Abstract:- Incremental development of software system play very important role in today software development system. It is the technique that will focus new ways of thinking and viewing problems as well as introducing strengths and weaknesses in software development. It also addresses the scenarios and probabilities of reusability of software component that increases the productivity of software component during overall software development life cycle of the system.

Keywords: Software Development, Incremental Development, Productivity, Software Measurement Process

I. INTRODUCTION

Over the past 60 years, there have been significant paradigm shifts in software development, such as structured programming, object oriented programming, Extreme programming and Aspect Oriented Programming and so on. Each evolutionary shift introduced new ways to thinking and viewing problems as well as introducing strengths and weaknesses in software development. In the initial days of software development the idea of software development would not have been as documented or structured as it is done in today’s software environment. In current times, the presence of software development projects without applying any kind of software development methodology presents a high probability of project failure [1]. Although we are applying software development life cycle with increased frequency to most development projects today as compared to the initial years, we are still plagued with these high failure rates. According to the 2009 Standish Group report, it shows more project failing and less successful projects. In other words, as per the Standish Group Report’s "CHAOS Summary 2009", “This year’s results show a marked decrease in project success rates, with 32% of all projects succeeding which are delivered on time, on budget, with required features and functions”[2]. In other words, current software development methodologies do provide a variety of options from which software developers may chose when tasked with matching system. Incremental development techniques provide this facility to the software developer not only in software development but also helps in to determine the productivity of the system. In this technique we combine the property of sequential model and iterative characteristics of prototype life cycle model. Here partial implementation of systems is construct after refine this implementation require functionality achieve.

A productivity measurement is commonly is determined by the ratio of output produced to the ratio of resource consumed. However, the observer has many different choices with respect to the scope and nature of both the outputs and resources considered. For example, outputs might be measured in terms of delivered product or functionality, while resources might be measured in terms of effort or monetary cost. Productivity numbers may be used in many different ways, e.g., for project estimation and process evaluation. An effective productivity measurement helps in the establishment of a baseline against which performance improvement can be measured. It also helps an organization make better decisions about investments in processes, methods, tools, and outsourcing. In addition to the wide range of possible inputs and outputs to be measured, the interpretation of the resulting productivity measures may be affected by other factors such as requirements changes and quality at delivery. Much of the debate about productivity measurement has focused narrowly on a simplistic choice between function points and lines of code as size measures, ignoring other options as well as many other equally important factors. Despite the complexity of the software engineering environment, some people believe that a single productivity measure can be defined that will work in all circumstances and satisfy all measurement users’ needs.

II. PROBLEM STATEMENT

As a considerable portion of software projects miss schedules, exceed their budgets, deliver software with poor quality and even wrong functionality, researchers and industry are seeking methods to improve productivity and software quality. Software reuse has been proposed as a remedy for decades. Reuse is an umbrella concept and the reusable software components, modules in a domain-specific framework, or entire software architectures and their components forming a product family. Object Oriented Development (OOD) provides techniques for the decomposition of a system into independent parts conforming to a component model, thereafter composition of systems from pre-built components. OOD advocates the acquisition and integration of reusable components. Components are more coarse-grained than objects, which may be an advantage in retrieving and assembly, and they conform to a component model, which facilitates composition. Incremental development is chosen to reduce the risks of changing requirements or environments. The basic idea is to allow the developers to take advantage of what was being learned during the development of earlier, deliverable versions of the system and to enhance the system in accordance with the demands of users or the market. While several technologies for software reuse, OOD and incremental development have emerged in
recent years, there are still many open questions. The impact of these technologies on software quality, schedule or cost should be analyzed. The risks associated with single technologies and their combinations should be identified. Case studies in industry play an important role in all these steps, since technologies should be studied in a real context, combined with industrial practices and tuned to fit the context. Incremental development, OOD and product family engineering are especially relevant for developing large, long-lived software systems. In these systems, the scope is gradually covered (and discovered), complexity is handled by decomposition into independent units and thereafter composition, and systems may share software architecture and some core assets to reduce cost and increase productivity. Empirical studies on large systems may answer questions on how certain technologies are applied and adapted for large-scale development.

III. INTERNATIONAL STANDARDS FOR PRODUCTIVITY MEASUREMENT

One might hope to look to the international standards community for guidance on a common industry problem such as productivity measurement. While some help is available from this direction, it is limited. The most relevant resources are as follows:

I. IEEE Standard 1045, Software Productivity Measurement\(^3\) describes the calculation of productivity in terms of effort combined with counts of lines of code or function points. It recommends variations to address software re-use and maintenance scenarios. It provides a project characterization form, but does not discuss how different characteristics might lead to different productivity measures.

II. ISO/IEC Standard 15939, Software Measurement Process [4]. This standard is the basis for the Measurement and Analysis Process Area of the Capability Maturity Model – Integration \(^5\). ISO/IEC Standard 15939 contains two key elements: a process model and an information model. The process model identifies the principal activities required for planning and performing measurement. The ISO/IEC information model defines three levels of measures: indicators, base measures, and derived measures.

A major factor in understanding productivity, especially in product line development, is taking into account the sources of the software that go into a delivery. Usually, at least three categories are used:

I. New – software that is developed specifically for this delivery

III. SEI technical reports discuss how to define efforts \(^6\) and size measures \(^7\), but give little guidance on how they can be combined to compute things such as productivity.

Thus, the SEI reports discuss considerations in defining base measures (using the ISO/IEC Standard 15939 terminology), while IEEE Standard 1045 suggests methods of combining base measures to form derived measures of productivity. Note that none of these standards systematically addresses the factors that should be considered in choosing appropriate base measures and constructing indicators of productivity for specific purposes.

IV. PRODUCTIVITY MEASUREMENT

There are many different approaches to measuring productivity have been adopted by software industries for different purposes. Here we will discuss the approach and makes recommendations during incremental development of software system. The basic equation for productivity is as follows:

\[
\text{Productivity} = \frac{\text{Ratio of Output Produced}}{\text{Ratio of Resource Consumed}} \quad \text{(i)}
\]

\[
\text{Productivity} = \frac{\text{Size}}{\text{Effort}} \quad \text{(ii)}
\]

The above equations are used for measurement and estimation of productivity of outputs verses resources (i) and size verses effort (ii). While productivity measurement takes place during incremental development of software system, the designer must address the following issues in defining a precise productivity indicator:

I. Scope of outputs (product) – which products get counted?

II. Scope of resources – which resources get counted?

III. Requirements (or other input) churn – what if the target changes during development?

IV. Quality at delivery – how are differences in quality accounted for?

II. Modified – software that is based on existing software, but that has been modified for this delivery

III. Reused – pre-existing software that is incorporated into the delivery without change

Measures of product size and resources must be carefully selected in deciding upon the construction of a
productivity indicator. It is not simply a choice between Function Points, Lines of Code, or another size measure. Many other factors also must be considered. These factors are:

I. Requirement Changes: in it percent of project efforts are \(10 - 40^{[8,9,10]}\]

II. Diseconomy of Scale: in it percent of project efforts are \(10 - 20^{[8]}\]

III. Post Delivery Repair: in it percent of project efforts are \(20 - 40^{[9,11]}\]

IV. Software Reuse: in it percent of project efforts are \(20 - 60^{[8,9]}\). 

V. GOAL OF INCREMENTAL DEVELOPMENT OF SOFTWARE SYSTEM

The basic idea behind the incremental development of software system is develop incrementally, allowing the developer to take advantage of what was being learned during the development of earlier, incremental, deliverable versions of the system. Learning comes from both the development and use of the system, where possible. Key steps in the process were to start with a simple implementation of a subset of the software requirements and iteratively enhance the evolving sequence of versions until the full system is implemented. During iterations of design modifications are made along with addition new functional capabilities. Reusable components are easier to maintain (over time) and typically have a higher quality \([12, 13]\) value (more robust and fewer errors). The practice of component reuse supports the motivation for development of incremental development of software project. The main goal of this research is design such kind metrics \([14, 15, 16]\), that is very beneficial to determine quality and productivity \([17, 18, 19]\) during incremental development of software system. These metrics not only measure the quality and productivity, but also determine the percentage of reusable software components during development of software systems. The incremental development \([20, 21, \text{and} 22]\) of software system reduced application development time, reduced application cost, and improved application quality and productivity. Therefore overall goal of research is:

1. Improve staff effectiveness (produce more with less through applied use of technology or incentives)

2. Decrease production workload (produce less new output by reusing previously produced materials to meet an equivalent or increased demand)

REFERENCES


