

EMGT 835 FIELD PROJECT

**Engineering Management approach to Software Production Plan, based
on the principles of Software Product Lines**

By

Rama Shetty, PMP

Master of Science

The University of Kansas

Fall Semester, 2005

An EMGT Field Project report submitted to the Engineering Management Program and the Faculty of the Graduate School of The University of Kansas in partial fulfillment of the requirements for the degree of Master of Science.

DR BOB ZERWEKH

DATE:

COMMITTEE CHAIR

PROF HERB TUTTLE

DATE:

COMMITTEE MEMBER

PROF CHICK KELLER. PMP

DATE:

COMMITTEE MEMBER

(THIS PAGE LEFT BLANK INTENTIONALLY)

ACKNOWLEDGMENTS

This thesis has been one of the most fulfilling endeavors for me in recent times. I am indebted to many who guided and assisted me in completing this journey. This thesis started as research on the concepts and realization of Software Factory. My research efforts led me to new avenues and provided intellectual insight to recent developments in software production such as the principles and practices of Software Product Lines.

Foremost of all, I would like to thank Dr Bob Zerwekh, my EMGT program advisor and the chairperson of the advisory committee for his early guidance in organizing the thesis; Prof Herb Tuttle and Prof Chick Keller for their continuous valuable feedback all along the research work.

I would like to sincerely acknowledge the University of Kansas, its library and its staff who worked behind the scenes to help me avail the required resources for my research. It provided me access to research materials, books and technical papers from the comfort of my home. It also facilitated requesting materials which were not available electronically.

Special thanks go to my employer Sprint Corporation for encouraging and financially supporting my quest for knowledge in management studies; ITS

director Gregory Bradley and my manager Anitha Narayan for guiding me define the scope of this thesis; and my colleagues who regularly provide valuable suggestions and learning opportunities.

My wife, Kanaka has been a wonderful support throughout the EMGT course, especially during the strenuous hours I spent burning the midnight oil.

Last but not the least; I would like to thank the KU writing club and my friend Abhijeet Chitale for their valuable time and effort in proof reading the manuscript and providing critical feedback.

(THIS SPACE LEFT BLANK INTENTIONALLY)

TABLE OF CONTENTS

ACKNOWLEDGMENTS	3
EXECUTIVE SUMMARY	6
INTRODUCTION	8
LITERATURE REVIEW	12
Chapter 1	14
SOFTWARE ENGINEERING AND MANAGEMENT	14
SOFTWARE PRODUCT LINES	17
Chapter 2	23
SOFTWARE PRODUCTION PLAN	23
Configuration Management	25
SCM Activities	27
Data Collection, Metrics and Tracking	30
The Value Chain Model	31
Inventory Management	32
Bottleneck	35
Process Definition	37
Technical Planning	41
Tool Support	45
Chapter 3	47
Findings and Discussions	47
Chapter 4	50
Suggestion For Additional Work	50
Summary	52
GLOSSARY	55
REFERENCES	58
BIBILOGRAPHY	61
APPENDIX A – SOFTWARE ARTIFACTS	64
APPENDIX B – SOFTWARE CONFIGURATION PLAN	69

EXECUTIVE SUMMARY

The concept of the software factory and discussions over its realization dates back more than three decades; however it is still only a vision for medium and large size Information Technology [IT] organizations on an earnest journey for software excellence. The principles and practices of Software Product Lines [SPL] can be a catalyst in realizing such a vision. SPL ensembles three elements namely, core asset development, product development, and management to run an efficient and consistent software factory. The core asset development is based on “Domain Engineering” and is fundamental to leveraging commonalities between software products & services developed. The product development encircles the core assets and provides the variability required in the product line. Finally, management acts as the strategic objective provider. Employing these principles of SPL, a software production plan can be devised with five essential practice areas such as Configuration Management; Data Collection, Metrics and Tracking; Process Definition; Technical Planning; and Tool Support.

Configuration Management is a disciplined approach of evaluating, coordinating, approving or disapproving and finally implementing changes in the software artifacts. Data Collection, Metrics and Tracking enable the software factory to assure quality and seek continuous improvements. Process

Definition ensures the software factory is scientific in its approach and delivers repeatable performance. Technical Planning chalks the roadmap of realizing the prophecy of reusability and fulfilling the software factory vision. Finally proper Tool Support ensures productivity, efficiency and consistent performance from all the stakeholders in the software factory.

(THIS SPACE LEFT BLANK INTENTIONALLY)

INTRODUCTION

The aim of this thesis is to outline a strategic software production plan which can be used uniformly across all Information Technology (IT) production facilities and can be enhanced iteratively. The major objective of the approach is to adopt practices that advocate strategic reuse of software; that facilitates measurements; that can be streamlined for production standards; and that can be employed repeatedly to different setups. The author's vision is to model a software factory employing the best practices in the industry to produce a family of software products of defined quality, consistently on time and on budget. The concept of Software Factory is not novel; it has been tried in many companies since late 1960s especially in Japan, where the supply of software professionals was very scarce. The first company to adopt the name "factory" (actually its Japanese equivalent "kojo") to label its software facility was Hitachi in 1969. Hitachi managers set two goals [Cusumanno, 1989]

1. Productivity and reliability improvement through process standards and control.
2. The transformation of software from a production of unstructured service to a product with a guaranteed level of quality.

The companies that have implemented software factories in early 1970s include NEC, Toshiba and Fujitsu. The software factory was easier to adopt and implement during the fourth generation of computing i.e. the early 1970s as there were very few competing vendors and usually one vendor would provide all the computing needs of the customer. The systems used were uniform and hence the standards and interfaces were compatible. With the advent of Open Systems, the computing landscape changed dramatically. No single vendor could keep up with the rapidly changing technology. Moreover, the decreasing cost of processing increased the demands of the customers. Customers preferred cheaper and faster services/products from multiple suppliers to single and expensive providers.

The power of computer processing has improved multi-fold since the introduction of the third generation computers in 1964. The revolution ignited by miniaturization of electronic chips and processors and its burgeoning supply has enabled every scientist, economist, academic, researcher and business to improve their productivity and enhance their capability by order of magnitude which may have not been possible otherwise [Evans, 1989]. The increase in processing power also enabled the improvement in software applications and utilities. The sophisticated software applications developed to run on these hardware platforms provided the real value enhancements. This

radical evolution in computing technology and the mammoth magnitude of software products, applications, and services, called for a structured software development process.

The classic waterfall model originally presented by W.W. Royce (1970) and later reevaluated by Barry Boehm (1976); enforced the fact that software is an engineering discipline and not an art. The IT landscape has drastically (though not radically) changed with the introduction of Concurrent Engineering, Object Oriented Technologies, Rapid Application Development (RAD), and Distributed Computing in parallel with significant changes in the operational landscape such as Business Process Re-engineering (BPR), Total Quality Management (TQM) and Six Sigma Quality Controls. The newer methodologies such as Extreme Programming (XP) and Feature Driven Development (FDD) emphasize, amongst others, the need for incremental and iterative cycles in the software development model. The most important impact in current computing facilities, industry wide, has been the need for “Economies of Scope” or capitalizing on existing software investments by reusing them to create new products and services.

There has also been other socio-economic factors such as the declining allocation of IT budgets in large firms; consolidation of competing IT suppliers; and competition from cheaper source of overseas IT suppliers that

have compelled internal IT functional departments and external IT vendors to revamp their IT strategies. With the ever increasing demands for faster, better and cheaper software it is now imperative that software factories adopt lean and agile methodologies and robust architecture in their software production.

(THIS SPACE LEFT BLANK INTENTIONALLY)

LITERATURE REVIEW

Book: THE SOFTWARE FACTORY

Author: Michael W. Evans Year: 1989

The book empowers the software manager with the knowledge of setting a vision and formulating strategies to build and run a software factory. The author starts by narrating the legacy of computer engineering and its evolution over the years and acquaints the reader with the key factors affecting the industry. The subsequent chapters deal in details, the software environment, the engineering processes, configuration management and data management; which are key elements governing an efficient software factory. The book also depicts the importance of vendor management in the overall software supply chain.

Book: SOFTWARE PRODUCT LINES

Author: Paul Clements, Linda Northrop Year: 2002

The book is about the 29 practice areas spread across the field of domain engineering, technical management and organization management that are required to fully integrate the organization into “one big” software development entity; to produce software product lines based fundamentally on the theory of “Economies of Scope”. The book also includes a few patterns to address specific problem areas in software production. The fundamental principle behind the Software production discussed in this thesis is based on “Software Product Lines”.

Book: Agile Management for Software Engineering

Author: David J. Anderson Year: 2004

This book is an excellent read for software managers on how to provide value to their customers. The foundation of the book is based on applying the theory of constraints (TOC) practiced in manufacturing to the software industry. The author also compares and contrasts modern software methodologies. He provides ways of accounting and measuring key metrics such as ROI and Net Profit. He has challenged some of the recent software industry practices and has quoted real life experiences to drive his points. The books coverage also includes the different roles of an agile manager.

IEEE Standard for Software Configuration Plan, IEEE Standard 828-1998

Specifications for Software Configuration Management plan, from the Software Engineering Standards Committee of the IEEE Computer Society.

This standard establishes the minimum required contents of a Software Configuration Management Plan (SCMP) along with the specific activities to be addressed and their requirements for any portion of a software product life cycle.

Chapter 1

SOFTWARE ENGINEERING AND MANAGEMENT

The software environment has three distinct components; technical, administrative and management. The technical methods and tools used define, design, develop, integrate, test and implement the software. The administrative methods follow standards and procedures; and provision data. The management is focused on reducing cost, minimizing risk, increasing stability and enhancing productivity [Evans, 1989].

The software engineering environment includes the infrastructure and automated facilities backed by policies and practices to deliver software products. It integrates many activities, resources, controls and technologies. The software factory must be concerned by the total project requirement (see the value chain model below), not just with the segment of software development. It is the means by which software engineering task of integration and the task of project support is achieved. In this environment, all end products or engineering data are developed, updated, maintained and controlled. The environment allows the software staff to specify, design,

implement, test, deliver and control the software data. The software factory produces acceptable data products, monitors performance and productivity, assesses effectiveness and project efficiency, and minimizes risks in production. Within the factory, separate functions work together to produce or support a software product. Technical development, Project Management, Configuration Management (CM), Data Control, Integration and Testing are linked to deliver the goods. Thus a generic software engineering environment enforces both engineering and management discipline.

An engineering manager thus has to strategically plan the mechanics and economics of a software factory. She or he has to synergize these plans with the corporate goals, marketing forecast and industry changes. According to the author, some of the challenges faced by engineering managers are:

1. Maintaining 99.99 % system stability
2. Delivering software product with zero defect
3. Estimating and planning for resources
4. Predicting future performance and scalability of systems required based on marketing data.
5. Reducing time to market

6. Being proactive not reactive
7. Coping with reorganization
8. Delivering continuous performance improvement at par with industry/competition
9. Provide business and project implementation metrics

In manufacturing, economies of scale can usually be achieved by simplifying the product line, scheduling longer production runs for fewer models, and using common parts and components in different models [Thompson, Gamble, Strickland, 2004, pg 118]. Similarly, in software engineering, economies of scope can be achieved by adopting practices from Software Product Lines (SPL), pioneered by Software Engineering Institute (SEI) at the Carnegie Mellon University (CMU) [<http://www.sei.cmu.edu/productlines>]

SOFTWARE PRODUCT LINES

A Software Product Line is a set of software-intensive systems sharing a common, managed set of features, that satisfy the specific needs of a particular market segment or mission and that are developed in a disciplined fashion using a common set of core assets in a prescribed way [Clements, 2001].

Software reuse has been achieved in different forms and at different levels, with maturation in the field of software engineering. Reuse of software has been achieved within many organizations at the subroutine and component level, and has drastically increased in recent years with the advent of Open Source programming. Some of the software packages developed by Open Source organization, which are available to use at no cost have become *de facto* standards across the industry. The pioneers of Software Product Lines have promoted the practice of software reuse from “component level” to “product level”.

Software product lines have emerged as a new software development paradigm of significant importance. At its essence, fielding a product line involves core asset development and product development using core assets,

under the aegis of technical as well as organizational management [Clements, 2001]. All three activities are iterative and interact with each other.



Fig 1: Three Elements of Software Product Lines

[<http://www.sei.cmu.edu/productlines>]

Core Asset Development

The development of core assets has set up the foundation of reuse; its goal is to establish a production capability for products. This activity is iterative and is also referred to as “domain engineering”. Domain engineering refers to the activities that involve, among other things, identifying commonalities and differences between product family members, and implementing a set of shared software artifacts (e.g. components and classes) [Deelstra, 2003]. Each

core asset is associated with an attached process that specifies how the core assets need to be used in developing new products. The core assets comprise of different artifacts, such as the architecture, which the products in the product line will share, reusable software components and/or products developed in house or Commercial Off The Shelf (COTS).

Product Development

The product development is the act of creating new products/services based on the requirements, using the core assets and using a predefined production plan. The software products are developed incrementally and iteratively. These products could be added to the core asset inventory or just be delivered as individual products. The product development adheres to the production plan associated with the core assets.

Management

Technical management practices are those management practices that are necessary to engineer the creation and evolution of both the core assets and products. They oversee the activities of core asset development and product development, ensuring proper processes and discipline are followed. Organizational management plays a critical role in ensuring the right organizational structure, allocating resources, providing training, rewarding

employees, negotiating contracts with external suppliers and institutionalizing the approach in a manner suitable for the organization.

Salient features

1. The creation of products has a strong feedback effect on the product line scope, the core assets, the production plan, and the requirements for specific products.
2. The three essential activities demand high discipline and continuous improvement.
3. The champion sets and maintains vision; and ensures objectives and metrics are in place.
4. The champion sustains high morale and deflects potential derailments.

(THIS SPACE LEFT BLANK INTENTIONALLY)

Practice Areas

A practice area is a group of activities than an organization must master for repeat performance to implement SPL. They help in measuring and controlling progress. The practice areas have specific goals and are listed below.

Software Engineering	Technical Management	Organizational Management
1. Architecture Definition	10. Configuration Management	18. Building a Business Case
2. Architecture Evaluation	11. Data Collection, Metrics and Tracking	19. Customer Interface Management
3. Component Development	12. Make/Buy/Mine/Commission Analysis	20. Developing an Acquisition Strategy
4. COTS Utilization	13. Process Definition	21. Funding
5. Mining Existing Assets	14. Scoping	22. Launching and Institutionalizing
6. Requirements Engineering	15. Technical Planning	23. Market Analysis
7. Software System Integration	16. Technical Risk Management	24. Operations
8. Testing	17. Tool Support	25. Organizational Planning
9. Understanding Relevant Domains		26. Organizational Risk Management
		27. Structuring the Organization
		28. Technology Forecasting
		29. Training

Benefits

Information Technology (IT), though a separate functional unit, is an integral part of business from every walk of life. A strategic initiative such as embracing Software Product Lines should benefit all, the executives, the internal and external customers and the major stakeholders - the knowledge workers.

Some of the benefits at individual levels advertised by SPL are

<p>CEO/COO/CIO</p> <ul style="list-style-type: none"> •Large scale productivity •Improved time-to-market •Efficient utilization of work force 	<p>Technical Manager</p> <ul style="list-style-type: none"> •Improve predictability •Roles and responsibility
<p>Product developer</p> <ul style="list-style-type: none"> •Focus on the uniqueness •Mobility 	<p>Architect</p> <ul style="list-style-type: none"> •Greater challenges •Profound impact
<p>Customer</p> <ul style="list-style-type: none"> •Fewer defect •Quicker turnaround 	

Chapter 2

SOFTWARE PRODUCTION PLAN

In SPL parlance, a production plan describes how the products are produced using the core assets. Each of the core assets have attached processes that define how they will be used in product development. The overall production plan puts all these processes together. The main intent of the production plan is to enable the software factory to deliver software products rapidly and consistently. The production plan could vary from being an informal guideline for the product builder or a detailed process model to be consistently followed by all in the software factory. Hence, the production plan depends on the organizational structure, the culture, the technologies and the tools. The shared software artifacts are constructed in such a way that the commonalities can be exploited economically, while preserving the variability [Deelstra, 2003]. The production plan should also describe the sequence of activities, the procedures and the tools to be used to deliver the final product or service.

SPL principle identifies the following practice area as essential for creating a production plan.

1. Configuration Management
2. Data Collection, Metric and Tracking
3. Process Planning
4. Technical Planning
5. Tool Support

(THIS SPACE LEFT BLANK INTENTIONALLY)

CONFIGURATION MANAGEMENT

Software Configuration Management (SCM) ensures the stability of the software project. It applies an engineering approach to the development, operation, and maintenance of the information systems. Its primary function is to ensure that current, controlled and consistent information is used in the planning, development, and maintenance of the system, subsystems, operational procedures, and support information at the engineering and life-cycle levels. The software engineering environment must decide how to provide a form of CM that is easily tailored to many projects. IT must provide sufficient discipline to control the engineering environment without becoming an administrative burden.

SCM is the means through which the integrity and traceability of the software system are recorded, communicated, and controlled during both development and maintenance. SCM also supports reduction of overall software life cycle cost by providing a foundation for product and project measurement [IEEE Standard 828-1998, pg 3]. It is a disciplined approach of evaluating, coordinating, approving or disapproving and finally implementing changes in artifacts (hardware, software or documents) that are used to

construct and maintain software systems. CM enables the management of artifacts from the initial concept through design, implementation, test, baseline, build, release and continued maintenance [Clements, 2001]. Managers who truly understand software development realize that CM is an integral part of any software engineering environment. It entails managing Software and Hardware. The hardware required ranges from Personal Computers (PC) to midrange computers. The hardware selection process is critical to the effectiveness and success of the software engineering environment. The following factors shape the selection process:

1. The size of the binary
2. Nature of the software engineering environment. For example, a distributed environment needs robust networking capabilities.
3. The number of simultaneous access requests. The hardware system must support the maximum anticipated access load.
4. The processing requirement of the application.
5. Security
6. Frequency of access.

Many times, engineering hardware is chosen based on “gut feel” or present technical trends, often times leading to poor performance of application due to

hardware inadequacies. Selecting hardware solely on the basis of processing power or memory capacity is inadequate for scaling the support requirement.

IEEE Standard for Software Configuration Management Plan, IEEE Standard 828-1998 [see Appendix B] and Standard 1042-1987 outlines SCM plan and includes a thorough overview of SCM plan. An SCM plan can be created for any software application using these standards.

SCM Activities

All projects depend upon good configuration management, and the project manager needs to understand what changes will take place during the project life cycle. These changes need to be managed. Configuration management has four prime areas of interest:

1. **Configuration identification:** This area is concerned with being able to identify all items on a project that must be placed under control.

[Sodhi, 2001]

2. **Configuration change control:** A process is established where the project stakeholders can track all change requests on the project and can assess the impact of these changes to the project schedule and resources

[Sodhi, 2001].

3. **Configuration accounting:** This area is concerned with being able to draw the necessary configuration reports on the project and account for all configuration items on the project [Sodhi, 2001]
4. **Configuration auditing:** This area is concerned with being able to assess and audit the project from a configuration perspective. The outcome of the audit will allow the project manager to implement the necessary corrective actions [Sodhi, 2001].

Successful CM requires a well defined and institutionalized set of policies and standards that clearly define the following

1. The artifacts and software deliverables that need to be included in Configuration Management.
2. The nomenclature scheme to be followed for each set of artifacts
3. The procedures and policies, including user rights to maintain the CM repository
4. The life cycle management of artifacts in a concurrent and parallel development environment
5. The authorized tools for ease of use
6. Version control for creating and maintaining multiple baselines

A brief comparison of a few commercially available CM tools is listed at
<http://better-scm.berlios.de/comparison/comparison.html>

Before selecting a CM tool it is important to design the CM process and use the CM process to select the best tool. This addresses the technical aspect of delivering quality and predictable software and the management aspect of managing software release consistently with concurrent development and geographically dispersed teams.

(THIS SPACE LEFT BLANK INTENTIONALLY)

DATA COLLECTION, METRICS AND TRACKING

“You cannot manage what you cannot measure ... and what gets measured gets done.”

----Bill Hewlett, cofounder of Hewlett Packard.

Traditional methods of controlling software production systems have focused on the use of effort-based metrics, mainly the Lines of Code (LOC). The problem with LOC is that it has no correlation to the value perceived by the customers. However, for want of a better metric, LOC remained popular for a long time. Another metric used is the man hours required to complete an activity. The man- hour is mapped to dollar amount to determine the fiscal amount. However, the activities involved in software development and deployment is so dynamic and volatile that the numbers provided vary drastically between projects, groups, regions and companies. This inconsistency prevents man-hours from being accepted as a true software metrics; they cannot be benchmarked or standardized. Another issue related to these metric is it inability to track non-functional requirements such as scalability, performance on load, 24x7 availability of the system.

Software Measurement Exercise should be goal driven; such as, measure improvement in production performance, measure compliance to process and policies or measure improvement in fiscal ratios.

The Value Chain Model

To derive software metrics, a symbolic model can be built where-in all concepts are represented by quantitatively defined variables and all relationships are represented mathematically. A symbolic model simplifies the complex software process and helps analyze key variables and their effect on the deliverables. Benchmarking is one of the tools used for the comparative analysis of price competitiveness of the company. A symbolic model can be designed to facilitate the measurement of key metric such as Inventory Turnover, Return on Investment (ROI), and Return on Assets (ROA). Such metrics help to define and better tune processes, eliminate bottlenecks, incorporate practices that are more cost effective. Dramatic cost advantages can emerge from finding innovative ways to eliminate or bypass cost-producing value chain activities [Thompson, Gamble, Strickland, 2004].

The software engineering endeavor namely, creating functional application for the customer from functional requirements and the software project management namely, managing the processes to deliver software on time, with in budget, with desired scope; can be generically modeled as shown below. This generic model depicts an iterative software development methodology.

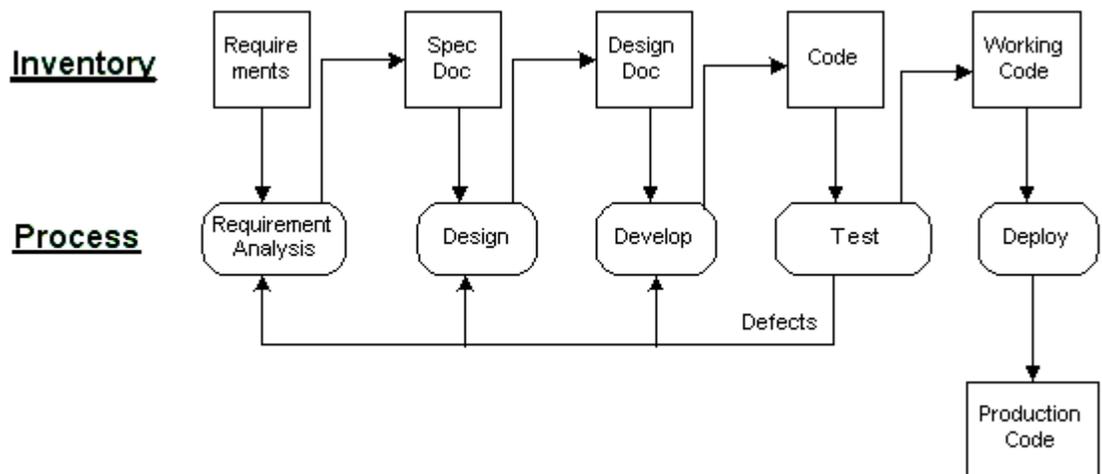


Figure 1: Software Value Chain Model

Inventory Management

Inventory can be expressed in different ways depending on the software engineering lifecycle method being used. In Rational Unified Process (RUP), a unit of inventory is a Use Case [See Appendix A], in Extreme Programming

(XP) it is a story point, in Feature Driven Development (FDD) it is feature from feature list [Anderson, 2003].

Software production is a multi-phased process; inventory can be accumulated at different phases of a Software Development Life Cycle (SDLC), such as analysis, design, code, test etc. Hence, the cost of inventory is averaged over a standard period called “release”, which could be monthly, bimonthly or quarterly. The cost of inventory constitutes two parts, the fixed cost or operating cost accounting for the sunk cost and the variable cost accounting for the variability of the average man hours that is required based on the complexity of the feature being worked on. Function Point Analysis (FPA) is the software industry standard used for measuring this variability. FPA can also be used to determine the average sale price of a function point. The sale price earned by delivering a release is thus the product of the number of function points and the average sale price of a function point [Anderson, 2003].

Some of the key metrics can be measured as follows:

$$\mathbf{ROI} = \frac{\mathbf{Total Sales - Opearting Expenses}}{\mathbf{Total Investment}}$$

$$\mathbf{ROA} = \frac{\mathbf{Total Sales - Opearting Expenses}}{\mathbf{Sunk Cost}}$$

$$\mathbf{Inventory Turnover} = \frac{\mathbf{Total Sunk Cost}}{\mathbf{Total Cost of Inventory}}$$

$$\mathbf{Inventory Turnover in days} = \frac{\mathbf{365}}{\mathbf{Inventory Turnover}}$$

Bottleneck

For efficient management of inventory, it is important to ensure that the machinery on the shop floor is constantly running and efficiently designed to meet the continuous flow of work. In other words a production facility is only as fast as the slowest machine (or activity). The slowest is considered a bottleneck. Once the bottleneck is identified, worked upon and fixed, the next slowest activity can be identified and worked on.

Measuring Software Productivity

The production rate of the software factory can be measured from the business sponsor's point of view or purely technical point of view. The business sponsor is more interested in the number (quantity) of use cases, stories or features being delivered in every release and the durability of the product (quality) generally measured in terms of number of defects. On the other hand the software production rate is also measured by the number of function points delivered in a given time period. This facilitates proper benchmarking and easy quantification in terms of dollars.

Software productivity can be measured separately at different phases of the value chain. This is primarily because the overall productivity is influenced by the skills, the technology, the processes, the requirements, the scope, the schedule, employee morale, the environmental setup and CM to name a few. However it also important to note that Software by it very nature is complex and has inherent difficulties preventing quantum improvement in productivity by improving on any of the incidental factors listed above.

In his essay “No Silver Bullet” Fredrick Brooks (1986) quotes “There is no single development, in either technology or management technique, which by itself promises even one order of magnitude improvement in productivity, in reliability, in simplicity of software projects”. Brooks divides the difficulties influencing the rate of progress as essence and accidents. By essence Brooks refers to the inherent nature of software such as being invisible, being complex and being an abstract representation of the physical model. From the essential difficulties comes communication errors, cost overrun and schedule delays. By accident Brooks refers to the difficulties in the implementation process, mainly due to human and resource limitation such as analytical skills, programming skills and availability of tools and technology to name a few.

PROCESS DEFINITION

“The process of scientific discovery is, in effect, a continual flight from wonder.”

---Albert Einstein

There has always been a debate on the use of processes, especially in the software industry. On one hand, there are arguments that processes need to be defined and communicated for an efficient software factory, while on the other hand, some team members argue that processes lead to bureaucracy and are a major hurdle in running an agile software factory. It is the diversity of roles and responsibilities in software engineering and management that defuses this debate. Software Product Lines maintain that processes that are essential to the long term efficiency and growth of the organization should not be ignored and should be harnessed to provide feedback for continuous improvement. Process definition is an essential aspect of software engineering as it disciplines a group of people to work together cooperatively to solve a common problem. The process is what people do; it is the work they perform [West, 2000, pg 188]. Well defined processes enable the knowledge workers to understand the complete game-plan and proactively collect information that would be required down the line. Processes also helps eliminate errors, gather metrics,

coordinate activities, reduce dependencies on specialization, level resources and helps achieve predictable outcome. This enables management to measure and hence control the capabilities of the factory. The level of details present in any process description should be limited to what is needed to achieve the organizational goal of commonality, while preserving the flexibility needed to run projects effectively and efficiently. Overly detailed corporate processes are a barrier to learning and innovation [Miranda, 2003].

The fundamental rule for the developers of core assets is to define processes that should be followed to integrate the core asset into the product line, since the product developers are not the original core asset developers. The product developers however should provide feedback and share experience with the core asset developers to improve these processes. Processes are also defined to explain how core assets are created, evaluated, maintained and evolved over the lifetime of the product line.

Potential causes of poor process definition are:

1. Process mismatch: The processes may be too complex for the workforce and may not match their skills, resulting in rework or abandonment of the process or too simplistic or generic, missing the

details required for quality work. The processes also should match the organizational structure and its culture [Clements, 2001]

2. Process doesn't not address product line needs: The processes defined by the core asset developers may not satisfy a particular product line need, hence; core asset developers should be involved in product line development and ensure continuous improvement of processes [Clements 2001].
3. Inadequate process support: The process implementations should be backed by proper training, motivation, examples and proper communication. Lack of such support will result in rejection of such processes [Clements, 2001].
4. Lack of buy-in: The organization may not buy into process definition, resulting in failure to adopt it [Clements, 2001].
5. Outsourcing Process Improvement: By definition of "institutionalization" the organizational unit must own the change & must own the processes [West, 2000, pg 218].

Process definition should address key activities such as Estimation, Budgeting, Quality Assurance, Risk Management, Logistics, Configuration Management, Change Management and Administration amongst others.

Processes should be improved naturally by methods such as Weeding and Nurturing. Weeding an organization's processes is a matter of finding and removing the policies, standards, procedures and processes which don't help people in the organization do their work [West 2000]. Nurturing involves standardizing best practices across the organization, identifying procedure that needs grooming and management support and developing templates that can be reused.

(THIS SPACE LEFT BLANK INTENTIONALLY)

TECHNICAL PLANNING

Planning is one of the fundamental functions of management, and a stepping stone for other functions such as execution and control. The technical planning is specific to projects, concerning mainly with developing specific product; a core asset or line product. The technical plan is a deliverable that provides guidance to the stakeholders of the software factory, to comprehend the state, sequence and schedule of each activity. There are different plans to address different purposes, such as software development plan, quality assurance plan, configuration management plan, risk management plan to name a few. The process of planning should include

1. establishing the plan and its contents
2. establishing commitments to the plan
3. establishing estimates of the resources required to carry out the plan
4. reviewing the plan for feasibility by those who will be bound by it

The planning process needs to be iterative and ongoing; the plans could change. Plans should be updated and revised as appropriate throughout their lives.

The technical plan should be strategic and should include the following:

Goals: A goal is a statement of desired state that will be achieved by the successful execution of the plan. It should be related to the Vision and Mission statement of the organization, business unit or functional unit. It should project “where we are going” and point the stakeholders in a particular direction.

Objectives: Objectives are the unit’s performance targets; the results and outcomes to be achieved. They should be **Specific, Measurable, Achievable, Realistic and Time bound (SMART)**.

Strategies: A unit’s strategy consists of the competitive moves and business approaches that managers employ to grow, conduct operations and meet objectives.

Activities: Activities are the steps and tactical actions that are executed in line with the strategy to meet the objectives.

Resources: Resources are the operational overheads consumed by the activities.

Organization Structure: The organization structure charts the roles and responsibilities and designates authority.

As part of the Software Product Line principle, the plan should also include:

Core asset development plans: These plans outline the strategic use of the core assets and how the repository is enriched and enhanced.

Production plans: These plans generally accompany the core assets and illustrate how the core assets are to be used in the production of the product line.

In essence, the Technical Plan should answer the following questions [Clements, 01]: What is the set of core assets? How will core assets be created initially? How will core asset be tested or evaluated? How will the core asset base be expanded and maintained, and how will components be certified for incorporation? How might these plans interact with product development plans?

Core asset development and maintenance plans and product development plans might have mutual dependencies illuminated by the questions like the following: What core assets must be available to support particular product development needs? When must they be available? Are any product development projects providing components to the core asset base? When are

they needed? How are they to be incorporated? How will the configuration of core assets and changes to the core assets be controlled? What process must be followed to utilize and adapt the core assets for use in each product? How will the required tailoring be accomplished? How will any development unique to the product be accomplished to supplement the core assets? How will product be tested? [SEI, 2000]

(THIS SPACE LEFT BLANK INTENTIONALLY)

TOOL SUPPORT

A good workman is known by his tools ----- A proverb

Some processes, tasks and techniques can be applied manually, while others must be supported by tools for practical implementation. Without adequate tool support, passing information amongst tasks is very inefficient [Hsueh, Houghton 1994]. Software development organizations use tools to perform their day to day activities; essentially, to transform customer requirements to software products. These tools not only improve the productivity of the workforce but also help in estimating efforts required to deliver a certain artifact. Incorporating these tools within the Software Factory should a strategic endeavor to integrating the various activities as much as possible. The tools should not be procured in isolation and should be evaluated with the interest of the whole software factory in mind. The tools used help automate analysis, design, code, test and package products. Companies use sophisticated automated tools, usually web enabled for ubiquitous access, to initiate, track, control, fund and approve projects.

The table below lists a few examples (and is not a comprehensive list) of different tools that are used during the different phases of SDLC.

SDLC Phase	Tools
Analysis	Rational Rose
Design	Rational Rose, Together J
Develop	Together J, WSAD, Code Guide
Test	Win Runner, Skill Worm
Build/ Deploy	Ant

Table 1: Software Development Life Cycle Tools

Efforts are on in building tools to generate products using existing software components dynamically on demand. Such a tool would drastically improve the time to market and the maintainability of the product. Defective features can be eliminated and new product can be regenerated to serve the customer, providing high stability.

Chapter 3

FINDINGS AND DISCUSSIONS

Though the conception and early practice of Software Factory dates back to the 1960s, it is still more of a vision than reality. Software production has been practiced more like an art rather than science. The software industry is about four decades old now, but the practice of software production varies tremendously from coast to coast, company to company. Even departments within the same organization do not agree upon a common approach or methodology of software production. Software production is complex and hence requires a thorough and proven plan, one that can be repeated for consistent results. The software industry even tried to replicate the success of outsourcing in the manufacturing industry, but to no avail. The outsourcing model was adopted by the software industry to cut costs, without acknowledging the fact that software factory has little in common to the traditional factories, In traditional factories people act as machinery in performing predetermined, repetitive tasks, where as, in a software factory knowledge workers strive to produce products or services which are

completely different each time. It is now acknowledged that the outsourcing model cannot be employed to shift a chaotic, ill-managed, process-less function to a supplier; only a well defined, process oriented, quality controlled shop floor can be considered a candidate for outsourcing. Hence, there is now a clear recognition and need for a disciplined engineering approach. The lack of discipline has resulted in inconsistent performance, low predictability, high cost and longer time to market. The Software Engineering Institute (SEI) has developed and documented principles of software production called “Software Product Lines”, with 29 practice areas, emphasizing the need for better communication between the producers and the consumers. The concept dwells on the philosophy of reuse at the product level, to reap order of magnitude improvement in cost and quality. Companies like Microsoft have recognized and have begun to adopt the principles of Software Product Line [Greenfield, 2004]. Instead of rewriting software from ground zero, many products or services can be created by using existing software. A few companies that have successfully implemented SPL practices are Celsius Tech, Cummins Inc, Market Maker Gmbh and Nokia [Donahue, 2005].

Most of the fortune 500 companies and the ones with huge investments in IT, rely on the top notch consulting companies such as Boston Consulting Group, Accenture, IBM and others to educate them about the new techniques, IT

practices and to help them develop their IT strategy. The SEI is a federally funded research and development center sponsored by the U.S Department of Defense; hence are restricted from providing direct consultation to any company. They however provide education and training to commercial companies in the field of software engineering and related disciplines, to ensure development and operation of systems with predictable and improved cost, schedule, and quality. SEI has also pioneered the Capability Maturity Model[®] Integration (CMMI) , a process improvement approach that provides organizations with the essential elements of effective processes. It can be used to guide process improvement across a project, a division, or an entire organization [<http://www.sei.cmu.edu/cmmi/general/general.html>].

SPL is not just a stepping stone, but a gamut of practice areas to adopt in order to improve efficiency and predictability; decrease cost and time.

Chapter 4

SUGGESTION FOR ADDITIONAL WORK

The scope of this thesis is limited to outlining a software production plan. Though the author has restricted this discussion only to configuration management, Data Collection, Metrics and Tracking, Process Definition, Technical Plan and Tool Support; there are some other areas that need special attention. The ones of significant importance are Vendor Management and People Management.

Vendor management

Many Software factories are dependent on vendors for providing contract resource: staff or tools. The dependence on vendors varies based on the technologies used, organization structure and business model. The vendor should be employed as a strategic partner and should be made aware of their critical role in the value chain. They should be evaluated and reevaluated periodically based on the value addition to the projects. The vendor resources

should be selected based on the priorities of the project: cost, schedule, quality or technical expertise.

People Management

Software industry is the most dynamic and rapidly changing industry ever known. Hence in this industry it's not the experts but the most adaptable that survive. The knowledge workers are expected to update their skills and technical knowledge and the management is expected to invest in training and preparing their work force for the new challenges.

(THIS SPACE LEFT BLANK INTENTIONALLY)

SUMMARY

Before adopting the principles of Software Production Lines, the organization should develop a vision and emphasize the strategic need for Software Product Lines. Champions are required for the successful implementation of SPL and should be from within the organization and must also have strong support from the higher management. Software Engineering and Management practices identified by Software Production Lines need a robust and sound architecture driven approach. A strong process oriented discipline is also required, to foster effective measurements and deliver consistent results. The organization should have a culture of adopting the best practices, enthusiastically embrace metrics and tracking methodologies and tools. A few companies have successfully implemented SPL; their results and experience are very encouraging. The SPL Conference held annually world-wide has garnered lot of interest from IT suppliers and medium to large organizations with in-house IT production. SPL does not promise to be an immediate panacea for all IT woes, but it lays out a strategic road map for an organization to realize an order of magnitude improvement through economies of scope. SPL would well be an option to lower IT cost and a substitute for outsourcing or off-shoring of IT services.

The key practices for software production plan namely Configuration management; Data collection, metrics and tracking; Process definition; Technical planning and Tool support were highlighted in this thesis. These practice areas laid over a strong foundation of Software Architecture can immensely improve the function of software factory, reducing cost, improving on-time delivery, reducing defects and rework and most of all provide predictable and incremental enhancements to product lines.

The champions of SPL practices should plan for success, by constantly communicating the strategic objectives for the new paradigm shift and should have the buy in from all the stakeholders. The strategic objectives help make the right decisions and prevent cost escalating, schedule hampering and quality compromising leeway. The configuration management should not only act as a passive repository but should also have advanced capabilities such as defect tracking and change management. The data collection and metrics should not be viewed as bureaucratic; instead should be embraced as a tool for improving productivity. The technical plan should not be a dictum from the architects; instead a guide for production line developers empowered to contribute to it. The tools employed should be selected and customized not just for automation but also for proactively determining problem areas.

Last but not the least the software factory should be designed and developed with the most important asset in mind; its people. The software factory can only thrive with the best interest of the people in the forefront.

(THIS SPACE LEFT BLANK INTENTIONALLY)

GLOSSARY

Architecture	The highest level concept of a system in its environment. The architecture of a software system (at a given point in time) is its organization or structure of significant components interacting through interfaces, those components being composed of successively smaller components and interfaces. The organizational structure of a system. Architecture can be recursively decomposed into parts that interact through interfaces, relationships that connect parts, and constraints for assembling parts. Parts that interact through interfaces include classes, components and subsystems.
Artifact	A piece of information that is used or produced by a software development process. An artifact can be a model, a description, or software.
Benchmarking	The process of investigating and identifying "best practices" and using them as a standard to improve one's own processes and activities.
CMM	Capability Maturity Model – defined by the Software Engineering Institute at Carnegie Mellon University
COTS	Commercial Off-The-Shelf (products)
Domain	An area of knowledge or field of practice.
Domain analysis	A process of capturing and representing the characteristics of a particular domain
Domain engineering	An engineering process that develops software assets for one or more domains.
Economies of scope	The condition wherein fewer inputs such as effort and timer are needed to produce a greater variety of outputs. Greater business value is achieved by jointly producing different outputs. Producing each output independently fails to leverage commonalities that affect costs. Economies of scope occur when it is less costly to combine two or more products in one production system than to produce them separately.
FDD	Feature Driven Development

FPA	Functional Point Analysis: A standard method for measuring software development from the customers point of view. It measures software by quantifying its functionality, provided to the user, based primarily on the logical design.
Inventory	Stock of unused, working in progress or finished software artifacts not released to production.
IT	Information technology
Iteration	A distinct sequence of activities with a base-lined plan and valuation criteria resulting in a release
Practice area	A body of work or collection of activities a organization must master to successfully execute.
Process	A series of linked activities that perform a specific objective. A process has a beginning, an end, and clearly identified inputs and outputs.
Production plan	The guide to how products in the will be constructed or service will be rendered.
Project Manager	The role with overall responsibility for the project. The Project Manager needs to ensure tasks are scheduled, allocated and completed in accordance with project schedules, budgets and quality requirements.
Quality	A customer's total experience with product or service. It includes features and the performance dimensions of those features such as reliability, usability, safety, maintainability.
RAD	Rapid Application Development
Release	A subset of the end-product that is the object of evaluation at a major milestone. A release is a stable, executable version of product, together with any artifacts necessary to use this release, such as release notes or installation instructions.
ROI	Return On Investment
RUP	Rational Unified Process
SCM	Software Configuration Management
SDLC	Software Development Life Cycle
SPL	Software product Lines
SLA	Service Level Agreement
Strategy	The way that an organization positions and differentiates itself from its competitors.
TOC	Theory of Constraints

UML	Unified Modeling Language
Value Chain	Popularized by Michael Porter; Chain of interlinked activities that adds value to provide the complete valued result.
XP	Extreme Programming

(THIS SPACE LEFT BLANK INTENTIONALLY)

REFERENCES

[**Anderson 2003**] David J. Anderson, “Agile Management for Software Engineering”, The Coad Series, Prentice Hall. ISBN 0-12-142460-2

[**Brook, 1995**] Frederick P. Brooks, “The Mythical Man-Month”, Addison-Wesley, ISBN 0-201-83595-9

[**Clements, 2001**] Paul Clements and Linda Northrop, “Software Product Lines”, SEI Series in Software Engineering, Boston, MA. Addison Wesley, 2001.

[**Cusumano, 1989**] Michael A. Cusumano, Massachusetts Institute of Technology, “The Software Factory: A Historical Interpretation” IEEE Software March 1989.

[**Deelstra, 2003**] Sybren Deelstra, Marco Sinnema and Jan Bosch, “Experiences in Software Product Families: Problems and Issues During Product Deviation, <http://segroups.cs.rug.nl>

[**Donahue, 2005**] Patrick Donahue, “Software Product Lines Course” Training material, Software Engineering Institute Carnegie Mellon University, Pittsburgh, PA

[Evans, 1989] Michael W. Evans, “The Software Factory”, John Wiley & Sons, 1989.

[Greenfield, 2004] Jack Greenfield, “Software Factories: Assembling Applications with Patterns, Models, Frameworks, and Tools”, Microsoft Corporation, November 2004

[Hsueh, Houghton, 1994] T. Richard Hsueh, Thomas F. Houghton, Joseph F. Maranzano, Gerald P. Pasternack, “Software production: From Art/Craft to Engineering”

[Miranda, 2003] Eduardo Miranda, “Running the Successful Hi-Tech Project Office”, Artech House © 2003 ISBN: 1580533736

[SEI, 2000] Software Engineering Institute, “Product Line Practice”
<http://www.sei.cmu.edu/productlines>

[Sodhi, 2001] IT Project Management Handbook by Jag Sodhi and Prince Sodhi ISBN: 1567260985 Management Concepts © 2001

[Thompson, Gamble, Strickland, 2004] Arthur A. Thompson, John E. Gamble, A. J. Strickland, “Strategy, Winning in the Marketplace” McGraw Hill ISBN 0-07-298990-4

[West, 2000] Michael West, “Real Process Improvements using CMMI”, Auerbach Publications, A CRC press company.

(THIS SPACE LEFT BLANK INTENTIONALLY)

BIBLIOGRAPHY

A feature Oriented Approach to modeling and Reusing Requirements of Software Product Lines, Hong Mei, Wei Zhang, Fang Gu, Institute of Software, School of Electronics Engineering and Computer Science, Peking University, Beijing, China

Agile Software Process and Its Experience, Mikio Aoyama, Department of Information and Electronics Engineering, Niigata Institute of Technology, Japan

Concepts in Configuration Management Systems, Susan Dart, Software Engineering Institute Carnegie-Mellon University

Configuration Management and Open Source Projects, Andre van der Hoek, Institute for Software Research, Department of Information and Computer Science.

Engineering Management in Our Modern Age, Wade H. Shaw, Engineering Management Process, College of Engineering, Florida Institute of Technology

E-Technologies for the Web Business, Thomas Schmidt, Karl Furst, Gerald Wippel, automation Control Institute- TU Vienna

IEEE Standard for Software Quality Assurance Plans, Std 730-1998, Software Engineering Standards Committee of the IEEE Computer Society.

Research Directions in Software Process Improvement, David N. Card, Software Productivity Consortium

Reusable Software Libraries, Wolf Zimmermann. IEEE Proceedings online 20051253

Software Economics, Barry W. Boehm and Kevin J. Sullivan University of Southern California and University of Virginia, boehm@cs.usc.edu; sullivan@Virginia.edu , December, 1999

Software Factory: a historical interpretation, Michael A. Cusumano, Journal: IEEE Software; Volume: 6 Issue:2 Year: 1992 pg 10-73

Software Factory Principles, Architecture and Experiments, Christer Fernstorm, Kjell-Hakan Narfelt, Lennart Ohlsson; Volume: 9 Issue: 2 Year:1992 pg 36-44

Software Product Lines, John D. McGregor, Clemson University and Luminary Software, U.S.A.. Journal: Journal of Object Technology; Volume: 3 Issue 10 Year: 2004

Software Production Strategies - From Multiple Viewpoint, H T Yeh,
AT&T Labs, ICSI 1992, Proceedings of the Second International Conference on
Systems Integration, pg 210-218

Software Production: From Art/Craft to Engineering, T. Richard Hseuh,
Thomas F. Houghton, Joseph F. Maranzano, Gerald P. Pasternack. Journal : AT;
Volume: 73, Issue: 1, pg 59-68

The Quest for Software Components Quality, Miguel Goulao, Fernando
Brito Abreu, Information systems Group, Departamento de Informatica, Portugal

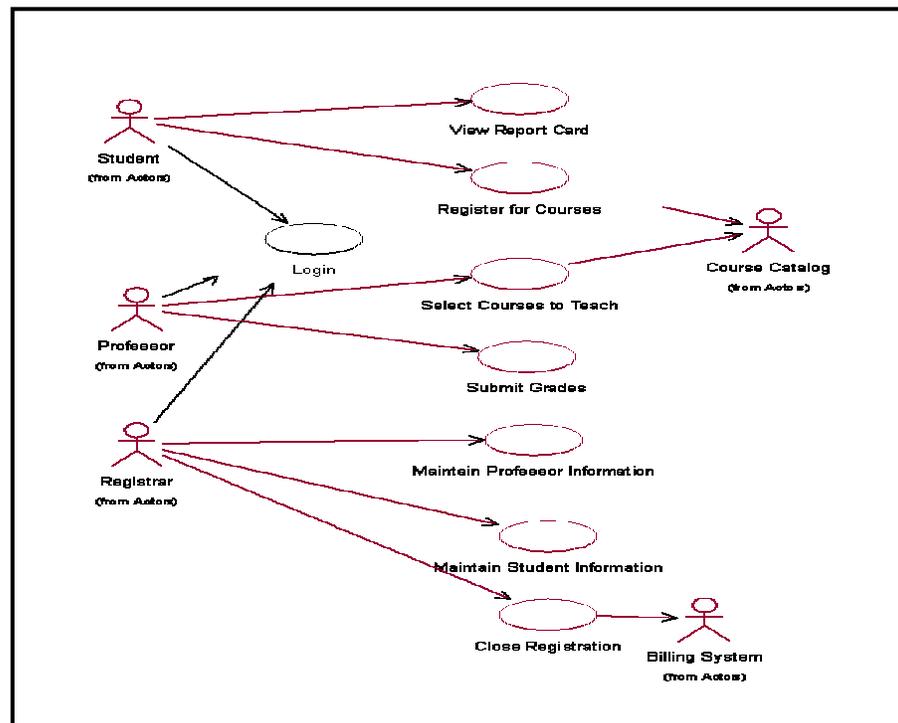
(THIS SPACE LEFT BLANK INTENTIONALLY)

APPENDIX A – SOFTWARE ARTIFACTS

A software artifact is any piece of information that is used or produced by a software development process. An artifact can be a model, a description, a architecture document or software binary.

The following examples depicts different software artifacts

1. Use Case Diagram



Source : Rational Unified Process documentation

2. Use Case

Login Use Case

1. **Brief Description**

This use case describes how a user logs into the Course Registration System.

The actors starting this use case are Student, Professor, and Registrar.

2. **Flow of Events**

The use case begins when the actor types his/her name and password on the login form.

1. **Basic Flow - Login**

1. The system validates the actor's password and logs him/her into the system.

2. The system displays the Main Form and the use case ends.

1. **Alternative Flows**

1. *Invalid Name / Password*

Issue: Need to decide whether password security is necessary for this application.

1. **Special Requirements**

Special requirements will be determined during the next iteration.

2. **Pre-Conditions**

Pre-conditions will be determined during the next iteration.

3. **Post-Conditions**

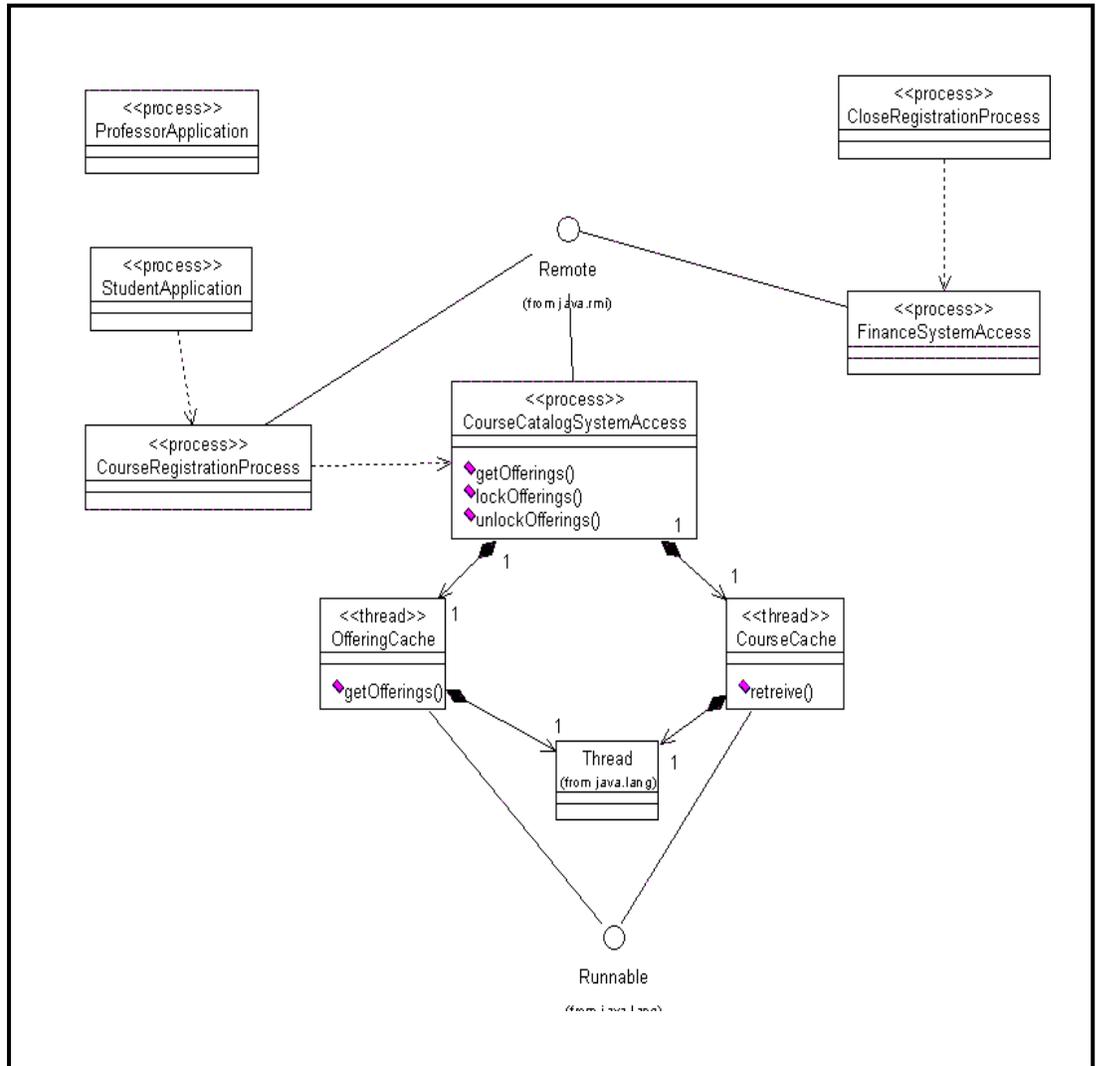
Post-conditions will be determined during the next iteration.

4. **Extension Points**

1. Extension points of the business use case will be identified during the Elaboration Phase.

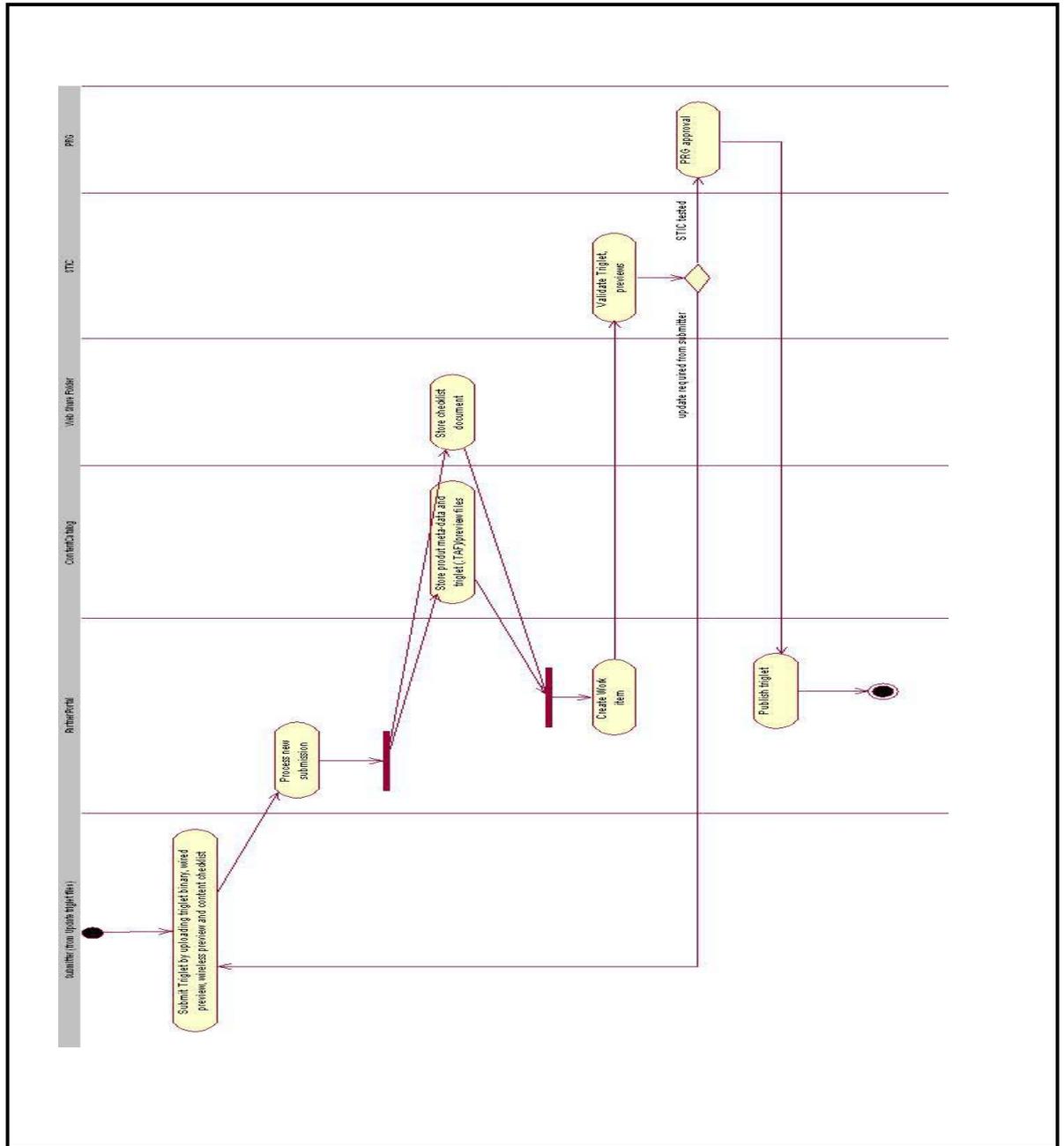
Source : Rational Unified Process documentation

3. Class Diagram

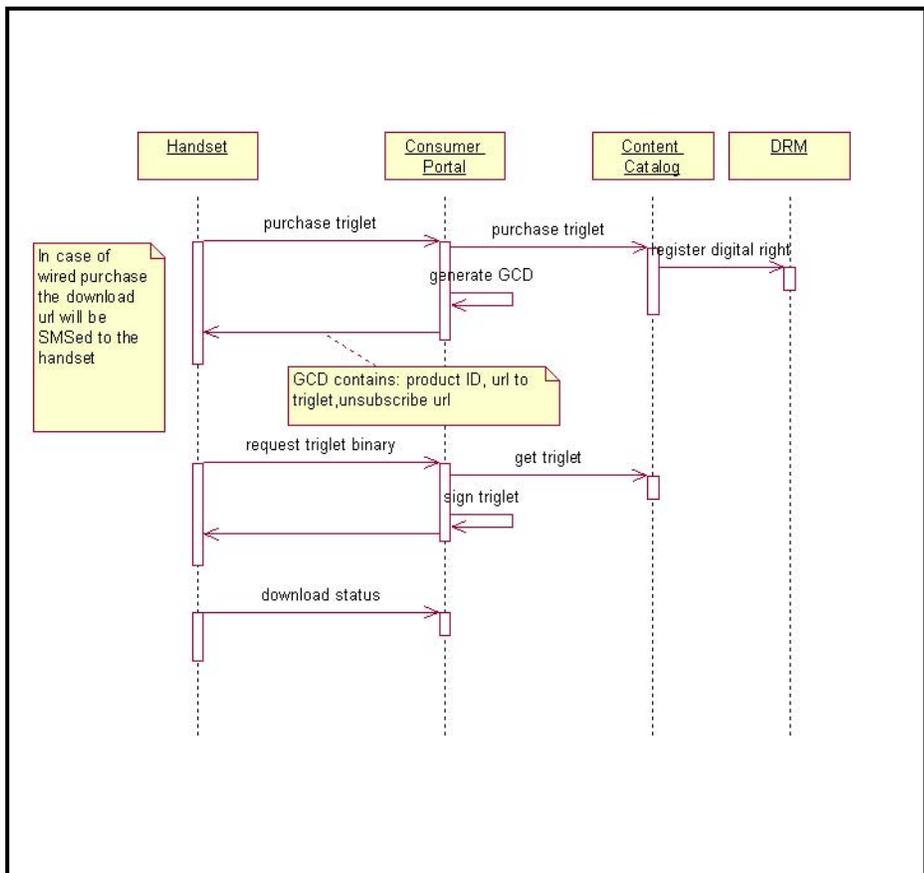


Source : Rational Unified Process documentation

4. Activity Diagram



5. Sequence Diagram



APPENDIX B – SOFTWARE CONFIGURATION PLAN

Source: IEEE Standards Association

<http://standards.ieee.org>

IEEE Std 828-1998 IEEE Standard for Software Configuration Management Plans –Description

Abstract: The minimum required contents of a Software Configuration Management Plan (SCMP) are established, and the specific activities to be addressed and their requirements for any portion of a software product's life cycle are defined.

Keywords: configuration control board, configuration items, software configuration management, software configuration management activities

Content list

- 1. Overview
 - 1.1 Scope
- 2. References
- 3. Definitions and acronyms
 - 3.1 Definitions
 - 3.2 Acronyms
- 4. The Software Configuration Management Plan
 - 4.1 Introduction
 - 4.2 SCM management
 - 4.2.1 Organization
 - 4.2.2 SCM responsibilities
 - 4.2.3 Applicable policies, directives, and procedures

- 4.3 SCM activities
 - 4.3.1 Configuration identification
 - 4.3.2 Configuration control
 - 4.3.3 Configuration status accounting
 - 4.3.4 Configuration audits and reviews
 - 4.3.5 Interface control
 - 4.3.6 Subcontractor/vendor control
- 4.4 SCM schedules
- 4.5 SCM resources
- 4.6 SCM plan maintenance
- 5. Tailoring of the plan
 - 5.1 Upward tailoring
 - 5.2 Downward tailoring
 - 5.3 Format
- 6. Conformance to the standard
 - 6.1 Minimum information
 - 6.2 Presentation format
 - 6.3 Consistency criteria
 - 6.4 Conformance declaration
- Annex A Cross-reference to 1042-1987
- Annex B Guidelines for compliance with IEEE/EIA 12207.1-1997
 - B.1 Overview
 - B.1.1 Scope and purpose
 - B.2 Correlation

- B.2.1 Terminology correlation
- B.2.2 Process correlation
- B.2.3 Life cycle data correlation
- B.3 Document compliance
 - B.3.1 Compliance with information requirements of IEEE/EIA 12207.0-1996
 - B.3.2 Compliance with generic content guidelines of IEEE/EIA 12207.1-1997
 - B.3.3 Compliance with specific content requirements of IEEE/EIA 12207.1-1997
 - B.3.4 Compliance with life cycle data characteristics objectives
- B.4 Conclusion