEC 15504 assessors. Eight key process areas are examined, key risks and improvement opportunities are identified, and action recommendations are made.

SPINI	SPINI is a SPICE-compatible assessment for small organizations with the goal of process improvement. SPINI analyzes the needs of the organization, performs process assessment, and produces a software process improvement plan. There is no monitoring, control, or feedback on the progress of the assessment.
FAME	FAME allows the execution of either a SPICE or BOOTSTRAP assessment focusing on improvement and, for small organizations, can be accomplished in a one-day workshop. FAME assessments are focused on business goals and requirements and utilize a GQM-based measurement program.
TOPS	TOPS is an ISO/IEC 15504-conformant assessment model that is based on three standard process areas and focuses on process improvement in order to promote innovation.
MPS Model	The MPS Model uses the principles of software engineering according to the main international approaches for software process definition, evaluation, and improvement in order to develop a generic model appropriate for small and medium software enterprises that need to make improvements in software processes in a short amount of time and at a low cost.
MoProSoft	MoProSoft is a software engineering process model based on CMM, ISO 12207, and ISO 9001 with the purpose of encouraging the use of standards in Mexico by using best practices in software engineering. The model is based on an organizational structure of top management, management, and operations and the steps to be performed by specified roles.
PROCESSUS	PROCESSUS performs an assessment using six processes based on the CMM and ISO 9001: customer relationship management, project management, software engineering, supporting activities, process management, and process automation.
Goal-Question-Metric	Goal-Question-Metric (GQM) is a methodology used to develop a metrics program based on the goals of the organization.

5- METRICS ACHIEVABLE BY SMALL ORGANIZATIONS

Few small organizations will implement sweeping metrics programs, but it is necessary that they measure and use metrics to improve the software process and products. The three purposes of a metrics program are: 1) to know the organization's capability or benchmark; 2) to make achievable estimates; and 3) to manage development in order to stay on schedule [24]. Appropriate metrics that do not require high overhead and upfront investment do exist. It is best for a small organization to start with a few easily-gathered measures that direct the focus of the organization to areas that will increase the possibility of achieving stated goals [25,27,28]. For such organizations, the most useful metrics are those most easily gathered, such as metrics related to the individual programmer, the project team, and the organization.

5-1 COST METRICS

Project cost may be one of the most easily-determined measures at the end of the project. Using gathered schedule data, both actual and estimated, the cost of the project is determined based on time information and salaries. Other project costs should be determined and included in the project cost. Cost metrics include comparisons of estimated and actual costs [26]. Management indicators within return-on-investment models of process improvement include measures of product quality, process quality, project predictability, and customer satisfaction; however, some of the biggest payoffs of process improvement are better job satisfaction, pride in work, an increased ability to attract and retain experts, and a reputation for excellence [22].

5-2 SIZE METRICS

Software sizing is most commonly predicted using lines of code (LOC) or function points (FP). LOC is very common in software engineering organizations while FP is more popular in information systems organizations. LOC is based on an internal view of the system as seen by the developer while FP takes into consideration an external view of the system as seen by the user [26]. Other ways to size software are to use object classes, requirements, user stories, or elements of the graphical user interface [27,24]. These counts are typically adjusted for complexity.

5-3 SCHEDULE METRICS

Each member of the project team should record daily activities (effort/time) on a simple tracking form so that the reporting becomes a habit. Perhaps ten minutes a day should be devoted to this activity [28]. The chosen measurements should be easily gathered so they do not take a lot of extra time. Spreadsheets and charts can be used to accumulate the information into a metrics database. Through the use of spreadsheets, the data can be sorted, and calculations can be made as needed [29]. Examples of schedule metrics include comparisons of actual completion dates to estimated completion dates. Schedule data may also be related to software size and cost information in order to determine the productivity of the project and the business.

5-4 QUALITY METRICS

Errors and defects are found throughout the software development process. Generally, errors and their severity are measured during system testing before the system is delivered to the customer, and defects are measured after the system is delivered to the customer. Defect distribution and the cost to find and fix defects are derived measures. The number of product failures is reported by end users through customer support and is an objective measure. The defect find rate (the number of defects found per hour by testers) helps to

determine the cost of testing and to evaluate how stable the system is. Defect densities should be recorded throughout the testing process [25].

In smaller organizations, complete and accurate information about errors is generally not collected until review and testing measurement practices have been in place for a while, but customer-found defects are reported regularly through user problem reports after the product has been installed. Process metrics are not as easily collected as product metrics, but they provide information that can be used to immediately improve the process and the result before the product is deployed. Product metrics, while easy to measure and readily available, do not provide data for process improvement until the end of the process [7,27].

For small software organizations, collecting easily-gathered measures and computing metrics is estimated to be from three to eight percent of the project budget at the beginning of a metrics program. After developers and project managers become familiar with the process, the cost drops to about one percent of the project budget. The organization can realize a substantial return on investment if the information gathered leads to meaningful process improvement [5].

Table 3 shows the measurements that are most helpful and easily-collected at the beginning of a metrics program for a small organization.

Table 3 Achievable Metrics for Small Software Organizations.

Category	Scope	Measurement
Cost	Individual and Project	Estimated Cost, Actual Cost
Size	Individual and Project	LOC/KLOC, FP, Object Classes, Requirements, User Stories, GUI Elements as appropriate for application
Schedule	Individual and Project	Estimated Duration, Actual Duration – time required to complete a task (individual) or the project in calendar units of time; Estimated Effort, Actual Effort; Work Effort Distribution – time spent in various development or maintenance activities
Quality	Individual and Project	Number, type, severity, and status of defects found by testing, peer reviews, and customers

After the metrics program has been established, additional metrics may be added as the organization feels the necessity. Some of these more comprehensive metrics are found in Table 4.

Table 4 More Comprehensive Metrics Program for Small Software Organizations (Wiegers, 2007)

Category	Scope	Measurement
Cost	Individual	N/A
	Project	Estimated Cost, Actual Cost
	Organization	Estimated Cost, Actual Cost
Size	Individual	LOC/KLOC, FP, Object classes, Requirements, User stories, GUI elements as appropriate for application
	Project	Product size
	Organization	Productivity
Schedule	Individual	Work effort distribution among various development and maintenance activities; Estimated and actual duration; Estimated and actual effort
	Project	Estimated and actual duration between major milestones; Estimated and actual staffing levels; Number of tasks planned and completed
	Organization	Overall project cycle time; Planned and actual schedule performance; Schedule/effort estimation accuracy
Quality	Individual	Number and type of defects found by unit testing; Number and type of defects found by peer reviews
	Project	Number of defects found by integration and system testing; Number of defects found by peer reviews; Defect status distribution; Percentage of tests passed
	Organization	Released defect levels

6- IMPROVEMENTS REALIZED IN SMALL ORGANIZATIONS

The implementation of informal process improvement approaches for small organizations resulted in a number of changes as indicated in the case studies and industrial reports gathered and analyzed for this work. Among them, increased project management activities, improvements to the testing process, and implementation of risk management strategies were the most common. Improvements in requirements gathering change management, measurement, and additional areas were also realized from these approaches. This section highlights common improvements small organizations were able to achieve as a result of metrics programs. These examples were reported in the case studies analyzed for this work. They are an indication of potential improvements and benefits that small organizations may realize.

6-1 IMPROVEMENTS IN PROJECT MANAGEMENT

Systematic project management was introduced in several organizations. Focus was placed on measurement and collection of project data for planning

purposes. In some cases, Microsoft Outlook was used as a project tracking device at the task level, and timesheets were used as a way to capture project status. Organizations began to generate a knowledge base of historical data to guide them in project estimation and planning. Several valuable data items were collected regularly, and some key systems were modified to improve data collection. Better predictability and fewer time overruns were the result of the establishment of these systematic processes, and awareness of the importance of measurement as a source of objective information on project status was increased [1,4,23].

6-2 IMPROVEMENTS TO THE TESTING PROCESS

Some organizations were performing minimal software testing, so the establishment of systematic test practices improved the quality of their software. Unit testing, functionality testing, performance testing, and deployment testing were implemented in several companies that had not previously seen a great value in the testing process. Most companies reduced costs through fewer errors and less rework. In one organization, a tester was appointed, test plans were formulated, and test logs and incidents were recorded in order to control risks regarding testing. Improvement in testing procedures gave these companies more confidence in their product releases [4,23].

6-3 IMPLEMENTATION OF RISK MANAGEMENT STRATEGIES

Many reported examples of risk assessment improvements. A risk assessment and management procedure was implemented at one company. In another company, risks were routinely identified for all projects, and risk mitigation strategies were defined. In a third company, an employee attended a Risk Management training course, and, upon return, established a risk assessment and management plan. This had a great impact on the quality management system and pointed out the necessity to change such procedures as testing, contract review and planning, and requirements control [4,18]

6-4 IMPROVEMENTS IN REQUIREMENTS GATHERING

In one organization, developers misunderstood initial requirements and had to rework many of the implemented features. This delayed new feature development for over 40% of the planned time. As a result of their process improvement program, a requirements tracking method was implemented at this organization using a simple template for describing requirements, and this change reduced the new feature development delay to only 10% of the planned time. Another company saw better product quality and increased customer satisfaction due to improvements in their requirements gathering process [4,23].

6-5 IMPROVEMENTS IN CHANGE MANAGEMENT

Due to management's spontaneous changes/additions to requirements, business and support teams at one organization had no idea of which requirements were being implemented in a release, and the change requests were disorganized and undocumented in the change management tool. Also, developers did not update the status of the change requests they worked on, so the tool was never current and could not be used to track project status. Increased use and daily updating of the existing change management tool increased control over and documentation of change requests which provided the ability to view project status and progress on change requests [23]. In another company, an organization-wide change request system was developed and implemented, and a defect tracking and management tool was introduced and put into practice [18].

6-6 IMPROVEMENTS IN MEASUREMENT

The companies in the case studies reviewed had no formal measurement process at the beginning of the assessment process, so any measurement activity was an improvement to the existing process. In most companies, much more data was being collected, but analysis of that data was not always accomplished, so the measurements did not have any impact on project performance. Suffice it to say that the organizations became aware of the importance of measurement as a source of objective information regarding the software development process. As analysis of the data was introduced and a historical data repository was established, the benefit of greater data collection and analysis became evident to the organizations [18].

6-7 IMPROVEMENTS IN OTHER AREAS

Additional improvements were realized in the areas of communication, customer support, and defect management. At one company in which there was little verbal or written communication, daily stand-up meetings were implemented, and change request and requirement documentation processes were established [23]. Another company began holding regular postmortems. A step toward continuous knowledge management and process improvement activities, postmortems allow the project team and the organization to learn from the successes and failures of a project. In another company, a customer support process was put in place, and a tool for managing customer requests was implemented. A reduction in customer requests resulted from the use of a help desk tool put into place following the assessment [4]. Defect management was implemented when a software package was purchased to help track and manage defects and issues in one company, and another company improved its requests and defects system which became the driver for all work in that company. Formal projects were linked to existing requests, and corrective maintenance was managed using the requests and defects system [18].

7- FRAMEWORK FOR REALIZING METRICS PROGRAMS

The investigation of reported experiences reveals that the mind-set "metrics only work in large settings" is false. Although well-known software process models work better in large settings, these models have been scaled down for smaller settings and, simultaneously, new models are being developed specifically for smaller organizations. In addition, many years of research in this area has resulted in some options for utilization of software metrics within or outside of a process improvement framework, regardless of the size of the organization. Small software organizations should plan to start small when developing their metrics programs.

A simple framework for realizing a metrics program is based on the Goal/Question/Metric (GQM) paradigm [39]. The GQM approach basically defines the goals an organization wishes to achieve, breaks each goal down into one or more questions that answer whether or not the goal has been reached, and identifies one or more metrics per question that will provide the data needed to answer the question. Note that each question may require more than one metric, and at the same time, a single metric may be used to answer multiple questions [30].

The presented framework includes four steps:

Deciding How to Measure: The first decision facing the small software organization is which approach to take for creating a metrics program. In the following section, three approaches are discussed: the alternatives of scaling down one of the large well-known process improvement models, adopting a smaller process improvement model, or creating a custom program. In this work, the custom program is the primary focus although the topics presented here should prove relevant regardless of which approach is taken.

Overcoming the Obstacles: Many obstacles appear to stand in the way of implementing a new metrics program. Some of these obstacles may be imagined while others are very real. Imaginary or perceived obstacles may include thoughts like "We don't have time to take measurements!" or "We have testing procedures in place already!" or "There's no extra money in the budget for this!" Although all of these may appear valid on the surface, they are really just excuses that distract from the real challenges of collecting useful measurements which may be used for the improvement of products and, ultimately, the bottom line.

Deciding Where to Start: Whenever a new opportunity presents itself, it is usually difficult to determine the best way to accomplish it. Only the organization itself can determine what metrics are best and what results define success. The methods shown in Table 2 allow any organization to ease into the measurement process on their own terms and to adjust their goals and metrics as needed.

Choosing Relevant Metrics: The decision to implement a software metrics program in a small organization is a big one and choosing which metrics to use can seem overwhelming. The next section presents an overview of the types of metrics available, discusses a few of the more popular metrics, and gives some suggestions for the use of some basic metrics as a starting point.

8- IMPLEMENTATION APPROACH

The implementation of a metrics program involves several framework steps. A small organization must consider and benefit from certain options and opportunities that are represented by reported practical experiences of organizations that already instituted metrics programs.

8-1 DECIDING HOW TO MEASURE

Software metrics are recognized as a required tool in process improvement. Several formal models (CMM, CMMI, etc.) have proven successful in helping large organizations that are able to devote much time and many resources to a process improvement program; however, these models have proven difficult to implement in small organizations where budgets and staff are usually at a premium [31]. Yet there is still hope for resource-limited organizations. Software improvement need not be part of a huge, cumbersome, expensive process that reworks the structure of the entire organization. Many improvements can be made by adapting or scaling down one of the large process models, adopting a process model designed for smaller organizations, or utilizing one or more generally accepted paradigms to create a custom improvement program.

Some small organizations have had success in adapting a well-known large process improvement framework on a smaller scale in part by combining roles, procedures, and documentation requirements to reduce the workload required to implement; however, it can be difficult to achieve improvements using this approach, and assistance from an outside party (such as a consultant) may be required to reach a high level of compliance [32].

There are several informal approaches to process improvement (Table 2). In addition, there are process models that are targeted at the small organization, such as LOGOS [33], CMMI in Small Settings, and SE Life-Cycle Profiles for Very Small Enterprises [34]. These relatively recent developments in process modeling indicate that industry and academia are recognizing both the relevance of small organizations and the importance of establishing methodologies for assisting these organizations to reach their improvement goals; however, there still does not appear to be widespread recognition and adoption of these newer models. A small organization may benefit by using one of these approaches, especially if they choose to work with an outside entity to assist with the implementation.

The third improvement option available to the small software organization is to define their own specific goals, determine what success is in terms of how and when these goals are reached, and choose specific measurements and data collection techniques to assess progress toward the goals. Because small software organizations' strengths lie in flexibility, innovation, and speed, this choice may appeal to many who wish to use a "just enough" approach [35]. Choosing to customize its own software process improvement strategy does demand some degree of knowledge, self-assessment ability, and willingness to experiment, and many small software organizations possess these qualities in abundance.

8-2 OVERCOMING THE OBSTACLES

Whichever path is taken towards process improvement, the task of collecting and effectively utilizing metrics data is the primary concern; however, problems that may need to be overcome before implementing a software metrics program can include staff that does not accept a need for change and is unwilling to take time away from their core responsibilities to collect data [35], management who do not understand the need for the additional overhead of an improvement program [30], and the challenge to develop a reliable process that is repeatable and useful going forward [36]. Small steps toward an improved process can go far toward reducing errors and enhancing productivity, so a limited scope is suggested for an initial program. As soon as the benefits of the program become clear, the obstacles should begin to diminish.

How can a software metrics program be justified? If the improvement program has begun, then the organization is already well aware of the requirements for measurement-taking that are embedded in those programs. If the organization is not involved in an organized process improvement program, but simply desires an increase in quality that can be tracked quantitatively, then a literature review of reported experiences should provide evidence of the potential gains inherent in using software metrics. Much research, many publications, and scores of success stories exist across a range of businesses, both large and small, related to software metrics [5].

How can management convince the staff to take time away from daily tasks, which in a small organization can be quite relevant to the group's survival, to take the needed measurements? One way to achieve this is by automated capture of the needed data. These days, much information is stored electronically including program code, project plans, and even staff timesheets. There is the possibility of gathering quite a bit of data from these existing electronic stores with little or no interruption to daily activities. In some cases, this may require some changes to document formats to make the data more consistent, easy to capture, etc. Although automatically captured data can be useful, it may not by itself provide enough detail. Some combination of automatic capture and human reporting of data is likely the ideal [37]. Whatever data people are asked to collect should be both easy to understand and easily collected [38] and should be kept to a minimum.

Convincing management of the benefits of a software metrics program may be easier than convincing engineering staff, especially if the management is well-grounded in quality management principles. Management should be reminded of the high long-term costs associated with poor quality products. All managers should be aware that quality control is essential to an effective manufacturing process, and software production is no exception. The benefits of a software metrics program include creating a repository of information that can be used to more accurately estimate future products, plan project timelines more effectively, and increase customer satisfaction while also having the potential to help reduce defects and maintenance costs.

8-3 DECIDING WHERE TO START

Perhaps the most difficult part of any metrics program is to determine what goals are to be achieved, what measurements need to be captured to assess progress, and how to use the captured data for the improvement of future as well as current projects. A good way to approach this is by utilizing the GQM paradigm. With the GQM approach, the organization defines its goals, breaks each goal down into one or more questions that will answer whether or not the goal has been reached, and identifies one or more metrics per question that will provide the data needed to answer the question. While, at first glance, this approach appears to generate a large number of metrics to be captured, this may not be so. Each goal may create more than one question, and each question may require more than one metric to provide the answer, but it is noted that a single metric may be used in answering multiple questions [39].

Any organization could probably come up with a long list of goals, a huge number of questions, and a very long list of potential metrics. Although every one of these may be valid and valuable, care should be taken to start any improvement program slowly. Trying to do too much too fast can be a recipe for disaster. Take time to work with people of various positions and responsibilities within the organization to get a consensus on an initial small set of goals that are believed to be relatively quickly achievable as well as important to the success of the business. A “quick win” could do wonders in terms of getting those people who had doubts on board with any new process, and beginning with a focus on the most important or urgent needs of the business will help ensure that an initial success will have a real, positive impact. After a small process is in place and real life experience in the implementation and use of software metrics has been gained, the GQM can be revisited to identify additional data to be captured and analyzed [40].

8-4 CHOOSING RELEVANT METRICS

Going through the GQM process should give the organization insight into what it wants to measure, and a review of the results should produce a short list of initial metrics that can answer the questions that will indicate whether those goals are being achieved; however, the metrics defined by a pass through the

GQM are likely somewhat high-level and must be clarified in order to define measurements that are both meaningful and easily captured. Many proposed software measurements cover all parts of the software process from design to development to testing with some being quite simple, others consisting of complex calculations, and still others targeted towards certain types of software such as object-oriented applications or web applications. Since there is no single metric that covers all aspects of software, and because many existing metrics are widely used but none is universally accepted, the organization is left to decide for itself which metric or metrics will best suit its needs.

Whatever metrics are chosen by the organization should be easy to understand, easily collectible, valid across the organization, and meaningful across time [38]. Being easy to understand is obviously a subjective criterion and so must be determined by the people who will actually be transforming collected data into the chosen metric. The exact definition of easily-collectible may also vary from one organization to the next although as mentioned above the more transparent and simple the collection process is the better, both in terms of the completeness of data captured and the reliability of this data [40]. A couple of quality passes through the GQM process should ensure that at least most of the metrics chosen for the improvement program will be applicable to the entire organization. Any metric that helps answer a question that defines a major goal for the organization should be considered as being a valid metric across the organization. The requirement that metrics be meaningful across time reinforces the idea that the metrics program (or the entire improvement process) is not a short-term endeavor but rather that the data collected today is expected to be useful for decisions to be made tomorrow, next year, and going forward [36].

Depending on how the goals and questions are chosen, the organization may require metrics across multiple categories. Some of the basic categories of metrics would include program complexity, development and testing effort, schedule, and quality. Once these categories are listed out, they all would appear to be very important, and there may be a temptation to integrate metrics for all of these categories into a new improvement program. This is certainly a noble intention, and if the resources are available to capture and utilize this data to effect actual improvement then there is no harm in proceeding; however, the focus is on the needs of a small organization with limited resources, so the scope must be limited to gathering the metrics identified in the GQM assessment above. It is important to note that while there are many things that could be measured and possibly improved, every organization has strengths as well as weaknesses, and attention must first be given to making progress in the areas where there is the most room for improvement.

There are some relatively well-known, and more or less widely-accepted, metrics that attempt to provide information for multiple categories (Table 3). One of these is function points (FP) which uses values derived from program attributes along with other values derived from answering questions which are then transformed using complexity weights to calculate the FP value for the

system. The FP calculation requires the use of judgment in determining both the values and the weights used in the calculations which may not yield the same results when applied to a system by different individuals. The cyclomatic complexity metric is another well-known calculation which computes the number of linearly-independent execution paths through the program code. Although the cyclomatic complexity calculations can appear quite daunting to those who are not mathematically inclined, and despite multiple criticisms over the years, the popularity of this metric persists. Many other more targeted metrics exist including various object-oriented metrics, component-level metrics, and user interface metrics.

The most basic metrics that could likely be applied to the categories (Table 3) would be LOC for program complexity, hours worked for development and testing effort, calendar dates for schedule, and defect counts for quality. Although this may sound simple, in a recent study, researchers tested several software packages that computed multiple popular software metrics against several open source systems. They found significant variations in results from one measurement package to the next even when measuring LOC. They attribute these variations to inexact definitions of the metrics as well as variations of the metrics that are difficult to distinguish [41].

9- MAKING METRICS WORK

An organization should not abandon all hope at this point and give up any idea of trying to use metrics to help improve their processes. It should be recognized that just as there are many metrics, and each metric can have many variations, each software organization is also unique. The end result of a metrics program can be valuable when the measurements are kept simple and tailored to each organization [36]. One key to success is consistency which, in regard to a metrics program, means to choose specific measurements, defining a particular calculation for each metric which never changes. Applying the same calculations using the same methodology over time must produce consistent results. On the other hand, metrics that are not working as anticipated should be dropped as it is better to drop a metric that is not providing reliable data and spend resources where they can be of more benefit [40].

One alternative for calculating software metrics is to perform the calculations within the organization. This does not necessarily mean getting out pencil and paper, but it can include internally-developed programs that calculate the chosen metrics. Whether using a spreadsheet application, home-grown software, or longhand calculations, everything remains completely transparent and controllable. Another option for calculating software metrics is the use of off-the-shelf programs. In spite of the findings mentioned above that showed a lack of consistency between various metrics-calculating programs, as long as one program or package is chosen and used for all calculations, the results should remain consistent over time. By ensuring that apples are always compared to

apples, the focus can be on improving the processes rather than continually questioning the data.

At this point, much data has been collected and computed. Of course, opinions abound as to what ideal values are for any given metric, and the values calculated for any given metric may be different than those another organization calculated. As noted previously, each organization is unique, so even if two organizations used the same exact formulas for their calculations, a "good" number for one company may not be excellent for the other, and vice versa. It is worthwhile to read the published research regarding the chosen metric (preferably prior to choosing it) to gain insight into how it is applied in other organizations and to learn from the experiences of those other organizations. It would also be a good idea to run the chosen metric calculations against existing code that is widely considered "good" and against code that is widely considered "bad" or in need of improvement which could help to establish a baseline for the organization to use going forward. It is also imperative that blame not be assigned to individuals as part of the metrics program [5]. This is not only bad for morale; it can undermine the data collection effort and threaten the entire improvement program.

The implementation of any improvement program can seem like a lonely task that can be difficult at times. There are many resources [41] and examples of commercial software [42,43] available for software metrics programs and software process improvement in general. These are listed just to give a small taste of what is available; a quick literature search will yield many more resources. An organization will need to research all of the approaches to software metrics to determine which approach would be best fit for its needs.

To facilitate the success of the metrics program, other developers should be told why the measurements are necessary and how the data will be used. They should know that the data will never be used against them but that they will never be rewarded as a result of the measurement information either. The data should only be made available to those whom it specifically measures. Only the individual developer should be aware of measurements specific to him, and only the team should be aware of measurements specific to the team. All of the data, however, can be broadcast across the organization in general ways to present organizational profiles. Personnel should keep an open mind about the trends that become apparent from the analysis of gathered data.

10- CONCLUSION

The practice of software measurement is lagging behind and yet mature enough to be sought widely across the software industry. Although the importance of metrics has been evident since the early 1970's, the adoption of the practice is moving slowly in software development. Because small software organizations make up a large percentage of software organizations through-

out the world, the adoption of programs of software process improvement and metrics are a necessity in order to begin to resolve the software crisis.

The implementation of these programs in a small organization is not a simple task. They require much effort before and after they are put into place. Some of the challenges to the implementation of these programs include cost, varying perspectives of managers and developers, fear of individual consequences, dishonesty in reporting metrics, little experience in software process improvement, and lack of organizational commitment.

Through the assessment process of an informal approach to software process improvement, strengths and weaknesses of an organization are identified, and these strengths and weaknesses are used in the creation of a measurement program that focuses on business goals using the Goal-Question-Metric methodology. The small organization should fulfill three requirements when choosing measurements for the first iteration of its metrics program: 1) the measurements should be low cost; 2) the measurements should be easy to gather; and 3) the measurements should be easy to analyze.

A Four-Step framework for guiding small organizations through the initial steps of establishing a metrics program was presented in this work. The approach outlines some of the options and opportunities that are available for organizations seeking to improve their development processes. The organization must start small with metrics that provide the highest value and the least possible cost to its operation. Chosen metrics should be easy to understand, easily collectible, valid across the organization, and meaningful across time. A successful metrics program will require the long-term commitment of the technical and management staff to ensure success by seeking to control the process.

The new metrics program should seek solutions for the most important issues and strive for improvement in specific areas of weakness. The exact metrics used will vary by organization, but the end goal will always be a more effective process for producing a higher quality product. The focus should be on creating a program that provides real improvements to the organization as many resources and tools are readily available at little or no cost.

Effective communication between managers and developers is an important factor in the success of software process improvement and metrics adoption. With the trust that is built as a result of effective communication, collected data will be accurate, precisely-documented definitions of metrics will be followed, and consistent data will be collected from project to project.

There are many decisions to be made both before and during the process, and there is no guarantee that any of the choices will be easy or that everyone involved will agree with every decision; however, sufficient evidence has been presented showing the benefits of a metrics program and how a well-planned and well-implemented metrics program promises improvements over the

short-term and the long-term. Making metrics work through an improvement process is essential to realizing these promises.

Future work will focus on further investigation of the GQM paradigm potential to facilitate the proposed framework and its applicability to industrial applications. Furthermore, a case study will be conducted in order to understand the effectiveness of the proposed framework in a practical setting.

REFERENCES

[1] F.J. Pino, F. Garcia, and M. Piattini, "Key Processes to Start Software Process Improvement in Small Companies," Proceedings of the 2009 ACM Symposium on Applied Computing, 2009.

[2] H. Oktaba and M. Piattini, Software Process Improvement for Small and Medium Enterprises: Techniques and Case Studies. Hershey IGI Publishing, 2008.

[3] F. McCaffery, P.S. Taylor, and G. Coleman, "Adept: A Unified Assessment Method for Small Software Companies," IEEE Software, pp. 24-31, January/February 2007.

[4] C. Wangenheim, A. Anacleto, and C. Salviano, "Helping Small Companies Assess Software Processes," IEEE Software, pp. 91-98, January/February 2006.

[5] R.S. Pressman, Software Engineering: A Practitioner's Approach, New York, NY: The McGraw Hill Companies, Inc., 2010.

[6] M. Umarji and C. Seaman, "Gauging Acceptance of Software Metrics: Comparing Perspectives of Managers and Developers," Proceedings of the 2009 Third International Symposium on Empirical Software Engineering and Measurement, Lake Buena Vista, Florida, pp. 236-247, October 2009.

[7] M. Umarji and C. Seaman, "Why Do Programmers Avoid Metrics?" Proceedings of the Second ACM-IEEE International Symposium on Empirical Software Engineering and Measurement, Kaiserslautern, Germany, pp. 129-138, October 2008.

[8] H.M. Haddad and D.E. Meredith, "Instituting Software Metrics in Small Organizations: A Practical Approach," Proceedings of the IEEE International Conference on Information Technology: New Generations, Las Vegas, Nevada, pp. 227-232, April 2011.

[9] H.M. Haddad and N. Ross, "Software Process Improvement and Metrics Adoption in Small Organizations," Journal of Information Systems Technology and Planning, Vol. 3, No. 6, pp. 6-29, December 2010.

[10] N. Ross and H.M. Haddad, "Metrics in Small Software Organizations: Challenges and Possibilities," Proceedings of the 2010 Intellectbase International Consortium Academic Conference, Atlanta, Georgia, pp. 32-41, October 2010.

[11] W.T. Ward, "Calculating the Real Cost of Software Defects," http://findarticles.com/p/articles/mi_m0HPJ/is_n4_v42/ai_11400873, Hewlett-Packard Journal, October 1991.

[12] T. DeMarco, Controlling Software Projects: Management, Measurement & Estimation, Yourdon Press, New York, USA, 1982.

[13] Department of Computer Science, University of Calgary, "SENG Focus Area: Evolutionary Software Engineering," http://www.cpsc.ucalgary.ca/cpsc_research/areas/evolutionary, 2010.

[14] Center for Systems and Software Engineering, University of Southern California, "COCOMO II," http://sunset.usc.edu/csse/research/COCOMOII/cocomo_main.html, 2011.

[15] Quantitative Software Management, Inc., "SLIM Software Cost Estimation, Planning, and Benchmarking Tools," http://www.qsm.com/, 2011.

[16] R. Lytz, "On Board Software for the Boeing 777," http://www.stsc.hill.af.mil/ resources/tech_docs/ gsam3/appenq.pdf, 2010.

[17] Cenqua Pty Ltd., "Clover, Code Coverage," http://www.cenqua.com/clover/doc/coverage/intro.html, 2007.

[18] A.P. Cater-Steel, "Process Improvement in Four Small Software Companies," Proceedings of the 13th Australian Software Engineering Conference, Canberra, ACT , Australia pp. 262-274, August 2001.

[19] A. Anacleto, C. Wangenheim, C. Salviano, and R. Savi, "A Method for Process Assessment in Small Software Companies," 4th International SPICE Conference on Process Assessment and Improvement, Portugal, 2004.

[20] P. Allen, M. Ramachandran, and H. Abushama, "PRISMS: An Approach to Software Process Improvement for Small to Medium Enterprises," Proceedings of the Third International Conference on Quality Software, pp. 211-214, August 2003.

[21] E. McGuire, Software Process Improvement: Concepts and Practices, Hershey, PA: Hershey, Pa., Idea Group Publishing, 1999.

[22] G. Santos, M. Montoni, J. Vasconcellos, S. Figueiredo, R. Cabral, C. Cerdeiral, A.E. Katsurayama, P. Lupo, D. Zanetti, and A.R. Rocha, "Implementing Software Process Improvement Initiatives in Small and Medium-Size Enterprises in Brazil," Sixth International Conference on the Quality of Information and Communications Technology, Lisbon, pp. 187-198, October 2007.

[23] M. Kajko-Mattsson and N. Nikitina, "From Knowing Nothing to Knowing a Little: Experiences Gained from Process Improvement in a Start-up Company," Proceedings of the 2008 International Conference on Computer Science and Software Engineering, Wuhan, China, pp. 617-621, December 2008.

[24] M.C. Mah and L.H. Putnam, "Software by the Numbers: An Aerial View of the Software Metrics Landscape, The IT Metrics and Productivity Institute: Best Practices in Software Development, Management, and Maintenance," http://www.qsm.com/aerialview.html, 1998.

[25] M.L. Hutcheson, Software Testing Fundamentals: Methods and Metrics, Indianapolis, Indiana, John Wiley & Sons, Inc., 2003.

[26] F.J. Koch, "Metrics and the Immature Software Process," http://www.qpmg.com/metrics.htm, 2002.

[27] K.E. Wiegers, Practical Project Initiation: A Handbook with Tools, Redmond, WA: Microsoft Press, 2007.

[28] V. Rubenstein and J. Boria, Small Organizations, "Small Interventions, The IT Metrics and Productivity Institute: Best Practices in Software Development, Management, and Maintenance," http://www.compaid.com/caiinternet/ezine/boria-interventions.pdf, 2007.

[29] L. Westfall, (2005). "12 Steps to Useful Software Metrics," http://www.westfallteam.com /Papers/12_steps_paper.pdf, 2005.

[30] R. Basili, F.E. McGarry, R. Pajerski, and M.V. Zelkowitz, "Lessons Learned From 25 Years of Process Improvement: The Rise and Fall of the NASA Software Engineering Laboratory," Proceedings of the 24th International Conference on Software Engineering, ACM Press, pp. 69–79, 2002.

[31] H. Bae, “Software Process Improvement for Small Organizations,” Proceedings of the 31st Annual International Computer Software and Applications Conference, IEEE Press, 2007.

[32] K.C. Dangle, P. Larsen, M. Shaw, and M.V. Zelkowitz, “Software Process Improvement in Small Organizations: A Case Study,” IEEE Software, pp. 68-75, November 2005.

[33] J.G. Brodman and D.L. Johnson, “A Software Process Improvement Approach Tailored for Small Organizations and Small Projects,” Proceedings of the 19th International Conference on Software Engineering, ACM Press, 1997.

[34] I. Richardson and C.G. von Wangenheim, “Why Are Small Software Organizations Different?” IEEE Software, pp. 18-22, January 2007.

[35] D.P. Kelly and B. Culleton, “Process Improvement for Small Organizations,” Computer, pp. 41-47, October 1999.

[36] K. Kautz, “Making Sense of Measurement for Small Organizations,” IEEE Software, pp. 14-20, March 1999.

[37] L. Hochstein, V.R. Basili, M.V. Zelkowitz, J.K. Hollingsworth, and J. Carver, “Combining Self-reported and Automatic Data to Improve Programming Effort Measurement,” Proceedings of the 10th European Software Engineering Conference, ACM Press, pp. 356-365, 2005.

[38] Y. Chen, R.L. Probert, and K. Robeson, “Effective Test Metrics for Test Strategy Evolution,” Proceedings of the 2004 Conference of the Center for Advanced Studies on Collaborative research, IBM Press, pp. 111-123, 2004.

[39] R. Basili, “Software Modeling and Measurement: The Goal/Question/Metric Paradigm (CS-TR-2956, UMIACS-TR-92-96),” College Park: University of Maryland.

[40] J. Iversen and L. Mathiassen, “Lessons From Implementing a Software Metrics Program,” Proceedings of the 33rd Hawaii International Conference on System Sciences, IEEE Press, 2000.

[41] Software Productivity Center, Inc., “Resources Metrics,” Vancouver, BC, Canada, http://www.spc.ca/resources_metrics.htm, 2010.

[42] McCabe Software, Inc., Columbia, MD, http://www.mccabe.com/iq.htm.

[43] Microguru Corporation, SLOC Metrics, Carlsbad, CA, http://slocmetrics.com, 2010.