İSTANBUL TECHNICAL UNIVERSITY ★ INSTITUTE OF SCIENCE AND TECHNOLOGY

CMMI IMPLEMENTATION FRAMEWORK

MASTER'S THESIS by
Didem KÖKTEN, B.Sc.

Department: Computer Engineering

Programme: Computer Engineering

JUNE 2007

İSTANBUL TECHNICAL UNIVERSITY ★ INSTITUTE OF SCIENCE AND TECHNOLOGY

CMMI IMPLEMENTATION FRAMEWORK

MASTER'S THESIS by
Didem KÖKTEN, B.Sc.
(504031547)

Date of submission : 21 May 2007
Date of defence examination: 14 June 2007

SuperVisor (Chairman): Prof. Dr. Eşref ADALI

Members of the Examining Committee Prof.Dr. Coşkun SÖNMEZ

Assoc. Prof.Dr. Mustafa KAMAŞAK

JUNE 2007

İSTANBUL TEKNİK ÜNİVERSİTESİ ★ FEN BİLİMLERİ ENSTİTÜSÜ

CMMI UYGULAMA ALTYAPI SİSTEMİ

YÜKSEK LİSANS TEZİ
Müh. Hatice Didem KÖKTEN
(504031547)

Tezin Enstitüye Verildiği Tarih: 21 Mayıs 2007
Tezin Savunulduğu Tarih: 14 Haziran 2007

Tez Danışmanı: Prof. Dr. Eşref ADALI

Diğer Jüri Üyeleri: Prof.Dr. Coşkun SÖNMEZ

Assoc. Prof.Dr. Mustafa KAMAŞAK

HAZİRAN 2007

ACKNOWLEDGEMENTS

I would like to express our sincere gratitude to my project supervisor Prof. Dr. Eşref Adalı for his guidance, motivation, patience and continuous support.

INDEX

ACKNOWLEDGEMENTS iii
INDEX iv
ACRONYMS vi
TABLE LIST vii
FIGURE LIST viii
ÖZET xi
SUMMARY xiii

1. INTRODUCTION 1
1.1 Thesis Introduction 1
1.2 General Concepts 3
1.3 Literature and Theoretical Focus 4
1.3.1 SPI Method 5
1.3.2 Software Process Reuse Repository 7
1.3.3 Software Process Evolution and Change Management 7
1.3.4 Knowledge-based Software Process Model 7
1.4 Strength of the Process Improvement in Practice 7
1.5 Target of the Thesis 9

2. MAIN CONCEPTS 11
2.1 SPI 11
2.1.1 What is SPI? 11
2.1.2 Importance of SPI 12
2.1.3 How is SPI Determined? 13
2.1.4 Key Methods for SPI 14
2.1.5 General-Purpose Process Improvement Cycles 15
2.1.6 General-Purpose Process Improvement Criteria 15
2.1.7 Software Process Modeling Notations 17
2.1.8 Software Engineering Standards 18
2.1.9 Software Engineering Life Cycles 19
2.1.10 Software Engineering Methodologies 19
2.1.11 Software Engineering Notations 19
2.1.12 Software Engineering Processes 20
2.1.13 Software Engineering Tools 20
2.2 CMMI 20
2.2.1 CMMI History & Background 21
2.2.2 CMMI Content 21
2.2.3 CMMI Models 22
2.2.4 Process Areas 24
2.2.5 Levels 25
2.2.6 Process Areas and their Relationships 29
2.3 Practical Implementation 36

3. PRESENTATION OF THE THESIS **39**
3.1 Subject of the Thesis 39
3.2 Process Areas and CMMI Compliance 40
3.3 Traceability Property 41
3.3.1 Requirements Management 41
3.3.2 Project Management 42
3.3.3 Analysis & Design and Implementation 42
3.3.4 Testing 43
3.3.5 Change Management 43

4. THE IMPLEMENTATION PROCESS **44**
4.1 Process Management Framework 44
4.1.1 Implementation Solution 44
4.1.2 Process Structure 44
4.1.3 Discipline Details: 47
4.2 Traceability Relations 149

5. RESULTS AND DISCUSSIONS **175**

6. CONCLUSION **177**

REFERENCES **179**

BIOGRAPHY **184**

ACRONYMS

CMMI	: Capability Maturity Model Inegration
SPI	: Software Process Improvement
SEI	: Software Engineering Ensititute
IT	: Information Technology
ROI	: Return on Investment
OSD	:Office of the Secretary of Defense
GQM	:Goal/Question/Metric
CAR	:Causal Analysis and Resolution
CM	:Configuration Management
DAR	:Decision Analysis and Resolution
IPM	:Integrated Project Management
MA	:Measurement and Analysis
OID	:Organizational Innovation and Deployment
OPD	:Organizational Process Definition
OPF	:Organizational Process Focus
OPP	:Organizational Process Performance
OT	:Organizational Training
PI	:Product Integration
PMC	:Project Monitoring and Control
PP	:Project Planning
PPQA	:Process and Product Quality Assurance
QPM	:Quantitative Project Management
RD	:Requirements Development
REQM	:Requirements Management
RSKM	:Risk Management
SAM	:Supplier Agreement Management
TS	:Technical Solution
VAL	:Validation
VER	:Verification
COBIT	: Control Objectives for Information and related Technology
RUP	:Rational Unified Process

TABLE LIST

		Page Number
Table 1.1	History of SPI ……..	5
Table 2.1	Comparison of Well vs. Poorly Designed Processes ……...	12
Table 2.2	Representation Capability and Maturity Levels ……..	27
Table 2.3	CSF's Identified Through Literature and Empirical Study …….	37
Table 3.1	CMMI & Process Framework Compliance List……	41

FIGURE LIST

Page Number

Figure 2.1 : Frameworks Quagmire..... 18
Figure 2.2 : Process Management Process Areas..... 30
Figure 2.3 : Project Management Process Areas..... 31
Figure 2.4 : Engineering Process Areas..... 34
Figure 2.5 : Support Process Areas..... 36
Figure 4.1 : Requirement Management – Introduction..... 47
Figure 4.2 : Requirement Management – Concepts..... 49
Figure 4.3 : Requirement Management – Workflow..... 50
Figure 4.4 : Requirement Management – Workflow – Analyze the Problem. 51
Figure 4.5 : Requirement Management – Workflow – Understand Stakeholder Needs..... 53
Figure 4.6 : Requirement Management – Workflow – Define the System..... 54
Figure 4.7 : Requirement Management – Workflow – Manage the Scope of the System..... 55
Figure 4.8 : Requirement Management – Workflow – Refine the System Definition..... 57
Figure 4.9 : Requirement Management – Workflow – Manage Changing Requests..... 58
Figure 4.10 : Requirement Management– Activities..... 60
Figure 4.11 : Requirement Management– Artifacts..... 61
Figure 4.12 : Analysis & Design – Introduction..... 63
Figure 4.13 : Analysis & Design – Concepts..... 64
Figure 4.14 : Analysis & Design – Workflow..... 65
Figure 4.15 : Analysis & Design – Workflow - Perform Architectural Synthesis..... 66
Figure 4.16 : Analysis & Design – Workflow - Define a Candidate Architecture..... 67
Figure 4.17 : Analysis & Design – Workflow – Refine Architecture..... 68
Figure 4.18 : Analysis & Design – Workflow – Analyze Behavior..... 71
Figure 4.19 : Analysis & Design – Workflow – Design Components..... 73
Figure 4.20 : Analysis & Design – Workflow – Design the Database..... 76
Figure 4.21 : Analysis & Design – Activities..... 78
Figure 4.22 : Analysis & Design – Workflow – Artifacts..... 79
Figure 4.23 : Implementation – Introduction..... 80
Figure 4.24 : Implementation – Concepts..... 81
Figure 4.25 : Implementation – Workflow..... 82
Figure 4.26 : Implementation – Structure the Implementation..... 83
Figure 4.27 : Implementation – Workflow – Plan the Integration..... 84
Figure 4.28 : Implementation – Workflow – Implement Components..... 85
Figure 4.29 : Implementation – Workflow - Integrate Each SubSystem..... 86
Figure 4.30 : Implementation – Workflow - Integrate the System..... 88

Figure 4.31 : Implementation – Workflow – Activities..... 89
Figure 4.32 : Implementation – Artifacts..... 90
Figure 4.33 : Test – Introduction..... 91
Figure 4.34 : Test – Concepts..... 93
Figure 4.35 : Test – Workflow..... 94
Figure 4.36 : Test – Workflow – Define Evaluation Mission..... 95
Figure 4.37 : Test – Workflow - Verify Test Approach..... 96
Figure 4.38 : Test – Workflow – Validate Build Stability..... 99
Figure 4.39 : Test – Workflow – Test and Evaluate..... 101
Figure 4.40 : Test – Workflow – Achieve Acceptable Mission..... 103
Figure 4.41 : Test – Workflow – Improve Test Assets..... 105
Figure 4.42 : Test – Activities..... 107
Figure 4.43 : Test – Workflow – Artifacts..... 108
Figure 4.44 : Deployment – Introduction..... 109
Figure 4.45 : Deployment – Concepts..... 110
Figure 4.46 : Deployment – Workflow..... 111
Figure 4.47 : Deployment – Workflow – Plan Deployment..... 112
Figure 4.48 : Deployment – Workflow – Develop Support Material..... 113
Figure 4.49 : Deployment – Workflow – Manage Acceptance Test..... 114
Figure 4.50 : Deployment – Workflow – Produce Deployment Unit..... 115
Figure 4.51 : Deployment – Workflow - Beta Test Product..... 116
Figure 4.52 : Deployment – Workflow – Package Product..... 117
Figure 4.53 : Deployment – Workflow – Provide Access to Download Site.... 118
Figure 4.54 : Deployment – Activities..... 119
Figure 4.55 : Deployment – Artifacts..... 120
Figure 4.56 : Change Management – Introduction..... 121
Figure 4.57 : Change Management – Concepts..... 123
Figure 4.58 : Change Management– Workflow..... 124
Figure 4.59 : Change Management– Workflow – Plan Project Configuration 125
Figure 4.60 : Change Management– Workflow – Create Project Configuration..... 126
Figure 4.61 : Change Management– Workflow – Manage Baselines and Releases..... 127
Figure 4.62 : Change Management– Workflow – Change and Deliver Configuration Items..... 128
Figure 4.63 : Change Management– Workflow – Monitor & Report Configuration..... 130
Figure 4.64 : Change Management– Workflow – Manage Change Request.... 131
Figure 4.65 : Change Managemen – Activities..... 132
Figure 4.66 : Change Management– Artifacts..... 133
Figure 4.67 : Project Management – Introduction..... 134
Figure 4.68 : Project Management – Concepts..... 135
Figure 4.69 : Project Management– Workflow..... 136
Figure 4.70 : Management– Workflow – Conceive New Project..... 137
Figure 4.71 : Management– Workflow – Evaluate Project Scope and Risk..... 138
Figure 4.72 : Project Management– Workflow – Plan the Project..... 139
Figure 4.73 : Project Management– Workflow – Plan for Next Iteration..... 140
Figure 4.74 : Project Management– Workflow – Manage Iteration..... 141
Figure 4.75 : Project Management– Workflow – Monitor and Control Project 143

Figure 4.76 : Project Management– Workflow – Close Out Phase..... 144
Figure 4.77 : Project Management– Workflow – Close Out Project..... 146
Figure 4.78 : Project Management– Activities..... 147
Figure 4.79 : Project Management– Artifacts..... 148
Figure 4.80 : Traceability Relation..... 150
Figure 4.81 : Traceability – Create Project..... 152
Figure 4.82 : Traceability – Create Project – User..... 153
Figure 4.83 : Traceability – Create Project – Confirmation..... 154
Figure 4.84 : Traceability – List Projects..... 155
Figure 4.85 : Traceability – Login..... 156
Figure 4.86 : Traceability – Disciplines..... 157
Figure 4.87 : Traceability – Artifact..... 158
Figure 4.88 : Traceability – Versioning..... 159
Figure 4.89 : Traceability – Template..... 160
Figure 4.90 : Traceability – Select Artifact..... 161
Figure 4.91 : Traceability – Requirements..... 162
Figure 4.92 : Traceability – Ceate Feature..... 163
Figure 4.93 : Traceability – Requirement Create..... 165
Figure 4.94 : Traceability – Create HLD Component..... 166
Figure 4.95 : Traceability – Create HLD..... 167
Figure 4.96 : Traceability – List Requirements..... 168
Figure 4.97 : Traceability – HLD Update..... 169
Figure 4.98 : Traceability – Plan..... 170
Figure 4.99 : Traceability – Create Project..... 171
Figure 4.100 : Traceability – Open MS Project..... 172
Figure 4.101 : Traceability – Update MS Project..... 172
Figure 4.102 : Traceability – Update Database..... 173
Figure 4.103 : Traceability – List HLD..... 174

CMMI UYGULAMA ALTYAPI SİSTEMİ

ÖZET

Tezin konusu, sistematik ve en iyi yazılım mühendisliği tecrübelerine dayanan bir süreç yönetimi yazılımı geliştirmektir. Tasarlanan ve geliştirilen yazılım, yazılım mühendisliğinin

- proje yonetimi,
- gereksinim yönetimi,
- analiz ve tasarm,
- uygulama geliştirme,
- test,
- değişiklik yönetimi ve
- aktarım süreçlerinde

gerekli olan adımlarının bir bütün içinde, yönetilebilir bir şekilde tutulmasını hedeflemektedir.

Her bir süreç alanı için literatürde ve uygulamada cok kapsamlı arastırmalar ve ürünler bulunmaktadır, ama son dönemde yapılan araştırmalar ve firmaların edindiği pratik tecrübeler göstermiştir ki, yazılım mühenisliğinden gercek anlamda fayda alınması süreçlerin bir bütün halinde işlemesine bağlıdır. Günümüz dinamikleri içinde pazar gereksinimleri, çok hızlı değişen teknolojiler ve rekabet piyasası, bilgi teknolojileri firmalarını çok zorlamaktadir. Koşullar söz konusu olduğunda firmalar her ne kadar yazılım mühendisliği süreçleriyle işlerini çok kolaylaştıracak olsalar da, pratikte bu süreçlerin gereksinimleri fazladan iş olarak görülmektedir.

Yazılım mühendisliği alalnında tecrübeli süreç mühendisleri tarafından süreç altyapısı kurulmadığı takdirde de bu yaklaşım çoğu zaman doğrudur. Süreçler aslında bir bütündür ve birbirleriyle olan ilişkileri çok kuvvetlidir. Bu ilişkiler göz önüne alınmayıp özel süreçler üzerine yoğunlaşmak sonucunda üretilen çıktılar genelde sadece dökümantasyon amaçlı kullanımın önüne geçemezler.

Projenin özelliği yazılım geliştirmenin temel süreçlerinin kontrollü bir şekilde belirlenmiş bir formatta uygulanmasını sağlamaktır. Geliştirilen sistem, hem süreç adımı ve uygulanışı hakkında bilgi vermekte, hem de üretilen çıktıların saklanması, değişiklerin yönetilebilmesi için versiyonlanması işlemlerini gerçekleştirmektedir. Süreçlerin bir bütün halinde çalıştığı ve birbirlerinin ürettikleri çıktıları otomatik kullanarak yeni çıkarımlar yaptığı bir altyapı bulunmaktadır.

Yazılım mühendisliği alanında çalışan üniversiteler, yazılım geliştirme enstitüleri gibi merkezler tarafından süreçlerin detaylarını ve aralarındaki ilişkileri gösteren pek çok model üretilmiştir. Beklenen bu modellere dayanarak şirketlerin kendilerine en uygun süreç yönetimi altyapısını oluşturmalarıdır. Bu nedenden dolayı da olayın bütününü gösteren, firmalara bir bütün halinde sunan araçlar bulunmamaktadır. Büyük ölçekli firmalar kalite ve süreç konuları için ayrı bir ekip ayırıp uyarlama

çalışmalarını sürdürürken, orta ve küçük ölçekli firmalar kaynak ve zaman sıkıntısı nedeniyle çalışmaları mümkün olmamaktadır.

Tezde amaçlanan orta ve küçük ölçekli firmaların süreç yönetimi konularına uygulamanın en hızlı ve en doğru yapabilecekleri bir ortam sunmaktır. Temel alınan sistem şu anda en yaygın kullanılan Canegie Mellon üniversitesinin kurduğu Software Engineering Enstitusunun (SEI) geliştirmiş olduğu Capability Maturity Model Integration modelidir. Tezin en önemli özelliği CMMI modelinde yer alan temel özellik olan izlenebilirliği firmalara uygulamak ve en yüksek faydayı almalarını sağlamaktır. Gereksinim üretimi ve yönetimiyle başlayan süreç, veritabanına girilen gereksinimlerden otomatik proje planlarının üretilmesi ile devam etmekte ve yazılım için üretilen proje planı üzerinden yine otomatik ilerleme ve tamamlama yüzdeleri alınabilmektedir.

CMMI IMPLEMENTATION FRAMEWORK

SUMMARY

This thesis studies the development of a systematic software solution to provide software engineering process management based on best practices. This software solution which has been designed and developed for this study aims to provide specific practices for integrated management of the following:

- project management,
- requirements management,
- analysis & design,
- implementation
- testing,
- change management, and
- deployment

There are various in depth research analyses as well as products that provide solutions to the abovementioned specific process areas. However it has also been recently recognized by various studies as well as through practical experience that it is crucial to operate all of these process areas in an integrated fashion to expedite maximum outcome from a software engineering study.

Today's information technology (IT) companies are driven by technologies that rapidly change, a vicious competition environment as well as steep market requirements. As much as software engineering is recommended as a process improvement solution to optimize these companies' output areas, it is however still practically seen as additional overhead. As a matter of fact, this usually turns out to be the fact if the process infrastructure is not developed and managed by an experienced team of process engineers.

In order to maximize the outcome of software engineering, the said processes should be treated as a united process flow, and the interdependences of each process should be treated exclusively. If these interdependencies are ignored and only specific process areas are focused on, the software engineering outputs can only help with documentation of existing inefficiencies at best.

The solution provided with this thesis focuses on management and execution of the key processes of software management and development in a controlled environment and format. The solution provides an information base on the specific process steps and their implementation as well as a storage management system for processed outcome, change management and versioning operations. Its infrastructure provides a business flow that establishes an integrated environment for all the said processes which can utilize each others outputs to provide a synergetic outcome.

There has been many models developed by universities as well as software development institutes which focus on process details as well as interdependencies between these processes. The expectation from an IT company is to choose and

integrate the best fitting model among these into their business practices. This, however, presents an additional business challenge to the companies as there is not one common tool that provides an integrated approach to the entirety of these process areas from a higher level approach. While large scale companies can afford separate teams for study and implementation quality and process activities, medium and small scale companies usually cannot afford such luxuries in neither planning nor implementation phases due to lack of time and resources.

Accordingly, this thesis aims to establish an environment that provides fast and applicable adaptation to such software engineering processes for small and medium scale companies. The solution provided within this thesis study is based on one of the most popular models developed by Carnegie Mellon University (U.S.) Software Engineering Institute, the Capability Maturity Model Integration (CMMI) model.

The main focus of this study is to implement the main target of CMMI, namely traceability to the companies to obtain maximum results from software engineering. The study follows a process that starts with production and management of requirements, continues with automatic development of project items and plans from the requirements database, and provides an environment for automatic project tracking and completion analysis.

1. INTRODUCTION

1.1 Thesis Introduction

Despite millions of software and IT professionals globally and the ubiquitous social presence of software, software engineering has only recently reached the status of a legitimate engineering discipline and a recognized profession. It is a key milestone in all disciplines to achieve consensus by the profession on a core body of knowledge. This has also been identified by the IEEE Computer Society as crucial for the evolution of software engineering towards professional status.

The IEEE Computer Society defines software engineering as follows:

(1) The application of a systematic, disciplined, quantifiable approach to the development, operation, and maintenance of software; that is, the application of engineering to software

(2) The study of approaches as in (1) [1].

This thesis studies the development of a systematic software solution to provide software engineering process management based on best practices. This software solution which has been designed and developed for this study aims to provide specific practices for integrated management of the following:

- requirements management,
- project management,
- analysis & design,
- implementation
- testing,

- change management, and
- deployment

There are various in depth research analyses as well as products that provide solutions to the abovementioned specific process areas. However it has also been recently recognized by various studies as well as through practical experience that it is crucial to operate all of these process areas in an integrated fashion to expedite maximum outcome from a software engineering study.

Today's information technology (IT) companies are driven by technologies that rapidly change, a vicious competition environment as well as steep market requirements. As much as software engineering is recommended as a process improvement solution to optimize these companies' output areas, it is however still practically seen as additional overhead. As a matter of fact, this usually turns out to be the fact if the process infrastructure is not developed and managed by an experienced team of process engineers.

In order to maximize the outcome of software engineering, the mentioned processes should be treated as a united process flow, and the interdependences of each process should be treated exclusively. If these interdependencies are ignored and only specific process areas are focused on, the software engineering outputs can only help with documentation of existing inefficiencies at best.

The solution provided with this thesis focuses on management and execution of the key processes of software management and development in a controlled environment and format. The solution provides an information base on the specific process steps and their implementation as well as a storage management system for processed outcome, change management and versioning operations. Its infrastructure provides a business flow that establishes an integrated environment for all the said processes which can utilize each others outputs to provide a synergetic outcome.

There has been many models developed by universities as well as software development institutes which focus on process details as well as interdependencies between these processes. The expectation from an IT company is to choose and integrate the best fitting model among these into their business practices. This,

however, presents an additional business challenge to the companies as there is not one common tool that provides an integrated approach to the entirety of these process areas from a higher level approach. While large scale companies can afford separate teams for study and implementation quality and process activities, medium and small scale companies usually cannot afford such luxuries in neither planning nor implementation phases due to lack of time and resources.

Accordingly, this thesis aims to establish an environment that provides fast and applicable adaptation to such software engineering processes for small and medium scale companies. The solution provided within this thesis study is based on one of the more popular models developed by Carnegie Mellon University (U.S.) Software Engineering Institute, the Capability Maturity Model Integration (CMMI) model.

The main focus of this study is to implement the main target of CMMI, namely tracebility to the companies to obtain maximum results from software engineering. The study follows a process that starts with production and management of requirements, continues with automatic development of project items and plans from the requirements database, and provides an environment for automatic project tracking and completion analysis.

1.2 General Concepts

As explained in the "Thesis Introduction" todays companies are dealing with software engineering. In a very competitive IT world, it is not enough just to build the product. Todays key factor is to build the product in quality. The quality of a system is highly influenced by the quality of the process used to acquire, develop, and maintain it.

This is area of software engineering. Every company, if they are alive and selling products are dealing somewhat with processes. So they are all doing software engineering. But the question is, whether their product is in good quality or not. The answer of this question will determine their profit, their market share, their growth capability. To compete with other companies, they have to improve the quality concept. To improve the quality of the product, they have to improve the way they produce it. They have to improve their processes.

Software Process Improvement (SPI) is an approach to design and define new and improved software processes to achieve basic business goals and objectives. Examples to these goals include increased revenues and profitability as well as decreased operating costs. The major benefits of SPI include quality, cost savings, cycle time reduction, increased customer satisfaction and productivity. It is also the means by which software companies can achieve significant increases in profitability and peak operating efficiency.

SPI is used to manipulate or change software processes to increase revenues or profits and decrease operating costs. This is accomplished by measuring the performance of an old software process, improving the process, and then implementing it. SPI also consists of measuring the performance of new software processes and institutionalizing them if they have improved.

It should be noted that the benefits of SPI provide the basis for calculating the return of investment (ROI) of SPI. Hence, SPI and the ROI of SPI are inseparably linked by basic origin, purpose, and function.

There are lots of SPI models since 1980's. Capability Maturity Model Integration (CMMI) is a popular process improvement approach that provides organizations with the essential elements of effective processes. The Software Engineering Institute (SEI) , which was established at Carnegie Mellon University in December of 1984 to address the need for improved software in U.S. Department of Defense operations, developed the Software Process Maturity Model for use both by the Department of Defense and by industrial software organizations.

CMMI helps integrate traditionally separate organizational functions, set process improvement goals and priorities, provide guidance for quality processes, and provide a point of reference for appraising current processes.

1.3 Literature and Theoretical Focus

There are various issues relating to the software process management system for software process generation and improvement in the literature.

1.3.1 SPI Method

Recently a lot of software process improvement(SPI) models and approaches have been suggested. Top-down approaches, assessment based approach, provide a high-level model of processes comprised of best practices in a software development organization(e.g., CMM, TRILLIUM, BOOTSTRAP, SPICE). They are based on descriptive and unstructured representations about what a software process ought to be, resulting in the difficulties to implement improvement initiatives as a software process model. On the other hands, bottom-up approaches start with understanding the processes that the organization owns. A process improvement is conducted based on measurement and experience. These approaches include Software Engineering Laboratory(SEL)'s approaches at the NASA and Goal/Question/Metric(GQM) approach [2]. It is difficult to reuse the mechanisms and knowledge of bottom-up without huge experience base. Table 1.1 shows the history of SPI.

Table 1.1: History of SPI

Year	Model /Standard
1983	NQI/CAE: 1st Canadian Award for Excellence (Canada)
1987	ISO 9001 released (initial release)
	NIST/MBNQA: 1st Malcolm Baldrige National Quality Award (USA)
	SEI-87-TR-24 (SW-CMM questionaire) released
1988	AS 3563 (Software Quality Management System) standard released
1991	IEEE 1074 released (initial release)
	ImproveIT V1.0 released (this is the beginning of TickIT)
	ISO 9000-3 released (initial release)
	SEI SW-CMM V1.0 released (initial release of model)

	Trillium V1.0 released (initial release)
1992	EFQM/BEA: 1st Business Excellence Award (Europe)
	IEEE adopts the Australian AS 3563 as "IEEE 1298"
	Control Objectives for Information and related Technology (COBIT)
	TickIT V2.0 released
1993	SEI SW-CMM V1.1 released
1994	ISO 9001 re-released
	Trillium V3.0 released
1995	ISO 12207 released (initial release)
	ISO 15504 (SPICE) initial "draft" released
1996	IEEE/EIA 12207 released
	COBIT v1 released
1997	ISO 9000-3 re-released
	SEI halts SW-CMM revisions in support for CMM Integration (CMMI)
1998	ISO 15504 (SPICE) released to public as "type 2" Technical Reports
	COBIT v2
	TickIT V4.0 released
2000	ISO 9000:2000 edition released
	SEI CMMI V1.02 released
	COBIT v3 released

2005	COBITv4 released
2006	CMMI for Development released
2007	CMMI for Aquisiton released

1.3.2 Software Process Reuse Repository

The concept of a Process Asset Library(PAL) has been introduced as an organizational repository for processes, supporting future reference and reuse. A prototype has been developed at SEI, as reported in [3], Other related work includes the "Experience Factory" concept of Basili and Rombach[4].

1.3.3 Software Process Evolution and Change Management

Some forms of planned changes are supported by several Policies and Mechanisms to Support Process Evolution (PSEE). A first class of PSEEs offers ad-hoc features to support process model evolution and contains an embedded policy of change. This is the case of MELMAC and Marvel. A second class 'of PSEEs is based on reflective PMLs which provide means to model the meta-process as part of the process model, and to manipulate the process model as any other process data, EPOS, IPSE 2.5, and SPADE offer this kind of support. In most cases, however, these systems offer only the ability to manipulate template variations [5].

1.3.4 Knowledge-based Software Process Model

We can find several knowledge based software process modeling environments. TAME project suggested a top-down goal-oriented approach to model and executes software engineering activities. In PROGEN project of George Mason University, a knowledge-based system was presented to generate, tailor and reuse processes [6].

1.4 Strength of the Process Improvement in Practice

Every process improvement methodology has its respective strengths and shortcomings.

But the common and main problem comes into play when interpreting the model to the organization and implementing these guidelines or best practices into a working environment. “Viewing software processes as blueprints emphasizes that design is separate from use, and thus that software process designers and users are independent. In the approach presented here, software processes are viewed as recipes; developers individually and collectively design their own software processes through facilitation, reflection, and improvisation“ [7]. This has to do with the concept of institutionalization. In case, if a middle or small scale company wants to begin a process improvement work, then it is very difficult for the organization to build a team and work on this improvement project in a long time. The improvement has a cost for company. This cost seems bigger and not worth under market and customer pressure. There are some published models for replacing the implementation need. “Software process improvement is a demanding and complex undertaking. To support the constitution and implementation of software process improvement schemes the Software Engineering Institute (SEI) proposes a framework, the so-called IDEAL model.” [8]. The most famous one is the IDEAL model. But also this model is a load for small scale companies. There are a lot of failure stories in literature. They are deeply researched. The capability maturity model (CMM) approach to software process improvement is the most dominant paradigm of organizational change that software organizations implement. While some organizations have achieved various levels of success with the CMM, the vast majority have failed. The thesis investigate the assumptions about organizational culture embedded in the CMM models and discuss their implications for software process improvement (SPI) initiatives. The well-known competing values are utilize model to surface and analyze the assumptions underlying the CMM. The analysis reveals contradictory sets of assumptions about organizational culture in the CMM approach. An understanding of these contradictions can help researchers address some of the difficulties that have been observed in implementing and institutionalizing SPI programs in organizations. Further, this research can help to open up a much-needed line of research that would examine the organization theory assumptions that underpin CMM. This type of research is important if CMM is to evolve as an effective organizational change paradigm for software organizations. [9]. In some cases, alltough it is seen that one of the benefits of process

improvement is to increase market share, it could be the opposite way in a small company if it is not managed well. “It may hurt the competitiveness of small companies and companies in highly innovative markets that according to the empirical study” [10].

1.5 Target of the Thesis

The target of the thesis is to find a solution for the strength of implementation of SPI models (specially CMMI) to the medium and small scale companies. “The research found that small businesses are faced not only with a lack of resources and funds required to implement many of the practices stated in the CMM, but also with the task of basing their process improvement initiatives on practices that do not apply to a small business and small software organization” [11]. Lots of models and standarts notice that company culture, business needs and policies are the most important aspects of process improvement project. But if small scale companies are the group which will be focused on this thesis, it will reported that the failure reasons shows some similarites. So the solution has to be created depending on these concerns.

Assuming that their failure is common, it will be thought that a common methodology could also be a solution to this problem. Common problems on institutionilization of process improvement methodologies are:

- Using single-discipline models that can result in confusion and higher costs.
- Customer rules
- Lack of experience and skill in provess improvement
- Resource problem, not enough time for process improvement actvities
- Resource problem, not enuogh time to create and maintain required artifacts
- No quantitative feedback on progress
- Wrong interpretation of improvement models

- Limited communication, project members could not see the project as a whole
- Do not really understand, need and use artifacts
- No overall traceability

At the end of this research, there will be a solution to every item listed above.

2. MAIN CONCEPTS

2.1 SPI

2.1.1 What is SPI?

SPI provides creation of new and improved software processes to achieve some level of benefits. These benefits are increased revenues or profits, decreased costs, and significant cost savings. It should be noted early attempts at SPI were designed to improve quality and reliability at any cost, however today it has also evolved to include cost savings.

The benefit cycle of SPI can be summarized as follows:

- Faster cycle times, shorter time to market, higher customer satisfaction, and alignment with strategic goals,
- Improved project management within the SPI framework, more accurate time and budget accounting as well as better cost and schedule performance,
- Lower defect rates, smaller module sizes, increased verification and validation efficiency, and increased productivity,
- Improvement certainly leads to better cost, quality, and reliability estimation and higher software quality and reliability.

SPI is used to create a new and improved software processes.

- Initially, statistical process control is used to measure the performance of an old software process.
- Then, a new and simplified software process is formed to improve performance.

- Usually the new and improved software process is piloted to measure its new performance.

- Finally, the new software process which exhibits the desired performance level may be institutionalized.

SPI is used to create new software processes for strategic software activities such as software project management, software quality management, and most importantly, software design management.

SPI of processes for software quality management is a proven discipline which yields orders-of-magnitude improvement. SPI of processes for software project management is starting to achieve international recognition. It is fueled by emerging data and hard economic justification for this discipline. SPI of processes for software design management is a discipline. Its economic underpinnings are anchored in the fields of software reuse and product line management.

2.1.2 Importance of SPI

SPI is the primary means by which a new and improved software process is created to achieve significant economic benefits at the least possible costs. A comparison of well-designed versus poorly designed software processes can be found below:

Table 2.1: Comparison of Well vs. Poorly Designed Processes

Well Designed Software Processes	**Poorly Designed Software Processes**
Positive bottom-line economic effects	Negative bottom-line economic effects
Increased Productivity	High cost of operations
Increased cost efficiency	Inefficient use of resources
Decreased costs	Lost of market opportunities and share
	Lack of quality & reliability
	Poor customer satisfaction & morale

In addition to its obvious benefits and the aforementioned ROI factor, SPI can also be used to create a new and improved software process to respond to a new industry standard. SPI is often performed to adhere to a new customer standard, lower operating capital, and changing skill requirements. Technological innovations, changes in organizational structures, and increased competition are also reasons to perform SPI. Thus, it can be said that SPI may be performed to effect incremental changes in operating efficiency as well as to support aggressive/new market maneuvers that require radically new software processes.

2.1.3 How is SPI Determined?

Statistical process control tools are used to determine SPI by measuring the performance of a new and improved process. Initially, the attributes or characteristics of an old software process are measured and analyzed to determine its performance. Then, the attributes or characteristics of a new software process are measured and analyzed to determine its performance. Typical attributes or characteristics include the following:

- effort (how many hours a process requires),
- cost (how much money a process requires),
- cycle time (how long a process takes),
- productivity (how many units a process yields),
- quality (how many defects a process yields),
- reliability (frequency of failures encountered),
- precision (exactness and conciseness),
- predictability (statistical accuracy),
- efficiency (resources consumed relative to process output),
- simplicity (process complexity),

- customer satisfaction (how well clients are served),
- degree of automation (a measure of eliminating the causes of human variation),
- consistency (a measure of minimal performance variation),
- repeatability (a measure of minimal performance variation),
- measurability (quantitative and often tangible or physical characteristic of a process or product),
- variety (a measure of process flexibility to satisfy multiple diverse customer requirements),
- innovation (a measure of the range and creativity of products and services).

2.1.4 Key Methods for SPI

Key methods for SPI consist of the following:

- general-purpose process improvement cycles
- general-purpose process improvement criteria
- software process modeling notations
- software engineering standards
- software engineering life cycles
- software engineering methodologies
- software engineering notations,
- software engineering processes,
- software engineering tools,
- software engineering measurement

2.1.5 General-Purpose Process Improvement Cycles

General-purpose process improvement cycles are used in conjunction with general-purpose process improvement criteria as the preferred methods. They are designed to be the basic frameworks necessary to begin the process of SPI. However, these frameworks tend to be diluted and ineffective at best, with little overall direction for improving software processes.

These methods are usually not recommended for novices who need specific help to identify high-impact and high-ROI SPI methods such as Six Sigma, statistical process control, plan-do-check-act, and initiating-diagnosing-establishing-acting-learning. Total quality management, total productivity management, and total cost management are also popular examples.

2.1.6 General-Purpose Process Improvement Criteria

These criteria are used in conjunction with general purpose process improvement cycles. General-purpose process improvement cycles tend to have an appraisal stage. This stage is used to leverage the specific requirements of general-purpose process improvement criteria and provides built-in mechanisms to:

- help organizations identify high-leverage areas for improvement.
- prioritize process improvements and utilize resources toward high-priority areas.

They tend to be more specific than general-purpose process improvement cycles and minimize some confusion for the novice. However, they tend to have so many criteria as to confuse and dilute the overall effectiveness of using them.

Examples of general-purpose process improvement criteria include ISO 9001, TL 9000, BOOTSTRAP, and TRILLIUM, The Malcolm Baldrige National Quality Award. There are also the following unique models, some of which constitute the basis for this thesis:

- Software Capability Maturity Model,

- Capability Maturity Model Integration,
- Systems Engineering Capability Maturity Model,
- Integrated Product Team Capability Maturity Model,
- Systems Security Engineering Capability Maturity Model,
- System Acquisition Capability Maturity Model,
- The Trusted Capability Maturity Model,
- People Capability Maturity Model, and
- Integrated Capability Maturity Model.

All of the models were created within an environment of evolving national and international standards and frameworks. As standards become used and accepted, maintaining harmonization between them and the improvement models becomes a continuing challenge, particularly across disciplines. In describing the complexity of this environment, Sarah Sheard of the Software Productivity Consortium has coined the term "the frameworks quagmire."[12] shows her depiction of the proliferation and heritage of the various systems and software engineering standards, life-cycle models, quality awards, and process-improvement models.

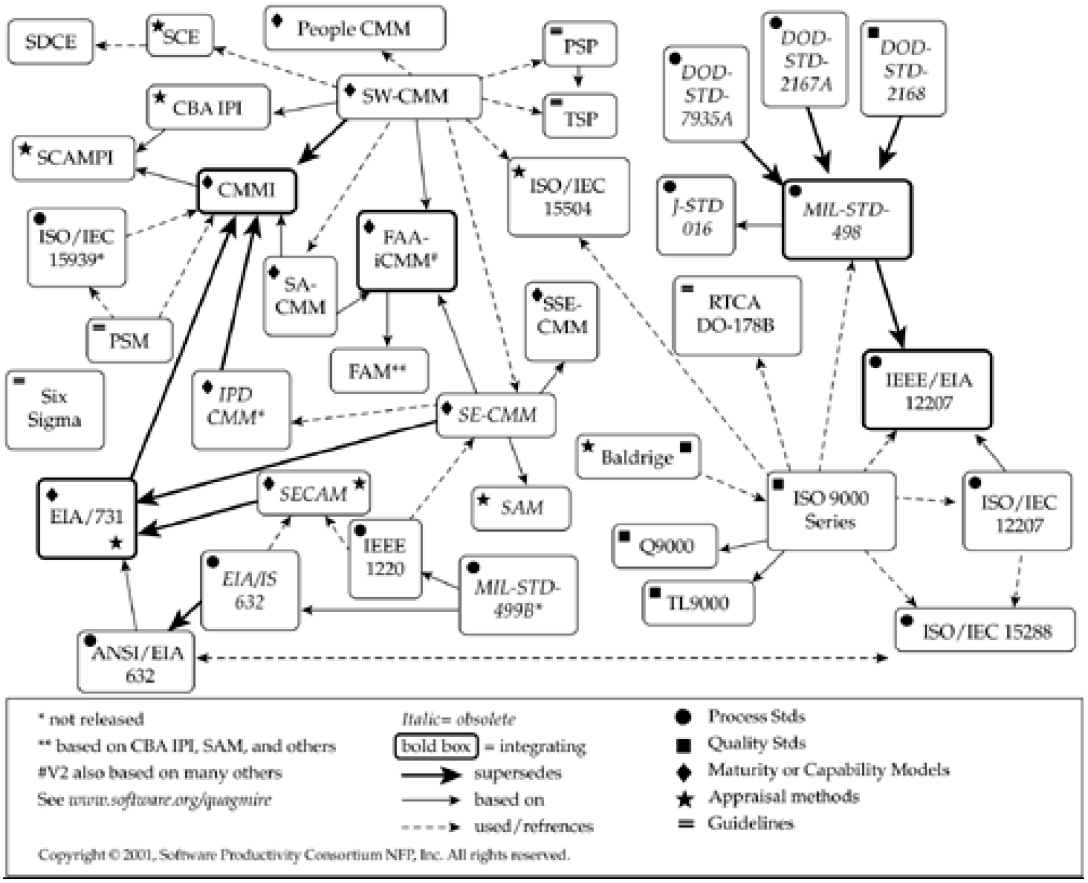

Figure 2.1: The Frameworks Quagmire[12]

In Figure 2.1, the arrows show where one model or standard contributed to the development of another.

The single disciplines and processes involved in contemporary engineering are closely intertwined. The overhead and confusion resulting from the application of multiple models are too costly in terms of business expenses and resource allocation. As a consequence, a means of addressing process improvement across anumber of disciplines within a single framework is needed. The bold boxes in the framework quagmire show integrated ones.

2.1.7 Software Process Modeling Notations

These are textual or visual aids designed to define and document software processes, and used to communicate, facilitate, and even use a new and improved software process.

Software process modeling notations may bring various challenges of use, as follows:

- Some are inadequate for expressing the depth of detail necessary to describe software processes which can lead to hindering the use, exploitation, and consistency of software processes.

- The choice of notation can lead to debilitating politics, which results in little progress toward the creation and use of a new and improved software process.

- Only one or two of many of the available notations may be effective. Few are recommended for defining new and improved software processes. These methods provide little direction for novices on what software processes to define and their depth of definition.

Examples of software process modeling notations include short checklists, textual descriptions, flowcharts, information mapping, input/output charts, and professional policy and procedure formats as well as proprietary notations built into workflow automation tools.

2.1.8 Software Engineering Standards

Software engineering standards are the minimum requirements for designing new software processes. These standards have greater breadth than general-purpose process improvement criteria and tend to offer better priorities for SPI. It should however also be noted that software engineering standards have much less depth than general-purpose process improvement criteria which can lead to ineffective guidance to achieve their purpose.

The recommended approach is to blend general-purpose process improvement criteria and software engineering standards to achieve a balance of both breadth and depth. Examples of software engineering standards include MIL-STD-1521B, MIL-STD-973, MIL-HDBK-61, and MIL-STD-2549, ISO 12207, and ISO 15288.

2.1.9 Software Engineering Life Cycles

Software engineering life cycles add integration, workflow, and tactical execution to software engineering processes to help organizations manage the design and development of software products and services. Unfortunately, software engineering life cycles lack the breadth of software engineering standards as well as the depth of general-purpose process improvement criteria. Instead, software engineering life cycles tend to offer much tactical guidance for novices. Examples include waterfall, spiral, evolutionary, prototyping, incremental, concurrent, concurrent incremental, and V model.

2.1.10 Software Engineering Methodologies

Software engineering methodologies are designed to string or thread multiple software engineering notations together to achieve the goal of specifying, designing, and implementing software-based systems. They tend to be based on graphical or mathematical notations to be used for capturing software requirements, software designs, and constructs for software implementation. Examples include structured analysis, structured design, information engineering, and object oriented analysis, object-oriented design, Clean Room, and Rational Unified Process (RUP) [13].

2.1.11 Software Engineering Notations

Software engineering notations are the building blocks of software engineering methodologies. They are used to create visual representations of software constructs to facilitate rational and logical software development, and also to influence software engineers to do more than just computer programming. Software engineering notations can be seen as the viewgraphs of SPI, and software engineering for that matter. Examples include data flow diagrams, state transition diagrams, entity relationship diagrams, control specifications, structure charts, and program design languages. Newer examples include the Object Modeling Technique and Unified Modeling Language UML.

2.1.12 Software Engineering Processes

Software engineering processes are designed to represent logical groupings of major software engineering activities. These standards are merely collections of software engineering activities, and are thought of as major sub-activities or sub-elements within the software life cycle. Configuration management is an example of a process that once embodied the entire discipline of software engineering. While some software engineering processes add negligible value, others offer an overwhelming amount of benefits.

Examples include software configuration management, software testing, and independent verification and validation. Commercial off-the-shelf integration, software architecture, and product line management are some of the latest examples.

2.1.13 Software Engineering Tools

Software engineering tools are designed to define and formalize software engineering processes, and automate tedious tasks that cannot be consistently performed by humans. They add great value, increase software productivity, and increase work product output, and perform many built-in verification and validation tasks.

Software engineering tools, fueled by SPI methods and computer systems, will answer many potential SPI challenges faced by corporations. Examples include computer-aided software engineering tools, software project management tools, software estimation tools, code generation tools, graphical user interface management systems, and automated static analysis tools. In addition. requirements management tools, office automation tools, Web-enabled tools, and operating systems are also good examples.

2.2 CMMI

Capability Maturity Model® Integration (CMMI) is a process improvement approach that provides organizations with the essential elements of effective processes. CMMI

is used to guide process improvement across a project, a division, or an entire organization.

CMMI helps integrate traditionally separate organizational functions, set process improvement goals and priorities, provide guidance for quality processes, and provide a point of reference for appraising current processes.

2.2.1 CMMI History & Background

Since 1984, the Carnegie Mellon® Software Engineering Institute (SEI) has served as a United States of America government-funded research and development center. As part of Carnegie Mellon University, the SEI has the following attributes [14] :

- SEI is sponsored by The Office of the Secretary of Defense (OSD) and the National Defense Industrial Association
- SEI capitalizes on the similarities of other process improvement models; eliminates differences that increase effort and expense of "stovepiping" models
- SEI began with the following source models:
    - SEI's Capability Maturity Model for Software (SW-CMM)
    - Electronic Industries Alliance Systems Engineering Capability Model, Interim Standard (EIA/IS 731) - the result of the merger of the SE-CMM, created by the Enterprise Process Improvement Collaboration (EPIC), and the SECAM, created by INCOSE
    - A draft model covering Integrated Product and Process Development (IPPD), the IPD-CMM, previously released in draft form by EPIC

2.2.2 CMMI Content

CMMI provides guidance for the managerial processes of companies. This includes establishing and maintaining a plan for managing the work, and making sure that everyone involved is committed to performing and supporting the plan. In addition,

when the plans are made, the development and maintenance costs, schedules, and product estimates should be available as well.

Upon execution of the plan, the performance and progress to the plan needs to be compared and corrective actions should be scheduled if actual and planned results are found to be out of sync. Agreements with suppliers should be established and maintained, and it should be made sure that these agreements are satisfied. Finally, there is also the management of the information on project risks and on creating and managing teams as well.

CMMI guidance on technical matters includes ways to develop, elaborate, and manage requirements, and to develop technical solutions that meet those requirements. CMMI is the key reminder of the fact that the integration of product components depends on good interface information, and it needs to be planned and verified. It should be made sure that the products and services that are developed are consistent with the requirements and satisfy the customer's needs through verification and validation practices [17].

CMMI also addresses support processes for technical and managerial activities. It provides methods of ensuring that the defined processes being followed and the products that are being developed meet the quality specifications that have been established. Finally, CMMI also helps figuring out the root cause of serious problems with the products or key processes.

2.2.3 CMMI Models

CMMI models describe what have been determined to be the best practices that organizations have found to be productive and useful to achieving their business objectives. The organizations must use professional judgment when interpreting the CMMI practices for their situation, needs, and business objectives. Although process areas depict the characteristics of an organization committed to process improvement, the organization must interpret the process areas using an in-depth knowledge of CMMI, the organization itself, the business environment, and the specific circumstances involved.

During the CMMI model initiation to improve the organization's processes, real-world processes are mapped to CMMI process areas. This mapping enables the organization to initially judge and later track the level of conformance to the CMMI model and to identify opportunities for improvement.

CMMI for Development is a reference model that covers the development and maintenance activities applied to both products and services. Organizations from many industries, including aerospace, banking, computer hardware, software, defense, automobile manufacturing, and telecommunications, use CMMI for Development [18].

Models in the CMMI for Development constellation contain practices that cover project management, process management, systems engineering, hardware engineering, software engineering, and other supporting processes used in development and maintenance.

Continuous Representation:

If the processes that need to be improved in the organization are already known and the dependencies among these processes understood, the continuous representation is the proper choice for the organization.

The continuous representation offers maximum flexibility when using a CMMI model for process improvement. An organization may choose to improve the performance of a single process-related trouble spot, or it can work on several areas that are closely aligned to the organization's business objectives. The continuous representation also allows an organization to improve different processes at different rates. There are some limitations on an organization's choices because of the dependencies among some process areas.

Staged Representation:

The staged representation offers a systematic, structured way to approach model-based process improvement one stage at a time. Achieving each stage ensures that an adequate process infrastructure has been laid as a foundation for the next stage.

Process areas are organized by maturity levels that take some of the guess work out of process improvement. The staged representation prescribes an order for implementing process areas according to maturity levels, which define the improvement path for an organization from the initial level to the optimizing level. Achieving each maturity level ensures that an adequate improvement foundation has been laid for the next maturity level and allows for lasting, incremental improvement.

2.2.4 Process Areas

A process area is a cluster of related practices in an area that, when implemented collectively, satisfy a set of goals considered important for making improvement in that area.

There are 22 process areas of CMMI, as listed in alphabetical order [15]:

- Causal Analysis and Resolution (CAR)
- Configuration Management (CM)
- Decision Analysis and Resolution (DAR)
- Integrated Project Management (IPM)
- Measurement and Analysis (MA)
- Organizational Innovation and Deployment (OID)
- Organizational Process Definition (OPD)
- Organizational Process Focus (OPF)
- Organizational Process Performance (OPP)
- Organizational Training (OT)
- Product Integration (PI)
- Project Monitoring and Control (PMC)

- Project Planning (PP)
- Process and Product Quality Assurance (PPQA)
- Quantitative Project Management (QPM)
- Requirements Development (RD)
- Requirements Management (REQM)
- Risk Management (RSKM)
- Supplier Agreement Management (SAM)
- Technical Solution (TS)
- Validation (VAL)
- Verification (VER)

2.2.5 Levels

CMMI uses "Levels" to describe an evolutionary path recommended for an organization that wants to improve the processes. Levels can also be the outcome of the rating activity of appraisals. Appraisals can be performed for organizations or for smaller groups such as a group of projects or a division within a company.

CMMI supports two improvement paths. One path enables organizations to incrementally improve processes corresponding to an individual process area (or process areas) selected by the organization. The other path enables organizations to improve a set of related processes by incrementally addressing successive sets of process areas.

These two improvement paths are associated with the two types of levels that correspond to the two representations discussed previously. For the continuous representation, the term "capability level" is used; for the staged representation, the term "maturity level" is used.

Regardless of which representation is selected, the concept of levels is the same. Levels characterize improvement from an ill-defined state to a state that uses quantitative information to determine and manage improvements that are needed to meet an organization's business objectives.

To reach a particular level, an organization must satisfy all of the appropriate goals of the process area or set of process areas that are targeted for improvement, regardless of whether it is a capability or a maturity level. Both representations provide the same essential content and use the same model components, as shown in the following table.

Table 2.2: Representation Capability and Maturity Levels

Level	**Continuous Representation Capability Levels**	**Staged Representation Maturity Levels**
Level 0	Incomplete	N/A
Level 1	Performed	Initial
Level 2	Managed	Managed
Level 3	Defined	Defined
Level 4	Quantitatively Managed	Quantitatively Managed
Level 5	Optimizing	Optimizing

Maturity Level 1: Initial

Processes are usually best defined as ad hoc and chaotic at maturity level 1. The

organization usually does not provide a stable environment to support the processes. Success in these organizations depends on the competence and heroics of the people in the organization and not on the use of proven processes.

In spite of this chaos, maturity level 1 organizations often produce products and services that work; however, they frequently exceed their budgets and do not meet their schedules.

Maturity level 1 organizations are characterized by a tendency to over commit, abandonment of processes in a time of crisis, and an inability to repeat their successes.

Maturity Level 2: Managed

At maturity level 2, processes are planned and executed in accordance with policy; the projects employ skilled people who have adequate resources to produce controlled outputs; involve relevant stakeholders; are monitored, controlled, and reviewed; and are evaluated for adherence to their process descriptions.

Maturity level 2 ensures that existing practices are retained during times of stress. When these practices are in place, projects are performed and managed according to their documented plans. In addition, the status of the work products and the delivery of services are visible to management at defined points along with commitments established among relevant stakeholders.

Maturity Level 3: Defined

At maturity level 3, processes are well characterized and understood, and are described in standards, procedures, tools, and methods. The basis of maturity level 3 actually consists of the organization's set of standard processes. These standard processes are used to establish consistency across the organization.

A critical distinction between maturity levels 2 and 3 is the scope of standards, process descriptions, and procedures. At maturity level 2, the standards, process descriptions, and procedures may be quite different in each specific instance of the process (e.g., on a particular project). At maturity level 3, the standards, process descriptions, and procedures for a project are tailored from the organization's set of standard processes to suit a particular project or organizational unit and therefore are more consistent, except for the differences allowed by the tailoring guidelines.

Another critical distinction is that at maturity level 3, processes are typically described more rigorously than at maturity level 2. A defined process clearly states the purpose, inputs, entry criteria, activities, roles, measures, verification steps, outputs, and exit criteria. At maturity level 3, processes are managed more

proactively using an understanding of the interrelationships of the process activities and detailed measures of the process, its work products, and its services.

Maturity Level 4: Quantitatively Managed

At maturity level 4, the organization and projects establish quantitative objectives for quality and process performance and use them as criteria in managing processes. Quantitative objectives are based on the needs of the customer, end users, organization, and process implementers.

Quality and process performance is understood in statistical terms and is managed throughout the life of the processes. For selected sub-processes, detailed measures of process performance are collected and statistically analyzed. Quality and process performance measures are incorporated into the organization's measurement repository to support fact-based decision making. Special causes of process variation are identified and, where appropriate, the sources of special causes are corrected to prevent future occurrences.

A critical distinction between maturity levels 3 and 4 is the predictability of process performance. At maturity level 4, the performance of processes is controlled using statistical and other quantitative techniques, and is quantitatively predictable. At maturity level 3, processes are typically only qualitatively predictable.

Maturity Level 5: Optimizing

At maturity level 5, an organization continually improves its processes based on a quantitative understanding of the common causes of variation inherent in processes.

Maturity level 5 focuses on continually improving process performance through incremental and innovative process and technological improvements. Quantitative process improvement objectives for the organization are established, continually revised to reflect changing business objectives, and used as criteria in managing process improvement [19]. The effects of deployed process improvements are measured and evaluated against the quantitative process improvement objectives.

A critical distinction between maturity levels 4 and 5 is the type of process variation addressed. At maturity level 4, the organization is concerned with addressing special

causes of process variation and providing statistical predictability of the results. Although processes may produce predictable results, the results may be insufficient to achieve the established objectives. At maturity level 5, the organization is concerned with addressing common causes of process variation and changing the process (to shift the mean of the process performance or reduce the inherent process variation experienced) to improve process performance and to achieve the established quantitative process improvement objectives.

2.2.6 Process Areas and their Relationships

Process areas can be grouped into four categories [15]. These areas often interact and have an effect on one another regardless of their defined group:

- Process Management
- Project Management
- Engineering
- Support

Process Management

Process Management contains the cross-project activities related to defining, planning, deploying, implementing, monitoring, controlling, appraising, measuring, and improving processes. It includes Organizational Process Focus (OPF), Organizational Process Definition (OPD) and Organizational Training (OT) process areas.

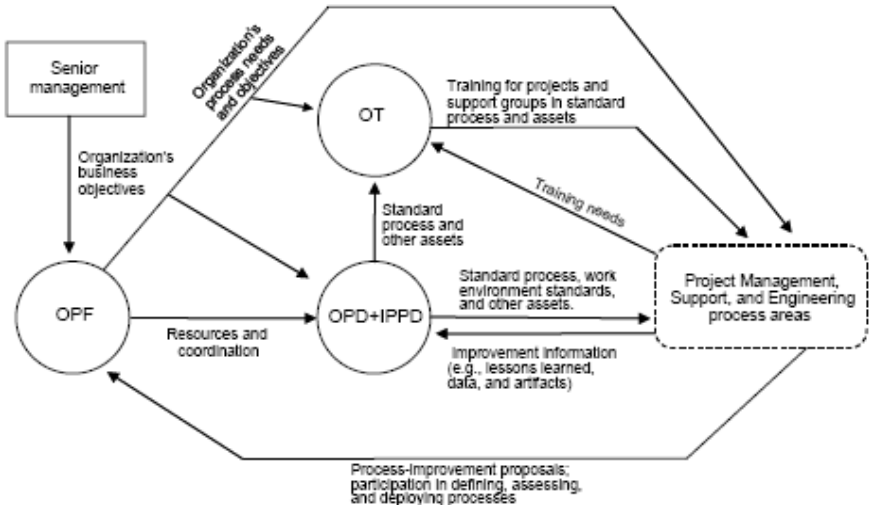

Figure 2.2: Process Management Process Areas [15]

The Organizational Process Focus (OPF) process area, as defined in the above Figure 2.2, helps the organization to plan, implement, and deploy organizational process improvements based on an understanding of the current strengths and weaknesses of the organization's processes and process assets.

Candidate improvements are obtained through various means such as process improvement proposals, measurement of the processes, lessons learned in implementing the processes, and results of process appraisal and product evaluation activities.

The Organizational Process Definition (OPD) process area establishes and maintains the organization's set of standard processes, work environment standards, and other assets based on the process needs and objectives of the organization. These other assets include descriptions of lifecycle models, process tailoring guidelines, and process-related documentation and data. Projects tailor the organization's set of standard processes to create their defined processes. The other assets support tailoring as well as implementation of the defined processes. Experiences and work products from performing these defined processes, including measurement data, process descriptions, process artifacts, and lessons learned, are incorporated as appropriate into the organization's set of standard processes and other assets[20] .

The Organizational Training (OT) process area identifies the strategic training needs of the organization as well as the tactical training needs that are common across projects and support groups. In particular, training is developed or obtained to develop the skills required to perform the organization's set of standard processes.

The main components of training include a managed training development program, documented plans, personnel with appropriate knowledge, and mechanisms for measuring the effectiveness of the training program.

Project Management

Project Management contains all project maintenance related process areas such as Project Planning (PP), Project Monitoring and Control (PMC), and the Supplier Agreement Management (SAM) process areas, as depicted in the following Figure 2.3.

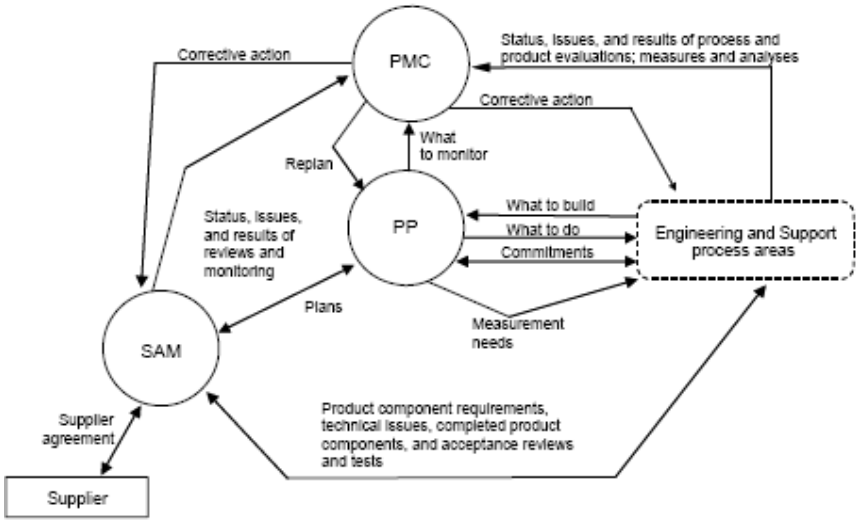

Figure 2.3: Project Management Process Areas [15]

The Project Planning (PP) process area includes developing the project plan, involving stakeholders appropriately, obtaining commitment to the plan, and maintaining the plan. Planning begins with requirements that define the product and project and the plan covers the various project management and development

activities performed by the project. These plans cover configuration management, verification, and measurement and analysis.

The Project Monitoring and Control (PMC) process area includes monitoring activities and taking corrective action. The project plan specifies the appropriate level of project monitoring, the frequency of progress reviews, and the measures used to monitor progress. Progress is determined primarily by comparing project status to the plan, and corrective actions, including re-planning, are taken as necessary.

The Supplier Agreement Management (SAM) process area addresses the portions of work that are produced by suppliers. As such. the supplier is selected, and a supplier agreement is established to manage the supplier. The supplier's progress and performance are tracked by monitoring selected work products and processes, and the supplier agreement is revised as appropriate. Acceptance reviews and tests are conducted on the supplier-produced product component.

Engineering

Engineering process area includes the complete end-to-end product engineering process, as described in Figure 2.4. It includes Requirements Development (RD), Technical Solution (TS), Product Integration (PI), Requirements Management (REQM), Verification (VER), and Validation (VAL) process areas.

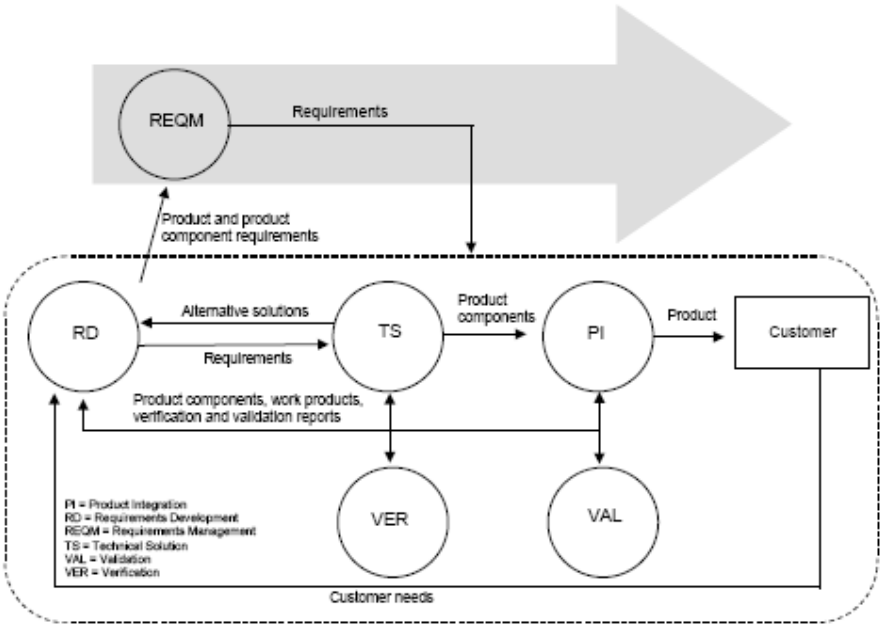

Figure 2.4: Engineering Process Areas [15]

The Requirements Development (RD) process area identifies customer needs and translates these needs into product requirements. The set of product requirements is analyzed to produce a high-level conceptual solution. This set of requirements is then allocated to establish an initial set of product component requirements. Other requirements that help define the product are derived and allocated to product components.

The Requirements Development process area supplies requirements to the Technical Solution (TS) process area, where the requirements are converted into the product architecture, the product component design, and the product component itself (e.g., coding and fabrication).

Requirements are also supplied to the Product Integration (PI) process area, where product components are combined and interfaces are verified to ensure that they meet the interface requirements supplied by Requirements Development.

The Requirements Management (REQM) process area maintains the requirements. It describes activities for obtaining and controlling requirement changes and ensuring that other relevant plans and data are kept current. It provides traceability of requirements from customer to product to product component. Requirements

Management ensures that changes to requirements are reflected in project plans, activities, and work products.

The Technical Solution (TS) process area develops technical data packages for product components that will be used by the Product Integration or Supplier Agreement Management process area. Alternative solutions are examined with the intent of selecting the optimum design based on established criteria. These criteria may be significantly different across products, depending on product type, operational environment, performance requirements, support requirements, and cost or delivery schedules. The task of selecting the final solution makes use of the specific practices in the Decision Analysis and Resolution process area.

The Technical Solution process area relies on the specific practices in the Verification (VER) process area to perform design verification and peer reviews during design and prior to final build. The Verification process area ensures that selected work products meet the specified requirements. The Verification process area selects work products and verification methods that will be used to verify work products against specified requirements. Verification is generally an incremental process and involves peer reviews.

The Validation (VAL) process area incrementally validates products against the customer's needs. Validation may be performed in the operational environment or in a simulated operational environment. Coordination with the customer on the validation requirements is an important element of this process area. The scope of the Validation process area includes validation of products, product components, selected intermediate work products, and processes.

The Product Integration (PI) process area contains the specific practices associated with generating the best possible integration sequence, integrating product components, and delivering the product to the customer. Product Integration uses the specific practices of both Verification and Validation in implementing the product integration process.

Support

The support process area mainly consists of Measurement and Analysis (MA), Process and Product Quality Assurance (PPQA), and the Configuration Management (CM) areas, as depicted in the following Figure 2.5.

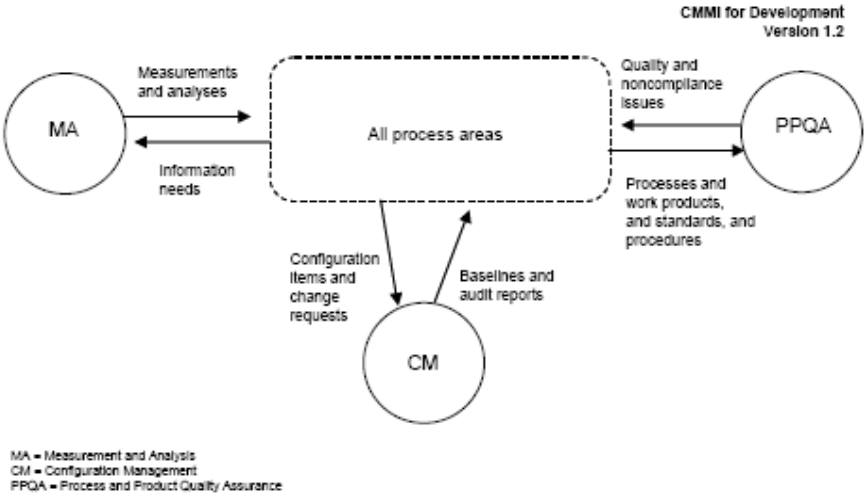

Figure 2.5: Support Process Areas [15]

The Measurement and Analysis (MA) process area supports all process areas by providing specific practices that guide projects and organizations in aligning measurement needs and objectives with a measurement approach that will provide objective results. These results can be used in making informed decisions and taking appropriate corrective actions.

The Process and Product Quality Assurance (PPQA) process area supports all process areas by providing specific practices for objectively evaluating performed processes, work products, and services against the applicable process descriptions, standards, and procedures, and ensuring that any issues arising from these reviews are addressed.

The Configuration Management (CM) process area supports all process areas by establishing and maintaining the integrity of work products using configuration identification, configuration control, configuration status accounting, and configuration audits. The work products placed under configuration management include the products that are delivered to the customer, designated internal work

products, acquired products, tools, and other items that are used in creating and describing these work products. Examples of work products that may be placed under configuration management include plans, process descriptions, requirements, design data, drawings, product specifications, code, compilers, product data files, and product technical publications.

2.3 Practical Implementation

It have been researched about a model, which have adopted a CMMI approach and developed a maturity model for SPI implementation in order to guide organizations in assessing and improving their SPI implementation processes. The basis of this model is the SPI literature and an empirical study. In the design of this maturity model the concept have extended the concept of critical success factors (CSFs). The model hasbeen conducted with 23 Australian practitioners. It has also analysed CSFs and critical barriers using 50 research articles (published experience reports and case studies) [16].

The Table 2.3 shows critical barriers to understand the nature of issues that undermine the SPI implementation programmes. The results are in comparison with the literature and an empirical study. The results show that most of the practitioners in literature consider lack of resources a major critical barrier for the implementation of SPI. The results also suggest that in practitioners' opinion time pressure and inexperienced staff can undermine the success of SPI implementation programmes. It shows that practitioners would prefer to avoid organizational politics during the implementation of SPI programmes [21].

Table 2.3: CSF's identified through literature and empirical study [16]

CSFs identified through literature and empirical study

Success factors	Occurrence in literature ($n = 47$)			Occurrence in CSF interviews ($n = 23$)		
	Frequency	%	Rank	Frequency	%	Rank
Assignment of responsibility of SPI	12	26	7	–	–	–
Clear and relevant SPI goals	12	26	7	–	–	–
Creating process action teams	15	31	5	2	9	8
Encouraging communication and collaboration	10	21	9	5	22	6
Experienced staff	13	28	6	8	35	4
Facilitation	–	–	–	6	26	5
Formal methodology	–	–	–	8	35	4
Managing the SPI project	7	15	10	3	13	7
Process ownership	11	23	8	–	–	–
Providing enhanced understanding	7	15	10	–	–	–
Reviews	13	28	6	2	9	8
Reward schemes	7	15	10	–	–	–
Senior management commitment	31	66	1	15	65	1
SPI awareness	–	–	–	12	52	2
Staff involvement	24	51	2	10	44	3
Staff time and resources	18	38	4	10	44	3
Tailoring improvement initiatives	7	15	10	2	9	8
Training and mentoring	23	49	3	15	65	1

Organizational politics is ranked highest in CSF interviews, i.e. 52%. Two new critical barriers—lack of formal methodology and lack of awareness—have been identified in our empirical study which have not been identified in the literature. The second most cited criticalbarrier in CSF interviews is lack of support. The critical barrier ''lack of resources'' is cited 35% in the CSF interviews.

Comparison of the critical barriers provides evidence that there are some clear similarities and differences between the findings of the two sets. There are seven barriers in common, i.e. inexperienced staff, lack of resources, lack of support, negative or bad experiences, organizational politics, paperwork required and time pressure.

There are also a number of differences between the findings. For example, ''changing the mindset of management and technical staff'' and ''staff turnover'' have not been cited in our empirical study but these barriers are present in the literature. Similarly, lack of awareness of SPI and lack of formal methodology are critical in our empirical study but have not been identified through the literature. This shows that practitioners, who took part in our study, are more concerned about SPI awareness activities and implementation methodology [22]. This is because

- SPI is an expensive and long-term approach that takes a long time to realise its real benefits. Hence, in order to get the support of management and

practitioners and to successfully continue SPI initiatives it is very important to provide and maintain sufficient awareness of SPI within organizations [23].

- Formal methodology has also emerged because little attention has been paid to the effective implementation of SPI initiatives. Studies show that 67% of SPI managers want guidance on how to implement SPI activities, rather than what SPI activities to actually implement. This new barrier suggests that in practitioners' opinion the lack of a formal SPI implementation methodology can undermine the implementation of SPI programmes [24][25].

3. PRESENTATION OF THE THESIS

3.1 Subject of the Thesis

The solution is a software engineering process framework and implementation tool. It provides a disciplined approach to assigning tasks and responsibilities within a development organization. Its goal is to ensure the production of high-quality software that meets the needs of its end users within a predictable schedule and budget.

There are three central elements that define the product:

- The underlying set of philosophies and practices for successful software development depending on CMMI.
- A process model repository, that includes all the versioned artifacts
- Automatic generated artifacts and traceability

If software development is a critical factor to the success of an organization, then the implementation tool will help the organization to built a process management framework. The implementation tool is developed with two primary groups of users:

- software development practitioners working as part of a project team, including the stakeholders of those software development projects.
- process engineering practitioners, specifically software process engineers and managers.

Software development practitioners can find guidance on what is required of them in the roles defined in process framework. A practitioner working on an implementation tool applied project is assigned to one or more of the roles defined in process framework, where each role partitions a set of activities and artifacts that role is

responsible for. Those roles collaborate in terms of the detailed work that is required to enact the workflow within an iteration.

Process engineers will have template process framework using the implementation tool. The only thing, they have to is create new projects and users. They can also work on defining, configuring, tailoring and implementing engineering processes.

3.2 Process Areas and CMMI Compliance

The process framework is designed to meet CMMI requirements. The Table 3.1 shows the compliance matrix between the created disciplines and CMMI process:

Table 3.1: CMMI & Process Framework Compliance List

CMMI	PROCESS FRAMEWORK
Causal Analysis and Resolution	N/A
Configuration Management	Change Management
Decision Analysis and Resolution	N/A
Integrated Project Management	Project Management
Measurement and Analysis	Project Management
Organizational Innovation and Deployment	Process Framework
Organizational Process Definition	Process Framework
Organizational Process Focus	Process Framework
Organizational Process Performance	Process Framework
Organizational Training	Process Framework
Product Integration	Deployment

Project Monitoring and Control	Project Management
Project Planning	Project Management
Process and Product Quality Assurance	Test
Quantitative Project Management	Project Management
Requirements Development	Requirements Management
Requirements Management	Requirements Management
Risk Management	Project Management
Supplier Agreement Management	N/A
Technical Solution	Implementation, Analysis & Design
Validation	Test
Verification	Test

3.3 Traceability Property

3.3.1 Requirements Management

The purpose of Requirements Management is to manage the requirements of the project's products and product components and to identify inconsistencies between those requirements and the project's plans and work products. Requirements Management has one specific goal: to manage requirements and identify inconsistencies with plans and work products. To manage requirements, the person or team that receives them needs to develop an understanding of what they mean before doing anything with the requirements. It should obtain a commitment from the people who implement the requirements. Once the requirements are received and understood, and a commitment is obtained, all changes to the requirements should be

managed, including recording change histories and evaluating change impacts. The project should provide for bidirectional traceability of the requirement and the associated plans and work products. Tracing the requirements provides a better basis for determining the ramifications of changes, and it ensures that all requirements have a parent and that the product design covers all high-level requirements. Finally, the project should identify inconsistencies between the requirements and the project plans and work products. Any corrective action required to fix inconsistencies is accomplished in the requirements development process, the project planning process, or possibly other processes.

3.3.2 Project Management

The tracebility part of Project Management is Project Monitoring and Control. The Purpose of Project Monitoring and Control is to provide an understanding of the project's progress so that appropriate corrective actions can be taken when the project's performance deviates significantly from the plan. It has two specific goals: one on monitoring actual performance, and another on managing corrective actions.

3.3.3 Analysis & Design and Implementation

The purpose of Technical Solution is to design, develop, and implement solutions to requirements. Solutions, designs, and implementations encompass products, product components, and product-related life-cycle processes either singly or in combinations as appropriate. Technical solution has three specific goals that address selecting product-component solutions, developing the design, and implementing the design. In the first goal—selecting product-component solutions alternative solutions are developed and analyzed, and the one that best satisfies the criteria is selected. The selected alternative may be used to develop more detailed requirements in the "Requirements Management" process area or designed in the second goal of Technical Solution. After the product components are designed, they are implemented together with support documentation in the third goal of Technical Solution.

3.3.4 Testing

The purpose of Testing is to demonstrate that a product or product component fulfills its intended use when placed in its intended environment. Testing has two specific goals that address preparing for validation and validating the product or product components. The validation practices are similar to those used in verification, but the two process areas focus on different topics. Validation addresses those activities needed to show that a product fulfills its intended use when it is placed in its intended environment, whereas verification shows that the work products meet their specified requirements.

3.3.5 Change Management

The purpose of Change Management is to establish and maintain the integrity of work products using configuration identification, configuration control, configuration status accounting, and configuration audits. Change Management has three specific goals that address establishing baselines, tracking and controlling changes, and establishing the integrity of baselines. It is assumed that it can occur at multiple levels of granularity and formality.

4. THE IMPLEMENTATION PROCESS

4.1 Process Management Framework

4.1.1 Implementation Solution

The solution is a software engineering process framework and implementation tool. It provides a disciplined approach to assigning tasks and responsibilities within a development organization. Its goal is to ensure the production of high-quality software that meets the needs of its end users within a predictable schedule and budget. [26].

The Solution has a JAVA based graphical user interface, which enables to visualize the process framework. The produced artifacts are stored in a repository. The repository consists of an Oracle database and a file server.

4.1.2 Process Structure

A process is a set of partially ordered steps intended to reach a goal. In software engineering, the goal is to build a software product or to enhance an existing one. In process engineering, the goal is to develop or enhance a process. In the implementation tool, these are organized into a set of disciplines that further define the workflows and other process elements.

The implementation tool is a CMMI dependend process framework for object-oriented software engineering. It describes a family of related software engineering processes that share a common structure and a common process architecture. The Implementation tool provides a disciplined approach to assigning tasks and responsibilities within a development organization.

The Process Management category of CMMI model is directly related with the process structure of the implementation tool. Each of the Process Management

process areas is strongly dependent on the ability to develop and deploy process and supporting assets. Here the implementation tool can play an important role. This can function as an additional organizational support asset and can help quantitative project management and statistical management of critical sub-processes for both projects and organization level. The organization analyses the process performance data collected from the defined processes to develop a quantitative understanding of product quality, service quality, and process performance of the organization's set of standard.The implementation tools "Process Structure" is explained below:

Disciplines:

A discipline is a collection of related activities that are related to a major area within the overall project.

WorkFlows:

A collection of all roles, activities and artifacts constitute a process, but it is not easy to understand in practical environment. To explain the structure in a better way, the implementation tool describes meaningful sequences of activities that produce some valuable result, and to show interactions between roles. A workflow is a sequence of activities that produces a result of observable value.

Workflow Details:

The implementation tool's process structure also includes workflow detail diagrams, which show groupings of activities that often are performed together. These diagrams show roles involved, input and output artifacts, and activities performed. The workflow detail diagrams are there for the following reasons:

The activities of a workflow are neither performed in sequence, nor done all at once. The workflow detail diagram shows how you often will work in workshops or team meetings while performing a workflow. People typically work in parallel on more than one activity, and look at more than one artifact while doing that. There are several workflow detail diagrams for a discipline.

It becomes too complex to show input and output artifacts for all activities of a discipline in one diagram. The workflow detail diagram allows to show activities and artifacts together, for one part of a workflow at a time.

The disciplines are not completely independent of one another. The workflow detail diagram can show a group of activities and artifacts in the discipline, together with closely related activities in another discipline.

Activities:

Roles have activities that define the work they perform. An activity is something that a role does that provides a meaningful result in the context of the project.

An activity is a unit of work that an individual playing the described role may be asked to perform. The activity has a clear purpose, usually expressed in terms of creating or updating some artifacts, such as a model, a class, or a plan. Every activity is assigned to a specific role. The granularity of an activity is generally a few hours to a few days, it usually involves one role, and affects one or only a small number of artifacts.

Activities may be repeated several times on the same artifact, especially when going from one iteration to another, refining and expanding the system, by the same role, but not necessarily the same individual[27].

Artifacts:

Activities have input and output artifacts. An artifact is a work product of the process. Roles use artifacts to perform activities, and produce artifacts in the course of performing activities. Artifacts are the responsibility of a single role, making responsibility easy to identify and understand, and promoting the idea that every piece of information produced in the process requires the appropriate set of skills. Even though one role may own the artifact, other roles will use the artifact, perhaps even updating it if the role has been given permission to do so. Artifacts don't have to be documents. Many processes have an excessive focus on documents, and in particular on paper documentation [28]. The most efficient approach to managing project artifacts is to maintain the artifacts within the appropriate tool used to create

and manage them. When necessary, you may generate documents from these tools, on a just-in-time basis. You should also consider delivering artifacts to the interested parties inside and together with the tool, rather than on paper. This approach ensures that the information is always up-to-date and based on actual project work, and it should not require any additional effort to produce .

4.1.3 Discipline Details

4.1.3.1 Requirements Management

Introduction:

In Figure 4.1 “Requirements Management” discipline purpose is explained.

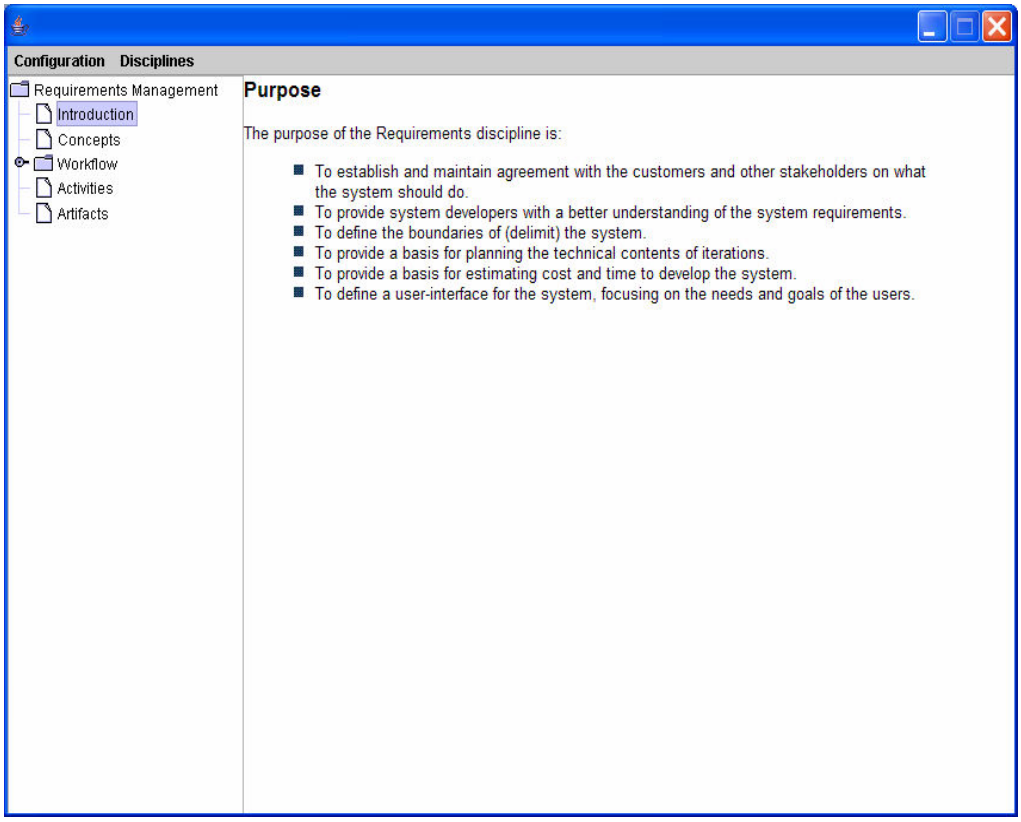

Figure 4.1: Requirement Management – Introduction

The purpose of the “Requirements Management” discipline is:

- To establish and maintain agreement with the customers and other stakeholders on what the system should do.
- To provide system developers with a better understanding of the system requirements.
- To define the boundaries of the system.
- To provide a basis for planning the technical contents of iterations.
- To provide a basis for estimating cost and time to develop the system.
- To define a user-interface for the system, focusing on the needs and goals of the users.
- To achieve these goals, it is important, first of all, to understand the definition and scope of the problem which is trying to solve with this system. The Business Rules, Business Use-Case Model and Business Analysis Model developed during Business Modeling will serve as valuable input to this effort. Stakeholders are identified and Stakeholder Requests are elicited, gathered and analyzed.

Concepts:

The main concepts are listed in this menuitem as keywords. In Figure 4.2 “Requirements Management” disciplines concepts are explained.

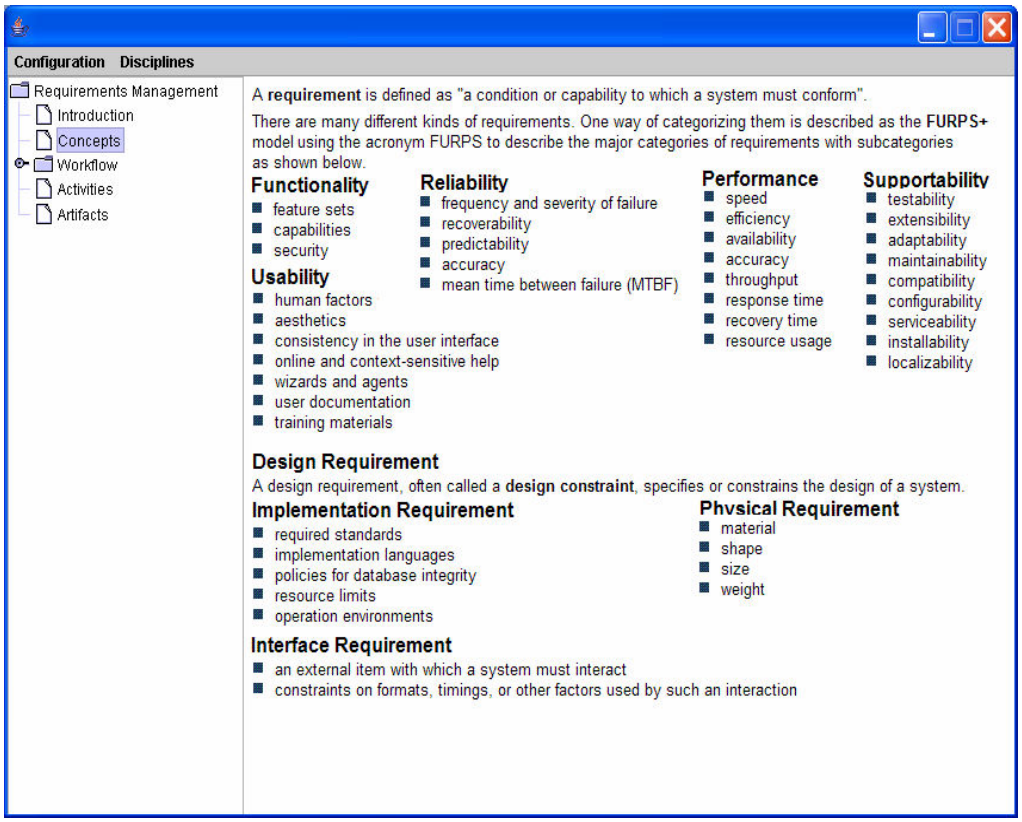

Figure 4.2: Requirement Management – Concepts

Workflow:

The main flowchart of this discipline is shown on this menu item. There are two ways to see the details. Project members could work on the flowchart by clicking on the activities listed on the screen or selecting from the submenu. In Figure 4.3, workflow details are shown.

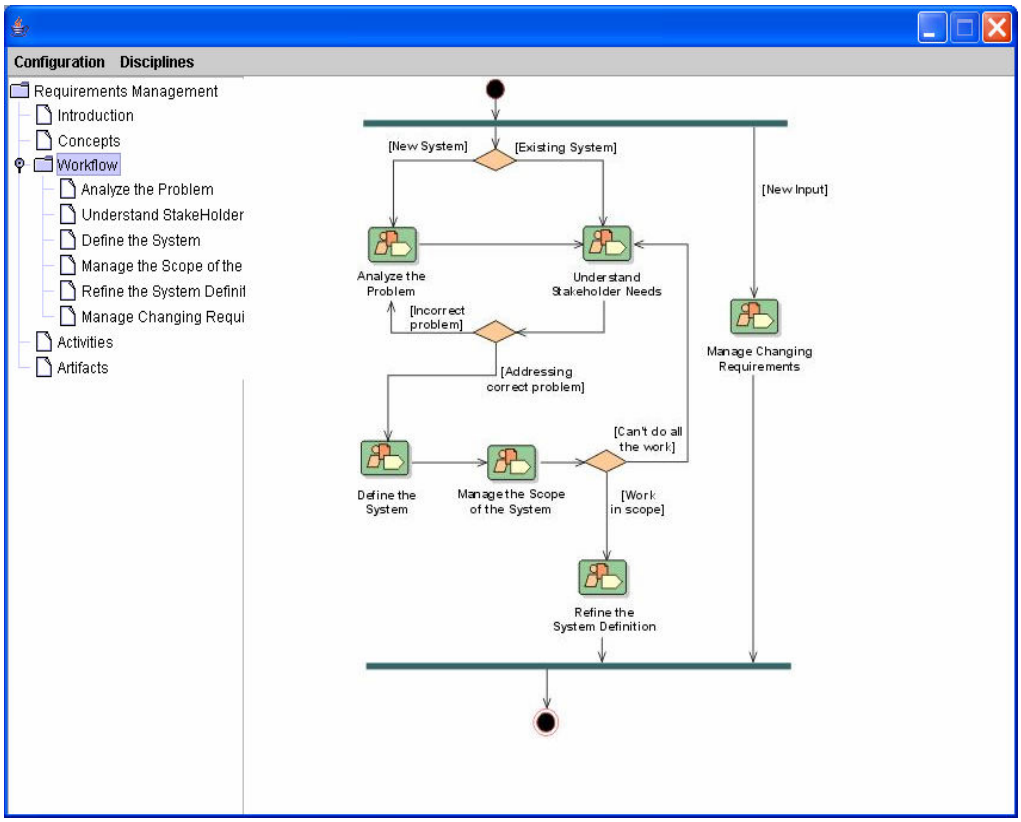

Figure 4.3: Requirement Management – Workflow

On Figure 4.4 “Analyze the Problem” workflow details are shown.

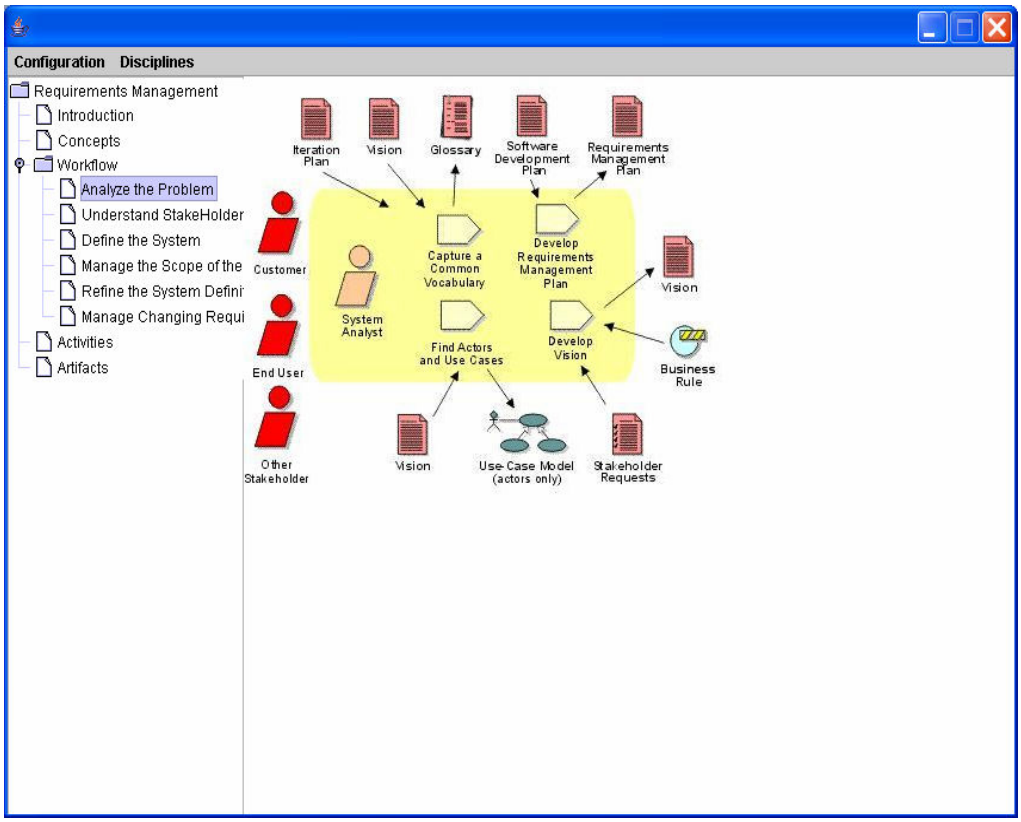

Figure 4.4: Requirement Management - Workflow – Analyze the Problem

The first step in any problem analysis is to make sure that all parties involved agree on what the problem is that needs to be solved-or opportunity that will be realized-by the system. In order to help avoid misunderstandings, it is important to agree on common terminology which will be used throughout the project. Starting early in the lifecycle, project terms should be defined in a glossary which will be maintained throughout the life of the project[31].

In order to fully understand the problems that need to be addressed, it is very important to know who the stakeholders are in the conceptual vision for the project. It should be noted that some of these stakeholders-the users of the system-will be represented by actors in your use-case model.

The requirements management plan is used to provide guidance on the requirements artifacts that you should develop, the types of requirements that should be managed for the project, the requirement attributes that should be collected and the approach to requirements traceability that will be used in managing the product requirements.

The primary artifact in which you capture the information gained from your problem analysis is the vision, which identifies the high-level user or customer view of the system to be built. In the vision, initial high-level requirements identify the key features it is desired that the appropriate solution will provide. These are typically expressed as a set of high-level features the system might possess in order to solve the most critical problems.

Key stakeholders should be involved in gathering the set of features to be considered, which might be gathered in a requirements workshop. The features can then be assigned attributes such as rationale, relative value or priority, source of request and so on, so that dependencies and work plans can begin to be managed.

To determine the initial scope for your project, the boundaries of the system must be agreed upon. The system analyst identifies users and systems - represented by actors - which will interact with the system.

In Figure 4.5, "Understand Stakeholder Needs" workflow details are shown.

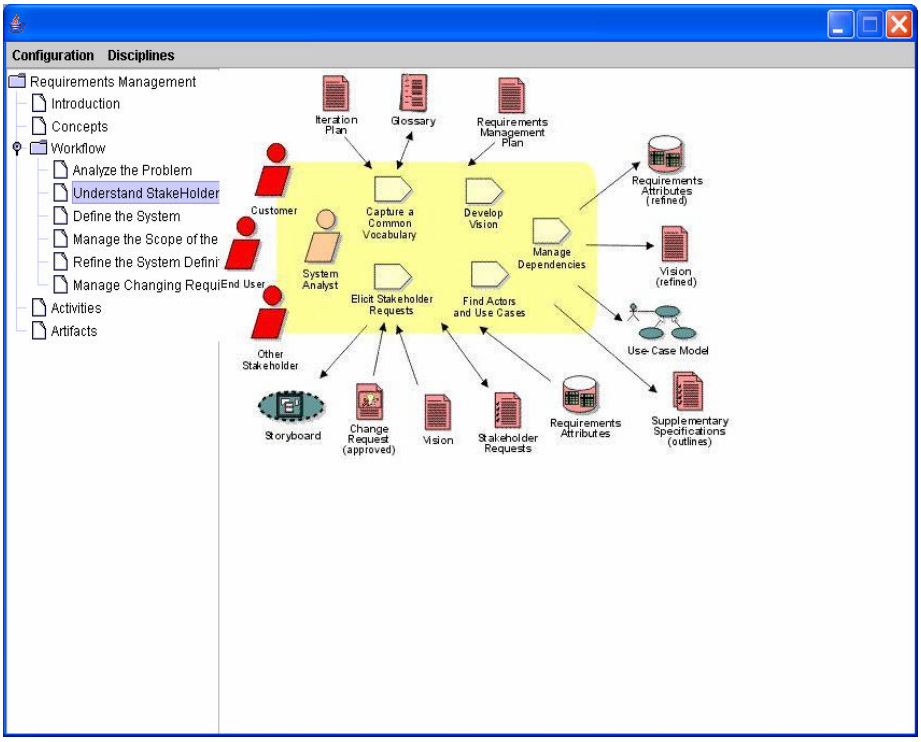

Figure 4.5: Requirement Management – Workflow – Understand Stakeholder Needs

This workflow detail addresses collecting and eliciting information from the stakeholders in the project in order to understand what their needs really are. The collected stakeholder requests can be regarded as a "wish list" that will be used as primary input to defining the high-level features of your system, as described in the vision, which drive the specification of the software requirements, as described in the use-case model, use cases and supplementary specifications.

This activity is performed during iterations in the inception and elaboration phases, however additional stakeholder requests will continue to be gathered throughout the project via change requests submitted and approved in accordance with your projects "Change Management" process.

The main objective is to elicit stakeholder requests using such input as interviews business rules, enhancement requests, and requirements workshops. The primary outputs are collections of prioritized features and their critical attributes, which will be used in defining the system and managing the scope of the system.

This information results in a refinement of the vision artifact, as well as a better understanding of the requirements attributes. Also, during the enactment of this workflow detail you may start discussing the functional requirements of the system in terms of its use cases and actors. Those non-functional requirements, which do not fit appropriately within the use-case specifications, should be documented in supplementary specifications.

Another important output is an updated glossary of terms to facilitate communication through the use of a common vocabulary among team members.

In Figure 4.6, "Define the System" workflow details are shown.

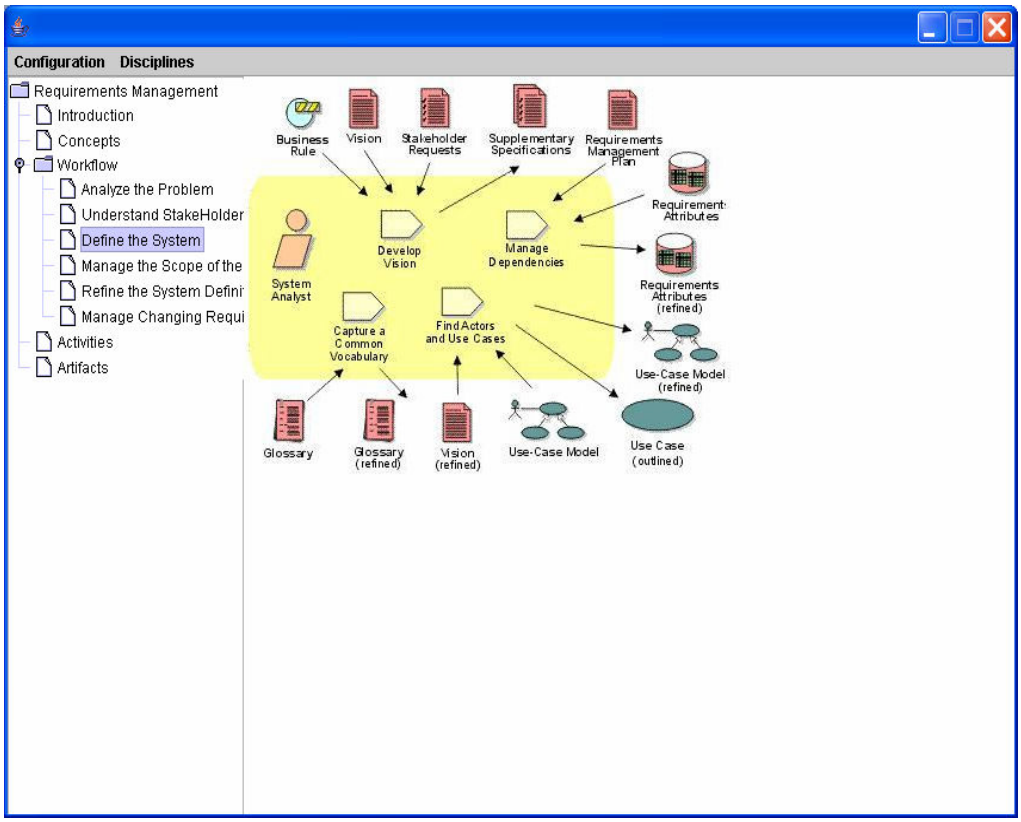

Figure 4.6: Requirement Management – Workflow – Define the System

The workflow detail addresses:

- Aligning the project team in their understanding of the system.
- Performing a high-level analysis on the results of collected stakeholder requests.
- Refining the vision to capture the key features that characterize the system.
- Refining the use-case model to include outlined use cases.
- Beginning to capture the results of the requirements elicitation activities in a more structured manner.

The activities that focus on problem analysis and understanding stakeholder needs create early iterations of key system definitions including the features defined in the vision and a first outline of the detailed requirements. In defining the system the

focus on identifying actors and use cases are more completely, and the global non-functional requirements are expanded as defined in the supplementary specifications.

Typically, this is primarily performed in iterations during the inception and elaboration phases, however it may be revisited as needed when managing scope and responding to changing requirements, as well as other changes in the project conditions.

In Figure 4.7, "Manage the Scope of the System" workflow details are shown.

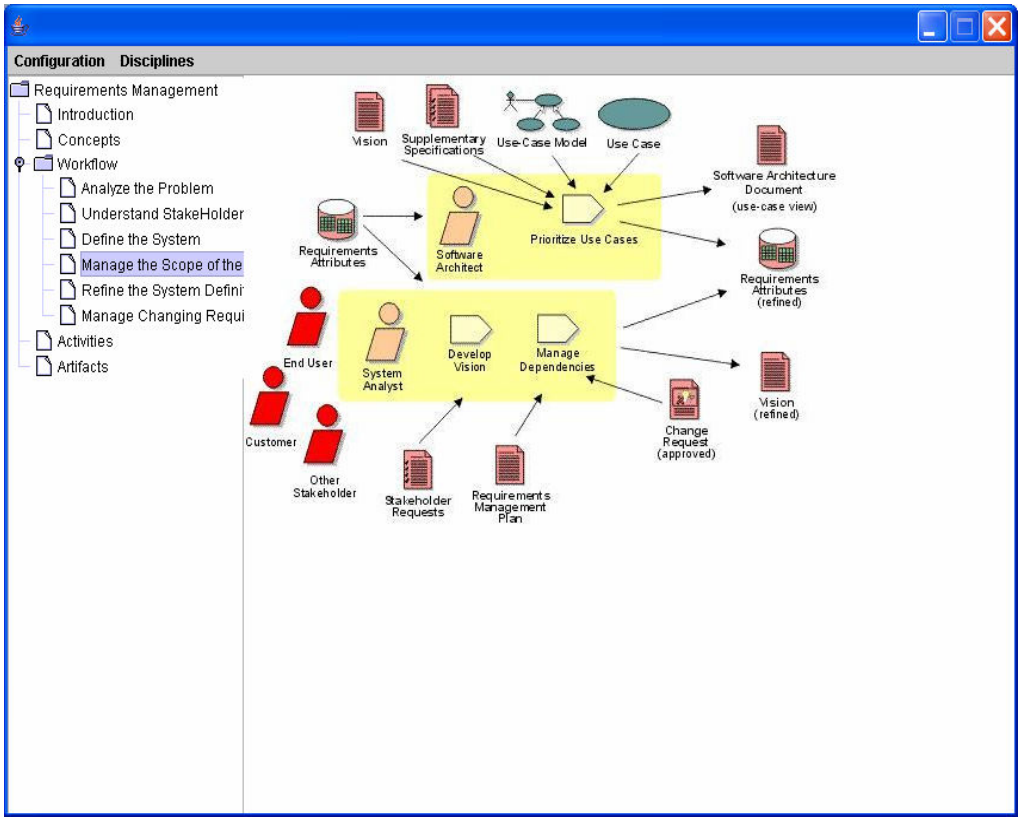

Figure 4.7: Requirement Management – Workflow – Manage the Scope of the System

This workflow detail addresses:

- Prioritizing and refining the input to the selection of features and requirements that are to be included in the current iteration.

- Defining the set of behavioral scenarios, for one or more use cases, that represent some significant central functionality.

- Defining how traceability will be maintained, including which requirement attributes and traceability relationships to maintain.

The scope of a project is defined by the set of requirements allocated to it. Managing project scope to fit the available resources (time, people, and money) is key to managing successful projects. Managing scope is a continuous activity that requires iterative or incremental development, which breaks project scope into smaller more manageable pieces.

Use requirement attributes, such as priority, effort, and risk, as the basis for negotiating the inclusion of a requirement is a particularly useful technique for managing scope. Focusing on the attributes rather than the requirements themselves helps desensitize negotiations that are otherwise contentious.

It is also helpful for team leaders to be trained in negotiation skills and for the project to have a champion in the organization, as well as on the customer side. Product/project champions should have the organizational power to refuse scope changes beyond the available resources or to expand resources to accommodate additional scope.

Project scope should be managed continuously throughout the project. A better understanding of system functionality can be formulated at the point that most actors and use cases have been identified and outlined. Non-functional requirements, which either do not fit in the use-case model or are general across multiple use cases, should be documented in the supplementary specifications. The system analyst role is responsible for determining values of priority, effort, cost, risk values etc., from the appropriate stakeholders, which are collected in the repository of requirements attributes. These will be used by staff in the project manager role when planning each iteration and will enable staff in the software architect role to identify the architecturally significant scenario's or complete use cases, which will help define the use-case view of the architecture.

In Figure 4.8, “Refine the System Definition” workflow details are shown.

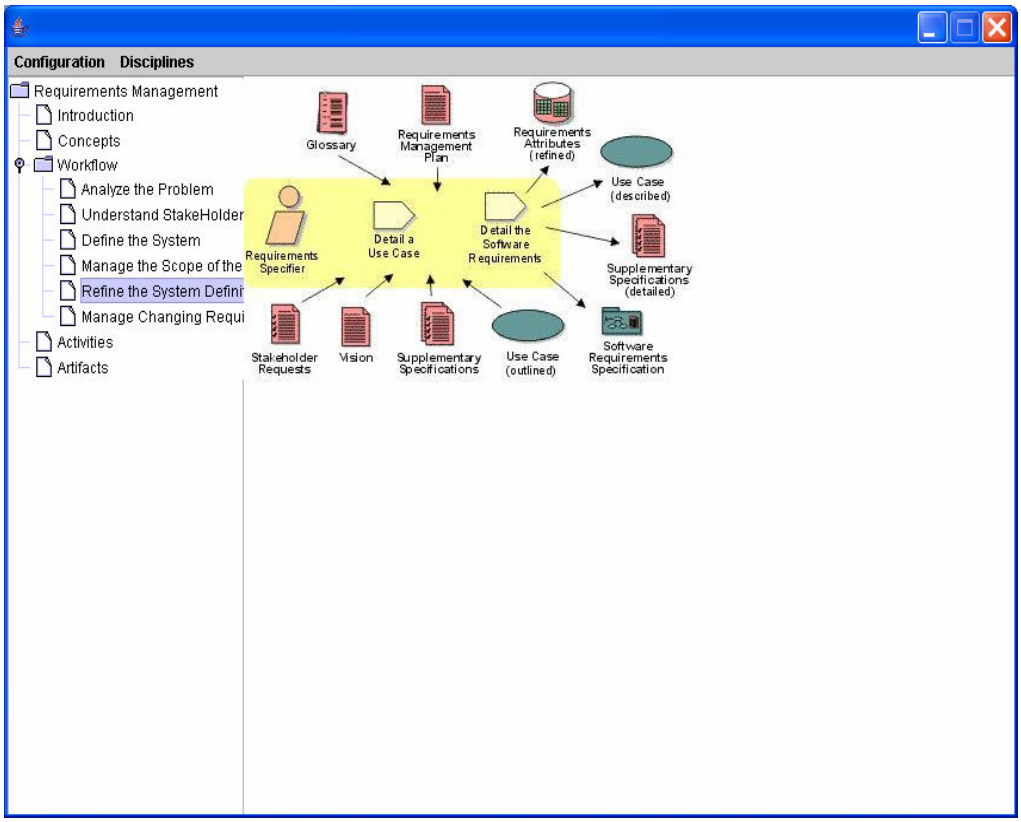

Figure 4.8: Requirement Management – Workflow – Refine the System Definition

The workflow detail addresses:

- Describing the use case flow of events in detail.
- Detailing Supplementary Specifications.
- Developing a Software Requirements Specification, if more detail is needed,

This workflow detail furthers the understanding of project scope reflected in the set of prioritized product features that it is believed can be achieved by fairly firm budgets and dates. The output is a more in-depth understanding of system functionality expressed in refined, detailed requirements in specification artifacts and outlined behavioral prototypes. The specification artifacts can take the form of detailed use cases and Supplementary Specifications and in some cases a formal Software Requirements Specification may be developed. This work typically starts by reviewing the existing actor definitions and if necessary least briefly describing

the actors, then continues with detailing the use cases that have been previously outlined for each actor.

Whenever the requirements specifications are changed, regular reviews and updates to the associated requirements attributes should be done as shown in the Manage Changing Requirements workflow detail.

In Figure 4.9, "Manage Changing Requests" workflow details are shown.

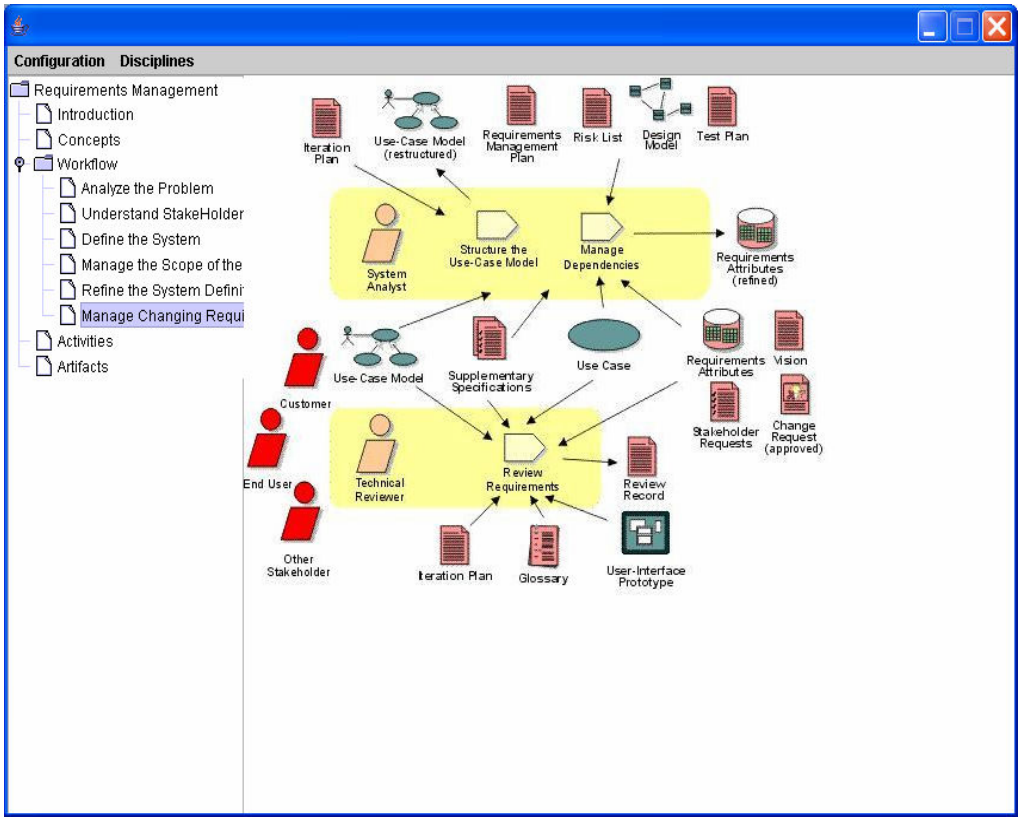

Figure 4.9: Requirement Management – Workflow – Manage Changing Requests

This workflow detail addresses:

- Evaluating requested changes and determining their impact on the existing requirement set.
- Structuring the use-case model.
- Setting up appropriate requirements attributes and traceability relationships.

- Verify that the results of the requirements work conform to the customer's view of the system.

Changes to requirements naturally impact downstream artifacts. For example the models produced in the course of analysis & design work, the tests developed to validate that the requirements have been met, and the end-user support materials. The traceability relationships identified in the manage dependency activity of this discipline, identify the relationships between requirements and other artifacts. These relationships are the key to understanding the impact of requirements change.

Another important consideration is the tracking of requirement history. By capturing the nature and rationale of requirements changes, reviewers (in this case the role is played by anyone on the software project team whose work is affected by the change) receive the information needed to respond to the change properly.

Regular reviews, along with updates to the requirement attributes and dependencies, should be done whenever the requirements are updated.

Activities:

In figure xx, acitivites of the “Requirements Management” discipline are shown. The details are explained in the workflow details meuitem. This view is added to the menuitems to list the roles’s responsibilities in a clear way.

In Figure 4.10, workflow activities are shown.

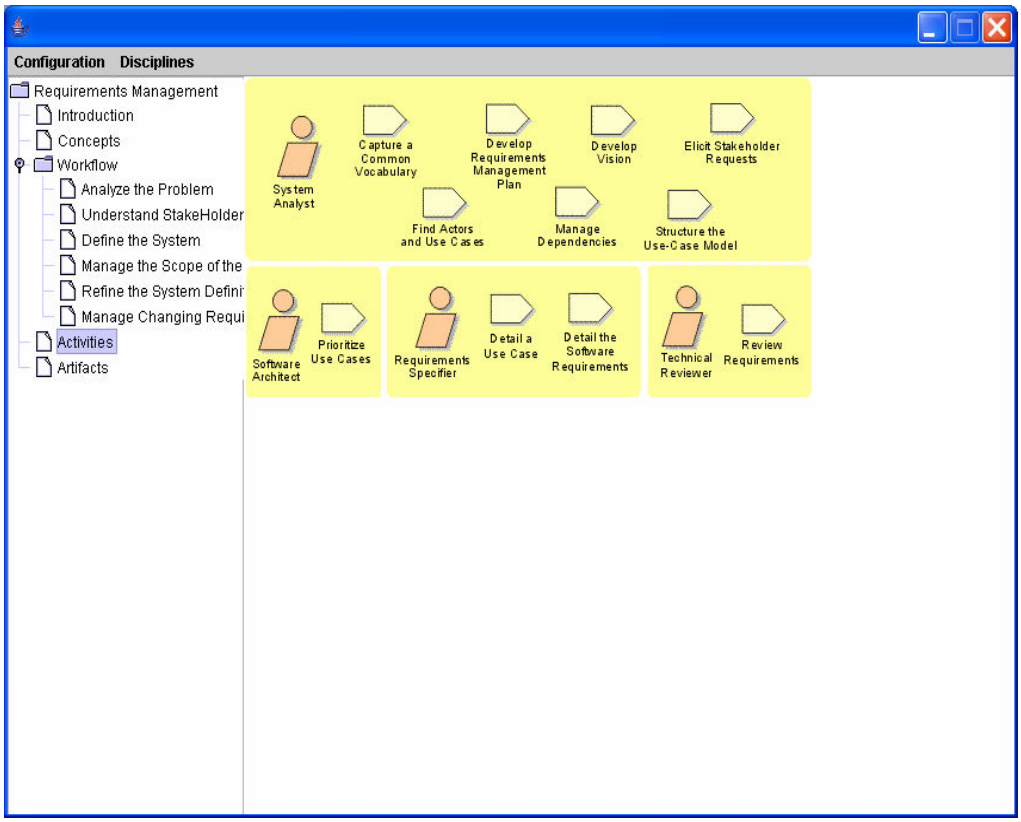

Figure 4.10: Requirement Management– Activities

Artifacts:

In Figure 4.11, artifacts of the "Requirements Management" discipline are shown. The workflow details are explained in the workflow details meuitem. This view is added to the menuitems to list the produced artifacts.

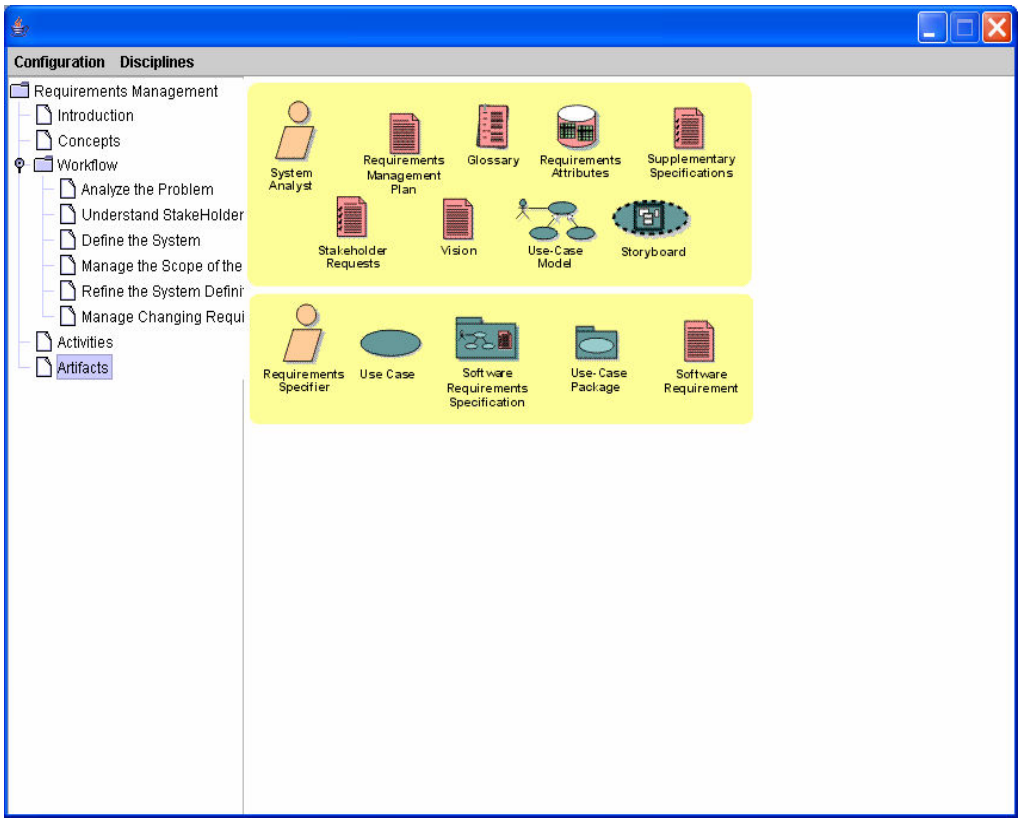

Figure 4.11: Requirement Management– Artifacts

A vision document, a use-case model, use cases and Supplementary Specification are developed to fully describe the system - what the system will do - in an effort that views all stakeholders, including customers and potential users, as important sources of information (in addition to system requirements).

Stakeholder requests are both actively elicited and gathered from existing sources to get a "wish list" of what different stakeholders of the project (customers, users, product champions) expect or desire the system to include, together with information on how each request has been considered by the project [32].

The vision document provides a complete vision for the software system under development and supports the contract between the funding authority and the development organization. Every project needs a source for capturing the expectations among stakeholders. The vision document is written from the customers' perspective, focusing on the essential features of the system and acceptable levels of quality. The vision should include a description of what features

will be included as well as those considered but not included. It should also specify operational capacities (volumes, response times, accuracies), user profiles (who will be using the system), and inter-operational interfaces with entities outside the system boundary, where applicable. The vision document provides the contractual basis for the requirements visible to the stakeholders.

The use-case model should serve as a communication medium and can serve as a contract between the customer, the users, and the system developers on the functionality of the system, which allows:

- Customers and users to validate that the system will become what they expected.
- System developers to build what is expected.

The use-case model consists of use cases and actors. Each use case in the model is described in detail, showing step-by-step how the system interacts with the actors, and what the system does in the use case. Use cases function as a unifying thread throughout the software lifecycle; the same use-case model is used in system analysis, design, implementation, and testing.

The supplementary specifications are an important complement to the use-case model, because together they capture all software requirements (functional and nonfunctional) that need to be described to serve as a complete software requirements specification.

A complete definition of the software requirements described in the use cases and supplementary specifications may be packaged together to define a software requirements specification (SRS) for a particular "feature" or other subsystem grouping.

A requirements management plan specifies the information and control mechanisms which will be collected and used for measuring, reporting, and controlling changes to the product requirements.

Complementary to the above mentioned artifacts, the following artifacts are also developed:

- Glossary
- Storyboard

The glossary is important because it defines a common terminology which is used consistently across the project or organization.

The storyboards may be generated during requirements elicitation, which are done in parallel with other requirements activities. They provide important feedback mechanisms in later iterations for discovering unknown or unclear requirements.

4.1.3.2 Analysis & Design:

Introduction:

In Figure 4.12 "Analysis & Design" disciplines purpose is explained.

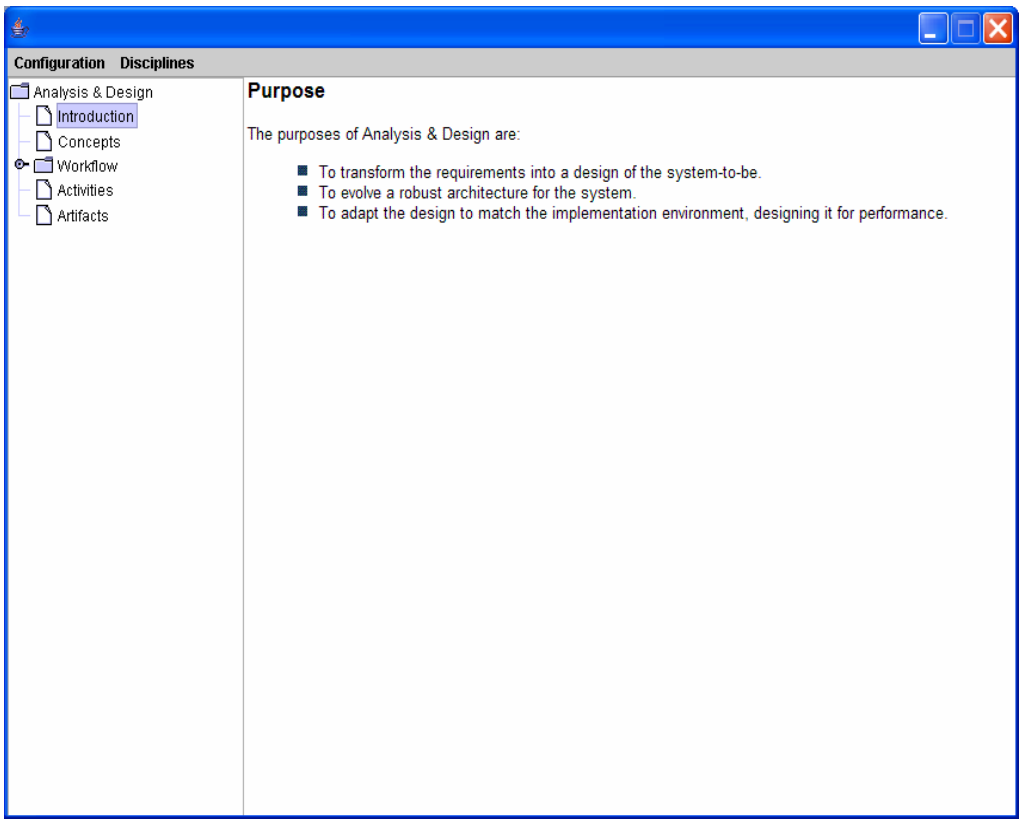

Figure 4.12: Analysis & Design – Introduction

The purposes of Analysis & Design are:

- To transform the requirements into a design of the system-to-be.
- To evolve a robust architecture for the system.
- To adapt the design to match the implementation environment, designing it for performance.

Concepts:

The main concepts are listed in this menuitem as keywords. In Figure 4.13 “Analysis & Design” disciplines purpose is explained.

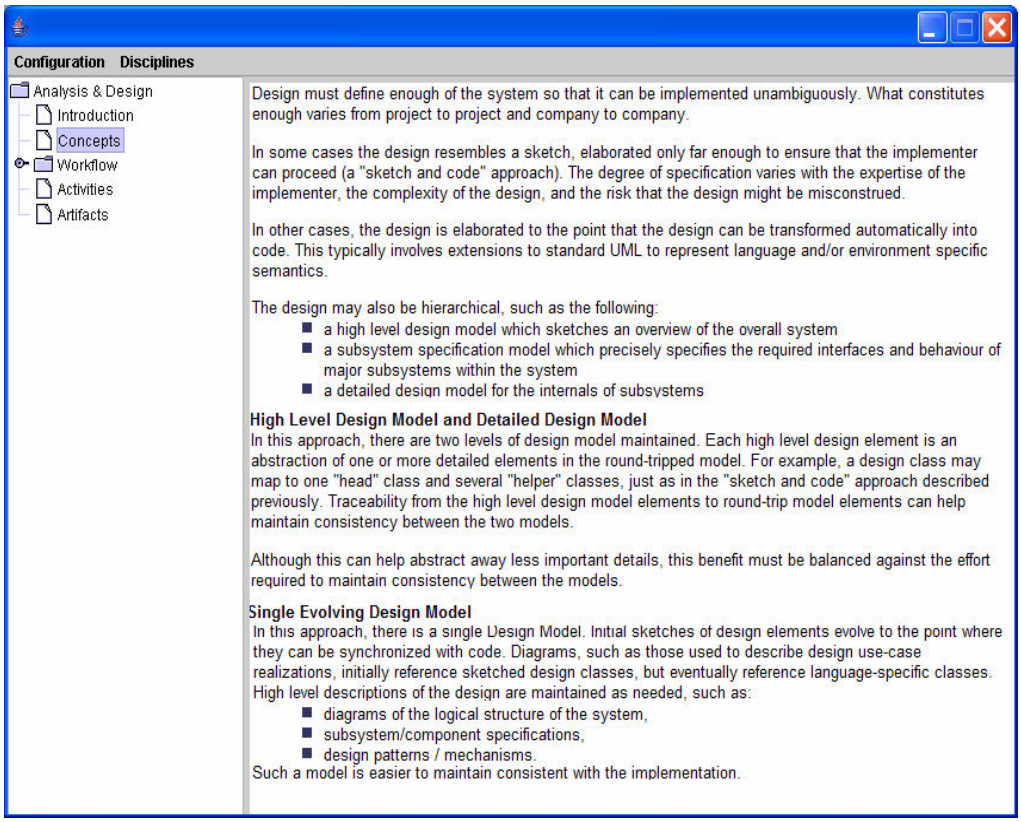

Figure 4.13: Analysis & Design – Concepts

Workflow:

The main flowchart of this discipline is shown on this menuitem. There are two ways to see the details. Project members could work on the flowchart by clicking on the activities listed on the screen or selecting from the submenu.

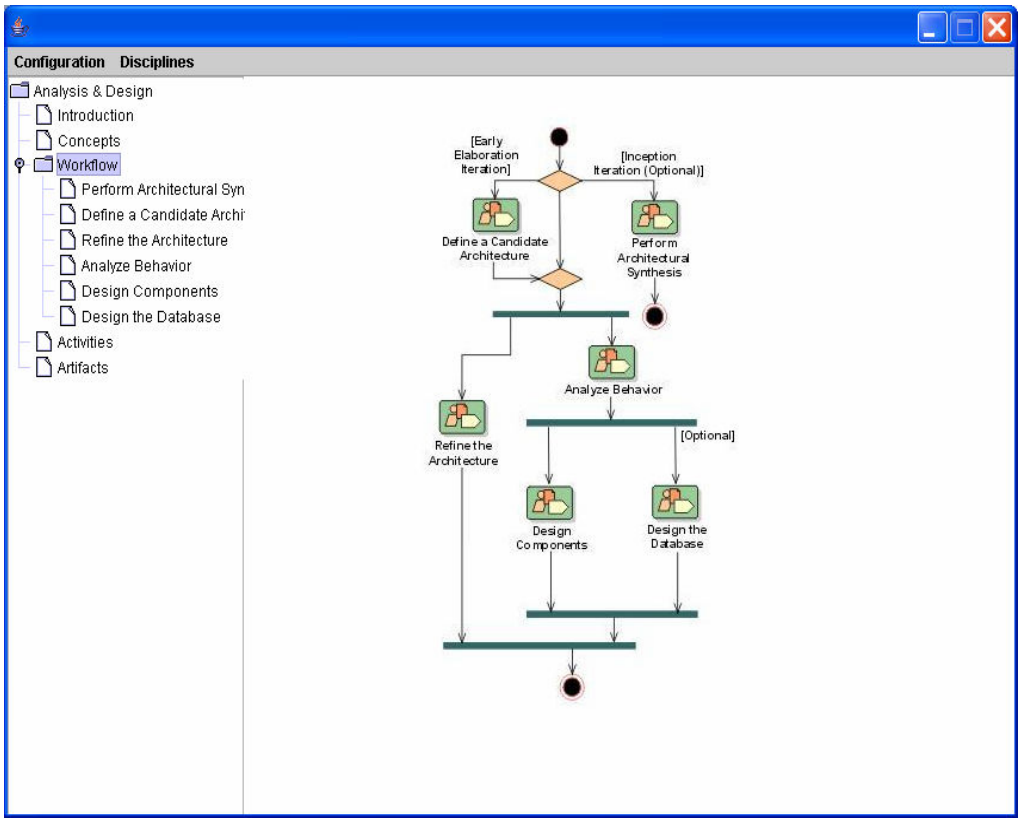

Figure 4.14: Analysis & Design – Workflow

In figure 4.15, "Perform Architectural Synthesis" workflow details are shown.

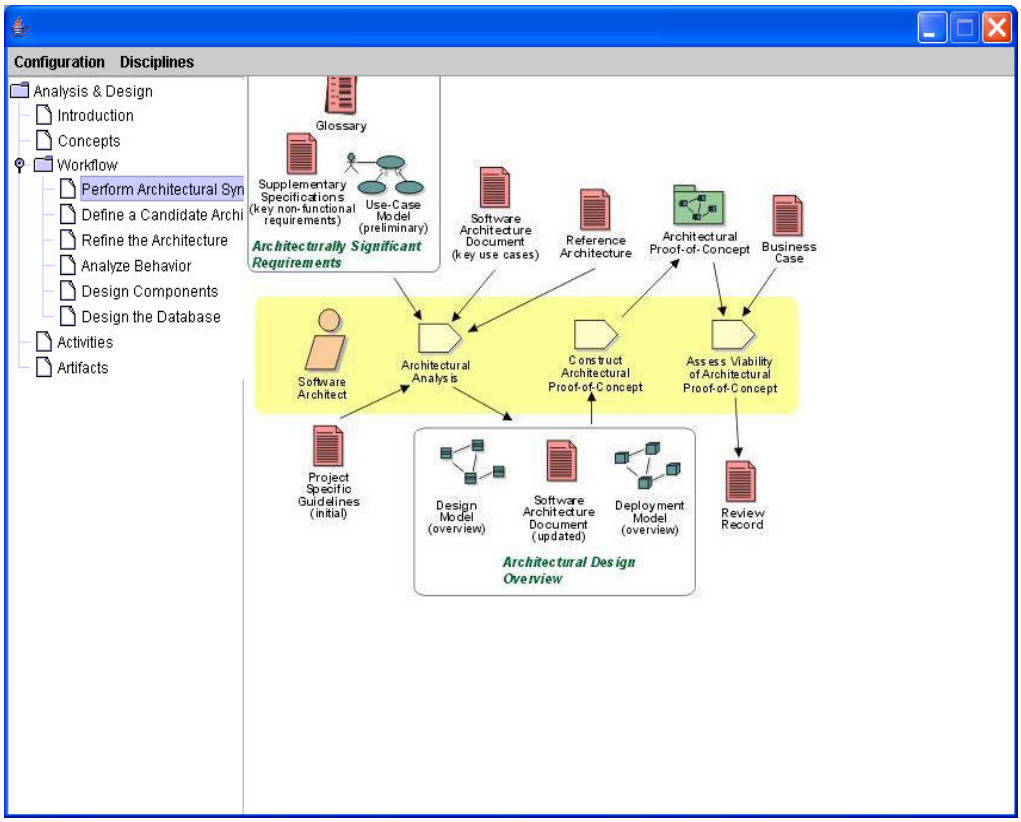

Figure 4.15: Analysis & Design – Workflow - Perform Architectural Synthesis

This workflow detail is about showing that there exists, or is likely to exist, a solution which will satisfy the architecturally significant requirements, thus showing that the system, as envisioned, is feasible.

As with "Workflow Detail: Define a Candidate Architecture", shown in Figure 4.16, these activities are best carried out by a small team staffed by cross-functional team members. Issues that are typically architecturally significant include performance, scaling, process and thread synchronization, and distribution. The team should also include members with domain experience who can identify key abstractions. The team should also have experience with model organization and layering. From these inputs, the team will need to be able to synthesize a model, or even a prototype, of a solution.

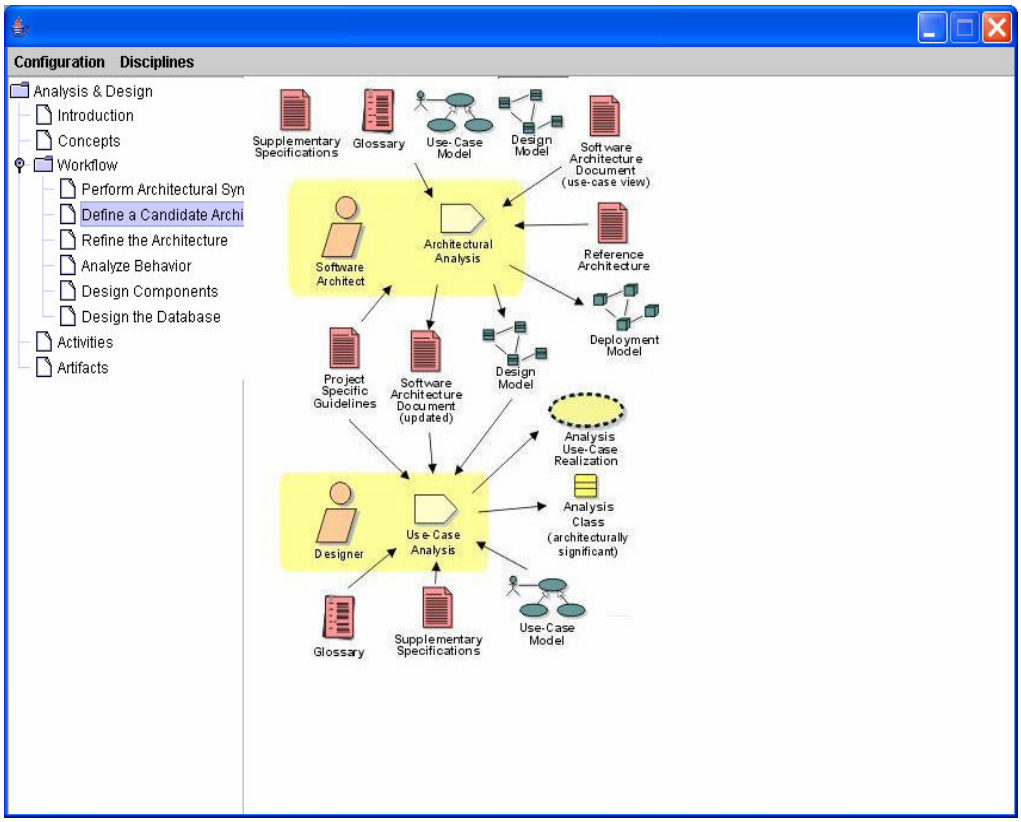

Figure 4.16: Analysis & Design – Workflow - Define a Candidate Architecture

This workflow detail has the following goals:

- Create an initial sketch of the architecture of the system
    - Define an initial set of architecturally significant elements to be used as the basis for analysis
    - Define an initial set of analysis mechanisms
    - Define the initial layering and organization of the system
    - Define the use-case realizations to be addressed in the current iteration
- Identify analysis classes from the architecturally significant use cases
- Update the use-case realizations with analysis class interactions

The work is best done in several sessions, perhaps performed over a time, with iteration between Architectural Analysis and Use-Case Analysis. Perform an initial pass at the architecture in Architectural Analysis, then choose architecturally significant use cases, performing Use-Case Analysis on each one. After each use case is analyzed, update the architecture as needed to reflect adaptations required to accommodate new behavior of the system and to address potential architectural problems which are identified.

Where the architecture already exists, change requests may need to be created to change the architecture to account for the new behavior the system must support. These changes may be to any artifact in the process, depending on the scope of the change.

In Figure 4.17, "Refine Architecture" workflow details are shown.

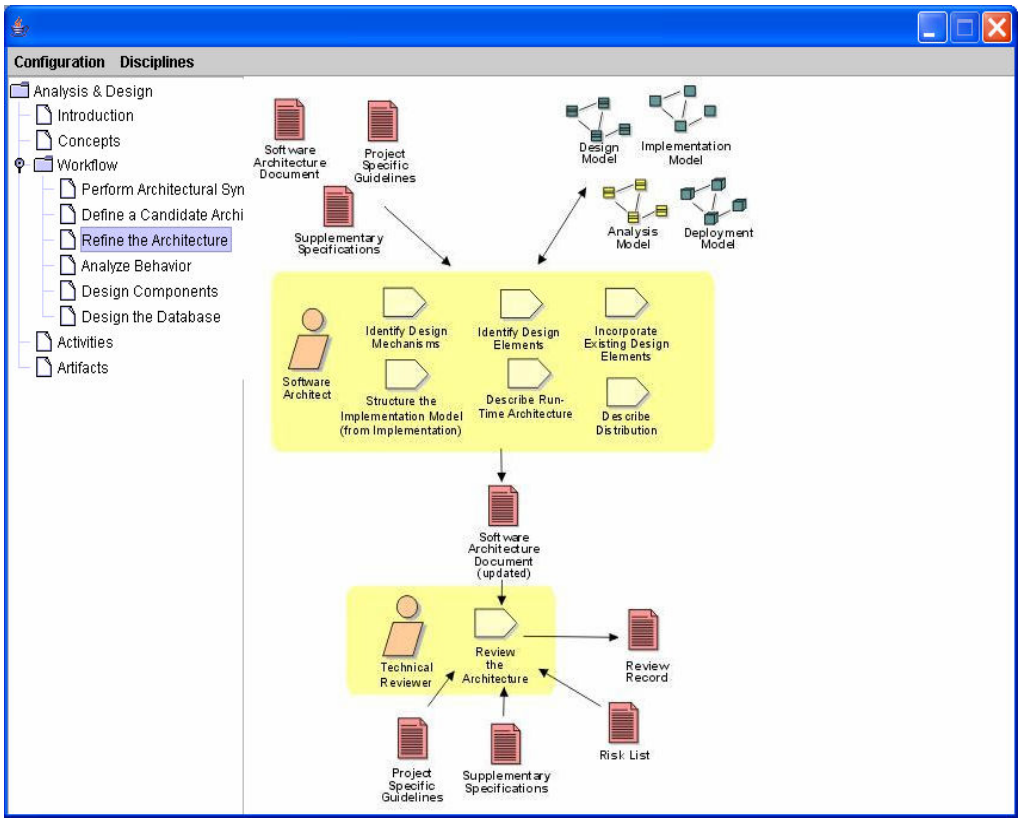

Figure 4.17: Analysis & Design – Workflow – Refine Architecture

This Workflow Detail:

- Provides the natural transition from analysis activities to design activities, identifying:
    - appropriate design elements from analysis elements
    - appropriate design mechanisms from related analysis mechanisms
- Describes the organization of the system's run-time and deployment architecture
- Organizes the implementation model so as to make the transition between design and implementation seamless
- Maintains the consistency and integrity of the architecture, ensuring that:
    - new design elements identified for the current iteration are integrated with pre-existing design elements.
    - maximal re-use of available components and design elements is achieved as early as possible in the design effort.

The work is best done in several sessions. The initial focus will be on the activities "Activity: Identify Design Mechanisms" and "Activity: Identify Design Elements", with a great deal of iteration with the "Activity: Incorporate Existing Design Elements" activity to make sure that new elements do not duplicate functionality of existing elements.

As the design emerges, concurrency and distribution issues are introduced in the activities "Activity: Describe the Run-time Architecture" and "Activity: Describe Distribution", respectively. As these issues are considered, changes to design elements may be required to split behavior across processes, threads or nodes.

As the individual models are refined to incorporate the architectural decisions, the results are documented in respective view sections in the Software Architecture Document. The resulting architecture is reviewed [33].

These activities are best carried out by a small team staffed by cross-functional team members. Issues that are typically architecturally significant include usability,

performance, scaling, process and thread synchronization, and distribution. The team should also include members with domain experience who can identify key abstractions. The team should also have experience with model organization and layering. The team will need to be able to pull all these disparate threads into a cohesive, coherent architecture.

Because the focus of the architecture effort is shifting toward implementation issues, greater attention needs to be paid to specific technology issues. This will force the architecture team to shift members or expand to include people with distribution and deployment expertise. In order to understand the potential impact of the structure on the implementation model on the ease of integration, expertise in the software build management process is useful to have.

At the same time, it is essential that the architecture team not be composed of a large extended team. A strategy for countering this trend is to retain a relatively small core team with a satellite group of extended team members that are brought in as "consultants" on key issues**.** This structure also works well for smaller projects where specific expertise may be borrowed or contracted from other organizations; they can be brought in as specific issues need to be addressed.

In Figure 4.18, "Analyze Behavior" workflow details are shown.

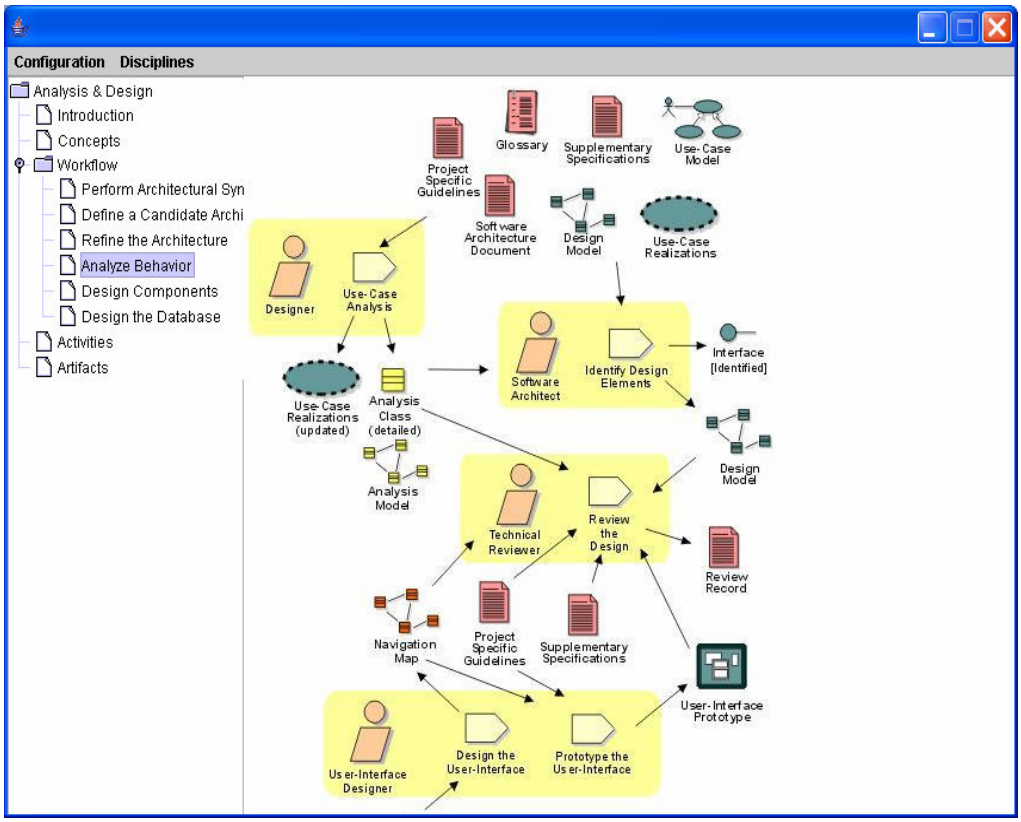

Figure 4.18: Analysis & Design – Workflow – Analyze Behavior

This workflow detail occurs in each iteration in which there are behavioural requirements to be analyzed and designed.

The analysis of behavioural requirements includes:

- identifying analysis classes that satisfy the required behaviour
- determining how these analysis classes fit into the logical architecture of the system. The analysis classes may be determined to belong to existing subsystems, require the creation of new subsystems, or cause existing subsystems and their interfaces to be redefined.

This Workflow Detail may also include modeling and prototyping of the user interface:

“Activity: Design the User-Interface” and “Activity: Prototype the User-Interface” are performed iteratively throughout the elaboration iterations. Early iterations focus

on the initial user interface design, which includes the identification and design of the key user interface elements and the navigation paths between them. “Storyboarding” is an effective technique that can be used during user-interface design to gain a better understanding of how the user interface should behave. Once consensus on the initial user-interface design has been reached, then the development of an executable user-interface prototype begins. Feedback on the prototype is fed back into the user-interface design. The initial prototype typically supports only a subset of the system's features. In subsequent iterations, the prototype is expanded, gradually adding broader coverage of the system's features. The main benefit of producing non-functional versions of the user-interface during user-interface design is to postpone the investment of more elaborate and costly functional user-interface prototypes until there is consensus on the overall user-interface design. It is important to work closely with users and potential users of the system when designing and prototyping the user-interface in order to confirm and validate the usability of the system.

A number of use-case analysis workshops may be organized in parallel, limited only by the available resource pool and the skills of the participants. As soon as possible following each use-case analysis workshop, some members of the workshop and some members of the architecture team should work to merge the results of the workshop in the “Activity: Identify Design Elements”. Members of both teams are essential: the use-case analysis team members understand the context in which the analysis classes were identified, while the architecture team understands the greater context of the design as well as other use cases which have already been identified.

As the design work matures and stabilizes, increasingly larger parts of it can and should be reviewed. Smaller, more focused reviews are better than large all-encompassing reviews; sixteen two-hour reviews focused on very specific aspects are significantly better than a single review spanning two days. In the focused reviews, define objectives to bound the focus of the review, and ensure that a small review team with the right skills for the review, given the objectives, is available for the review. Early reviews should focus primarily on the integrity of layering and packaging in the design, the stability and quality of the interfaces, and the completeness of coverage of the use case behavior. Later reviews should drill down

into packages and subsystems to ensure that their contents completely and correctly realize their defined interfaces, and that the dependencies and associations between design elements are necessary, sufficient and correct.

In Figure 4.19, "Design Components" workflow details are shown.

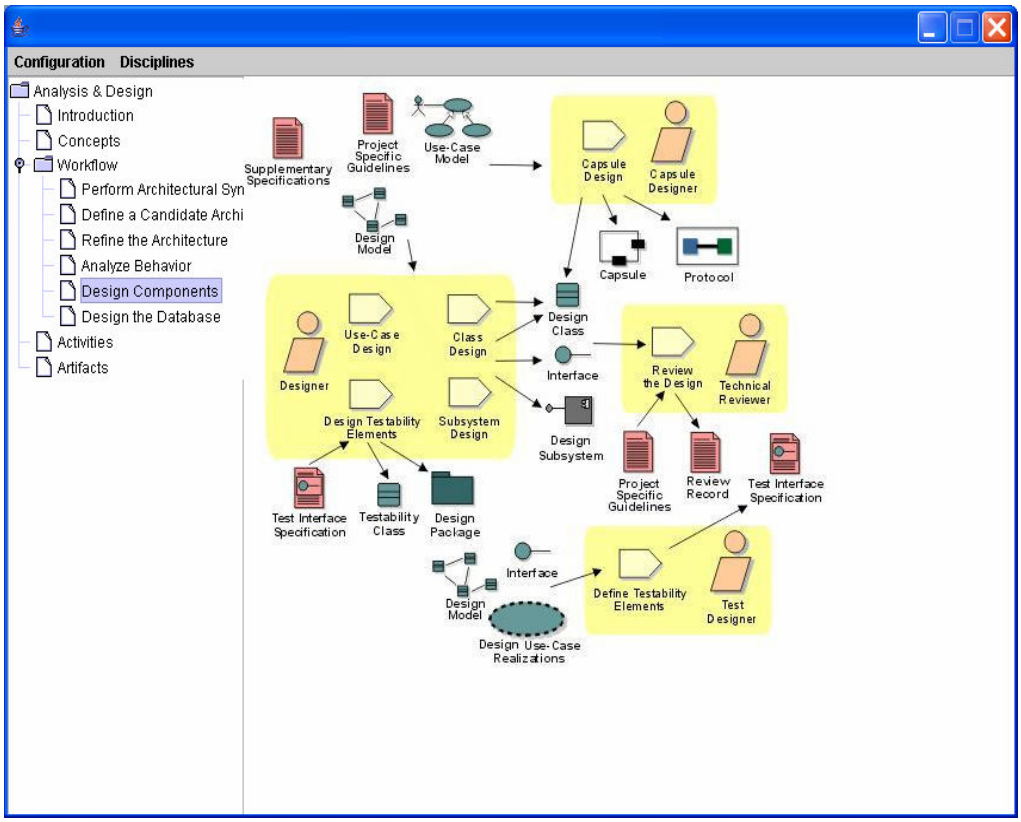

Figure 4.19: Analysis & Design – Workflow – Design Components

This Workflow Detail has the following goals:

- Refine the definitions of design elements (including capsules and protocols) by working out the 'details' of how the design elements realize the behavior required of them.
- Refine and update the use-case realizations based on new design element identified.
- Reviewing the design as it evolves

Typically the work here is carried in individually or in small teams, with informal inter-group interactions where needed to communicate changes between the teams. As design elements are refined, responsibilities often shift between them, requiring simultaneous changes to a number of design elements and use-case realizations. Because of the interplay of responsibilities, it is almost impossible for design team members to work in complete isolation. To keep the design effort focused on the required behavior of the system, a typical pattern of interaction emerges:

- design elements are refined by the responsible persons or teams
- a small group gathers informally to work out the impact of the new design elements on a set of existing use-case realizations
- in the course of the discussion, changes to both the use-case realization and the participating design elements are identified
- the cycle repeats until all required behaviour for the iteration is designed.

Because the process itself is iterative, the criteria for 'all required behaviour for the iteration' will vary depending on the position in the lifecycle:

- In the elaboration phase, the focus will be on architecturally-significant behaviors, with all other 'details' effectively ignored.
- In the construction phase there is a shift to completeness and consistency of the design, so that by the end of the construction phase there are no unresolved design issues.

Note that the design for an iteration does not need to be complete before beginning implementation and test activities. Partially implementing and testing a design as it evolves can be an effective means of validating and refining design, even within an iteration.

Typically, one person or a small team is responsible for a set of design elements, usually one or more packages or subsystems containing other design elements. This person/team is responsible for fleshing out the design details for the elements contained in the package or subsystem: completing all operation definitions and the

definition of relationships to other design elements. The “Activity: Capsule Design” focuses on the recursive decomposition of functionality in the system in terms of capsules and classes. The “Activity: Class Design” focuses on refining the design of passive class design elements, while the “Activity: Subsystem Design” focuses on the allocation of behavior mapped to the subsystem itself to contained design elements (either contained capsules and classes or subsystems). Typically subsystems are used primarily as large-grained model organization structures, while capsules being used for the bulk of the work and "ordinary" classes being relegated largely to passive stores of information.

The individuals or teams responsible for designing capsules should be knowledgeable in the implementation language as well as possessing expertise in the concurrency issues in general. Individuals responsible for designing passive classes should also be knowledgeable in the implementation language as well as in algorithms or technologies to be employed by the class. Individuals or teams responsible for subsystems should be more generalists, able to make decisions on the proper partitioning of functionality between design elements, and able to understand the inherent trade-offs involved in various design alternatives.

While the individual design elements are refined, the use-case realizations must be refined to reflect the evolving responsibilities of the design elements. Typically, one person or a small team is responsible for refining one or more related use-case realizations. As design elements are added or refined, the use-case realizations need to be reconsidered and evolved as they become outdated, or as improvements in the design model allow for simplifications in the use-case realizations. The individuals or teams responsible for use-case realizations need to have broader understanding of the behavior required by the use cases and of the trade-offs of different approaches to allocating this behavior amongst design elements. In addition, since they are responsible for selecting the elements that will perform the use cases, they need to have a deep understanding of external (public) behaviors of the design elements themselves.

In Figure 4.20, “Design the Database” workflow details are shown.

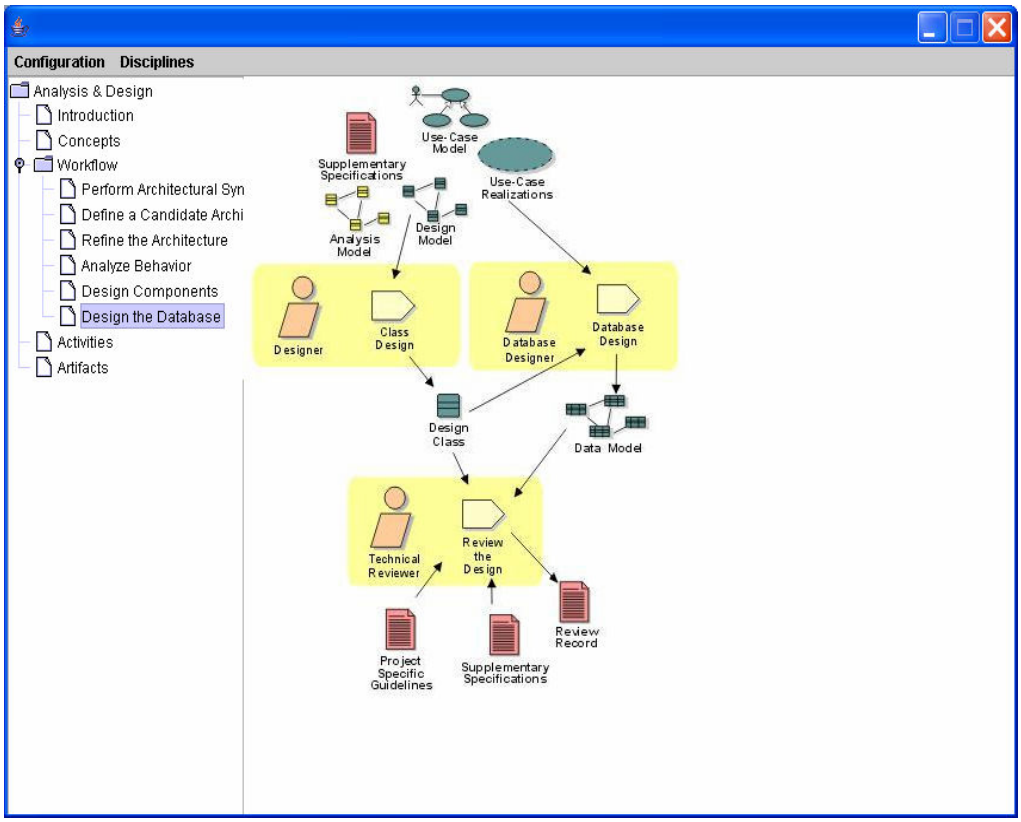

Figure 4.20: Analysis & Design – Workflow – Design the Database

This Workflow Detail includes:

- Identifying the persistent classes in the design
- Designing appropriate database structures to store the persistent classes
- Defining mechanisms and strategies for storing and retrieving persistent data in such a way that the performance criteria for the system are met

The database and persistent data storage and retrieval mechanisms, are implemented and tested as part of the overall implementation of the components and subsystems of the application.

In the elaboration phase, this workflow focuses on ensuring that the persistence strategy is scalable and that the database design and persistence mechanism will support the throughput requirements of the system. Persistent classes identified in "Activity: Class Design" are mapped to the persistence mechanism and data-

intensive use cases are analyzed to ensure the mechanisms will be scalable. The persistence mechanism and database design is assessed and validated.

Persistence must be treated as an integral part of the design effort, and close collaboration between designers and database designers is essential. Typically the database designer is a 'floating' resource, shared between several teams as a consulting resource to address persistence issues. The database designer is also typically responsible for the persistence mechanisms; if the persistence mechanism is built rather than bought, there will typically be a team of people working on this. Larger projects will typically require a small team of database designers who will need to coordinate work between both design teams and amongst themselves to ensure that persistence is consistently implemented across the project.

The Designers responsible for persistent classes need to have an understanding of the persistence in general and the persistence mechanisms in specific. Their primary responsibility is to ensure that persistent classes are identified and that these classes utilize the persistence mechanisms in an appropriate manner. The Database Designer needs to understand the persistent classes in the design model and so must have a working understanding of object-oriented design and implementation techniques. The Database Designer also needs a strong background in database concurrency and distribution issues.

Activities:

In Figure 4.21, acitivites of the “Analysis and Design” discipline are shown. The details are explained in the workflow details menu item. This view is added to the menu items to list the roles’s responsibilities in a clear way.

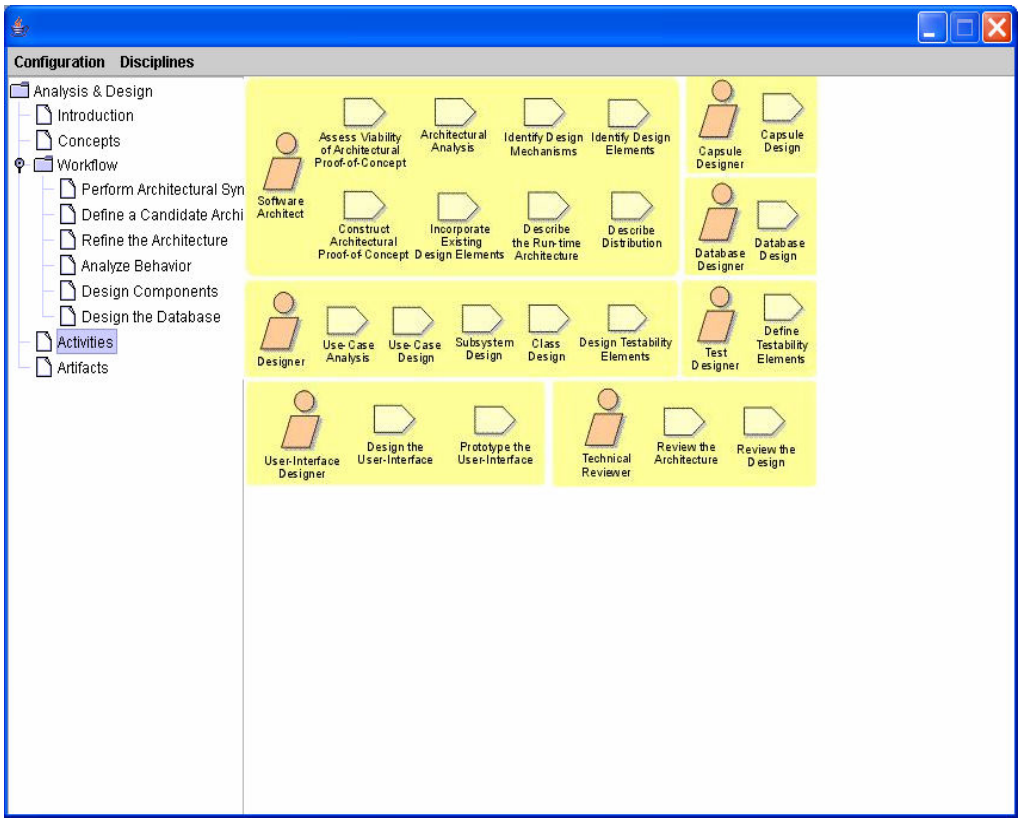

Figure 4.21: Analysis & Design – Activities

Artifacts:

In Figure 4.22, artifacts of the "Analysis and Design" discipline are shown. The workflow details are explained in the workflow details menu item. This view is added to the menu items to list the produced artifacts.

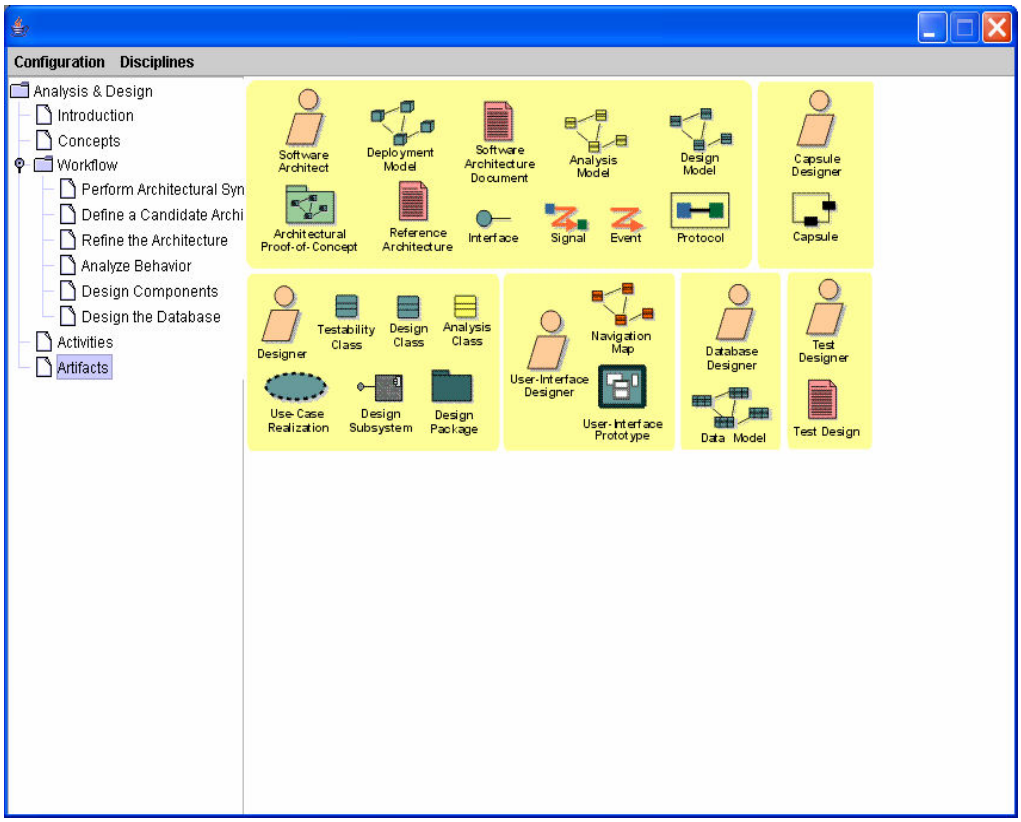

Figure 4.22: Analysis & Design – Workflow – Artifacts

4.1.3.3 Implementation:

Introduction:

In Figure 4.23 “Implementation” discipline purpose is explained.

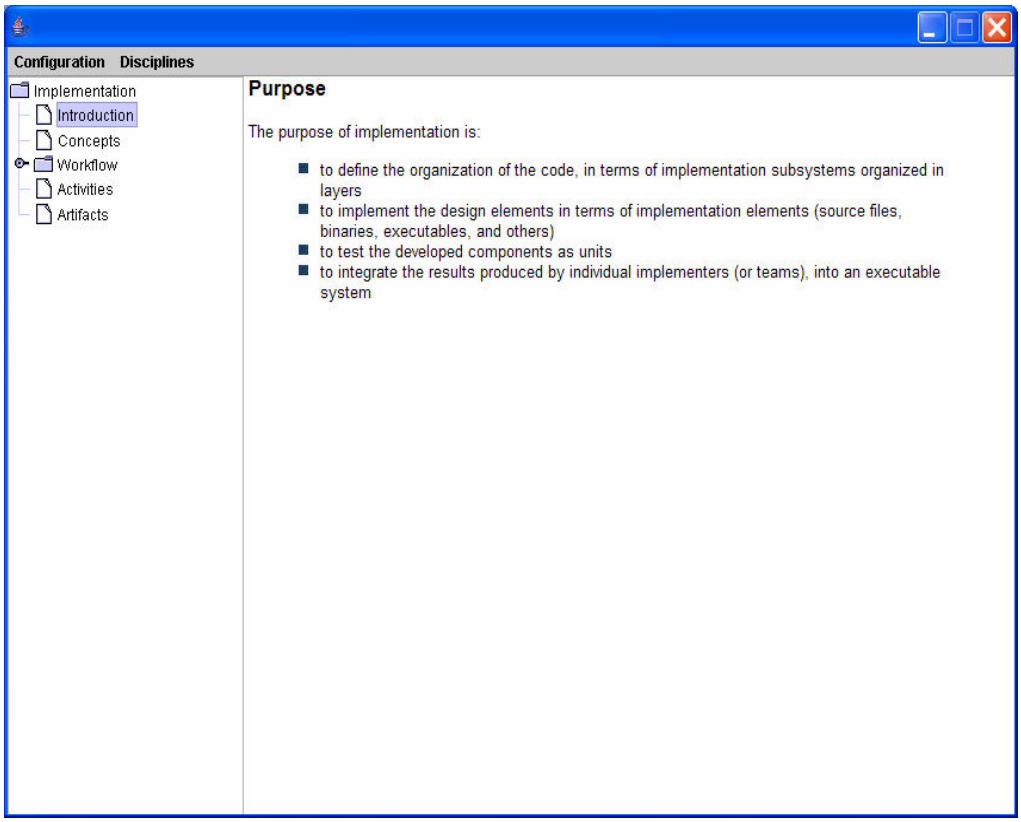

Figure 4.23: Implementation – Introduction

The purpose of implementation is:

- to define the organization of the code, in terms of implementation subsystems organized in layers
- to implement the design elements in terms of implementation elements (source files, binaries, executables, and others)
- to test the developed components as units

to integrate the results produced by individual implementers (or teams), into an executable system

The Implementation discipline limits its scope to how individual classes are to be unit tested. System test and integration test are described in the Test discipline.

Concepts:

In Figure 4.24, "Implementation" concepts are shown.

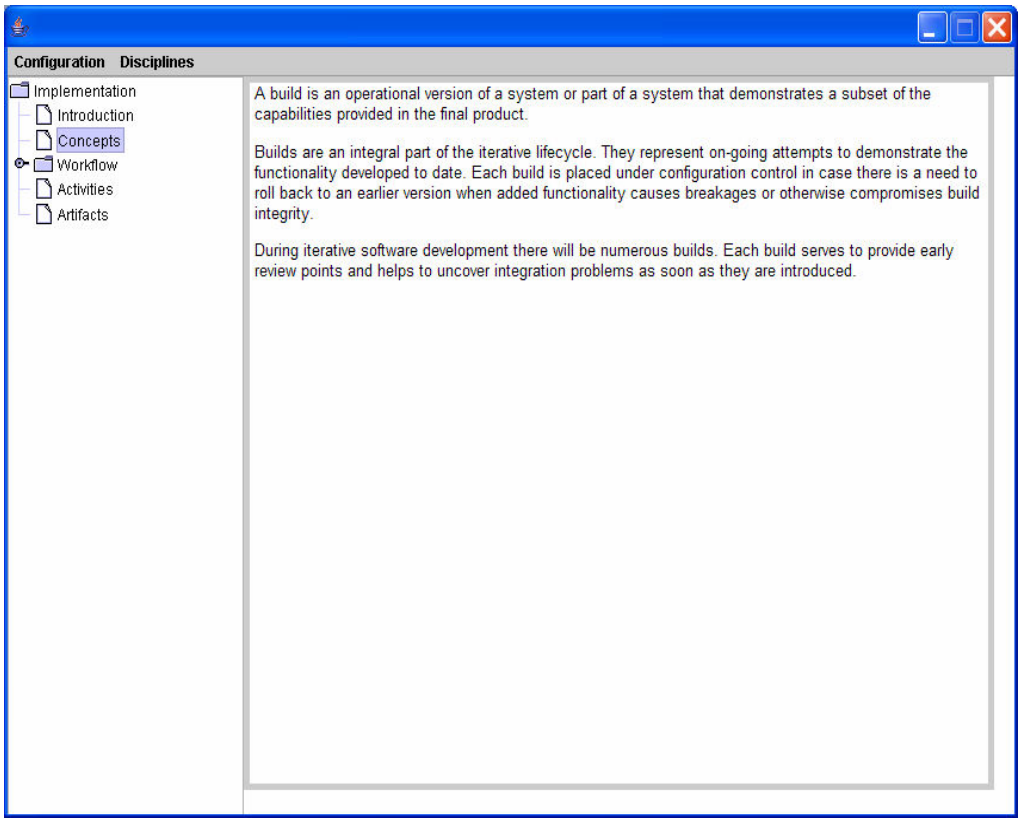

Figure 4.24: Implementation – Concepts

Workflow:

The main flowchart of this discipline is shown on this menu item. There are two ways to see the details. Project members could work on the flowchart by clicking on the activities listed on the screen or selecting from the submenu.

In Figure 4.25, "Implementation" workflow are shown.

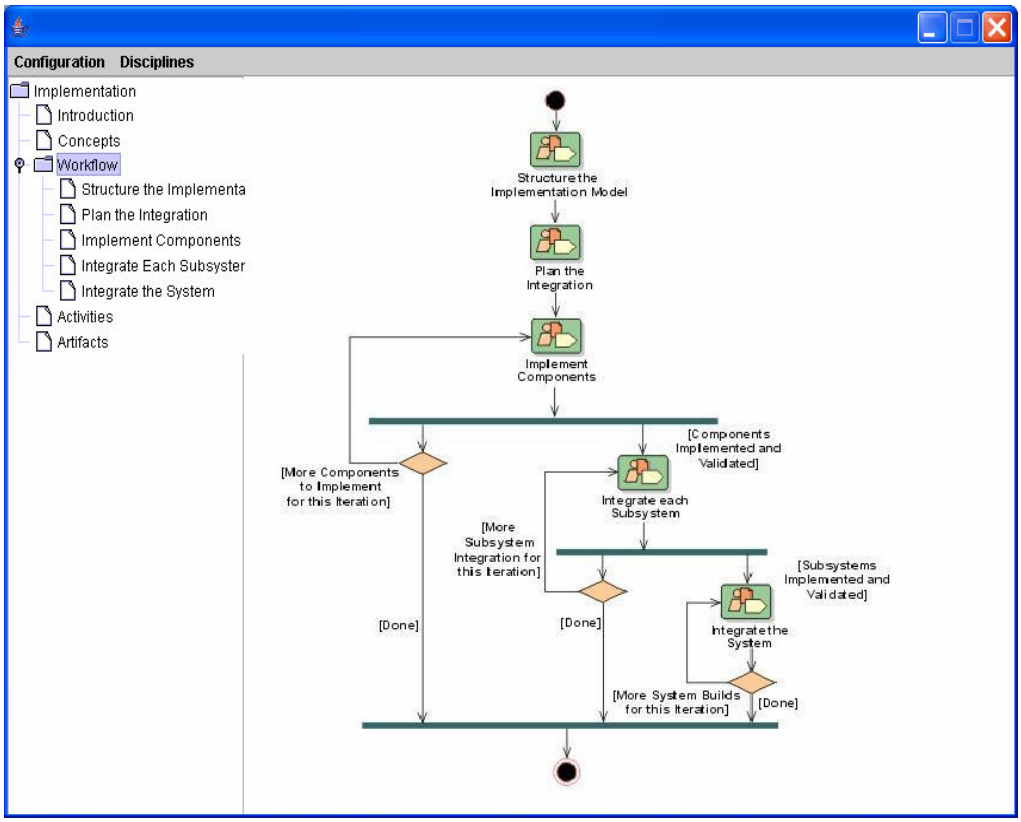

Figure 4.25: Implementation – Workflow

In Figure 4.26, "Structure the Implementation" workflow details are shown.

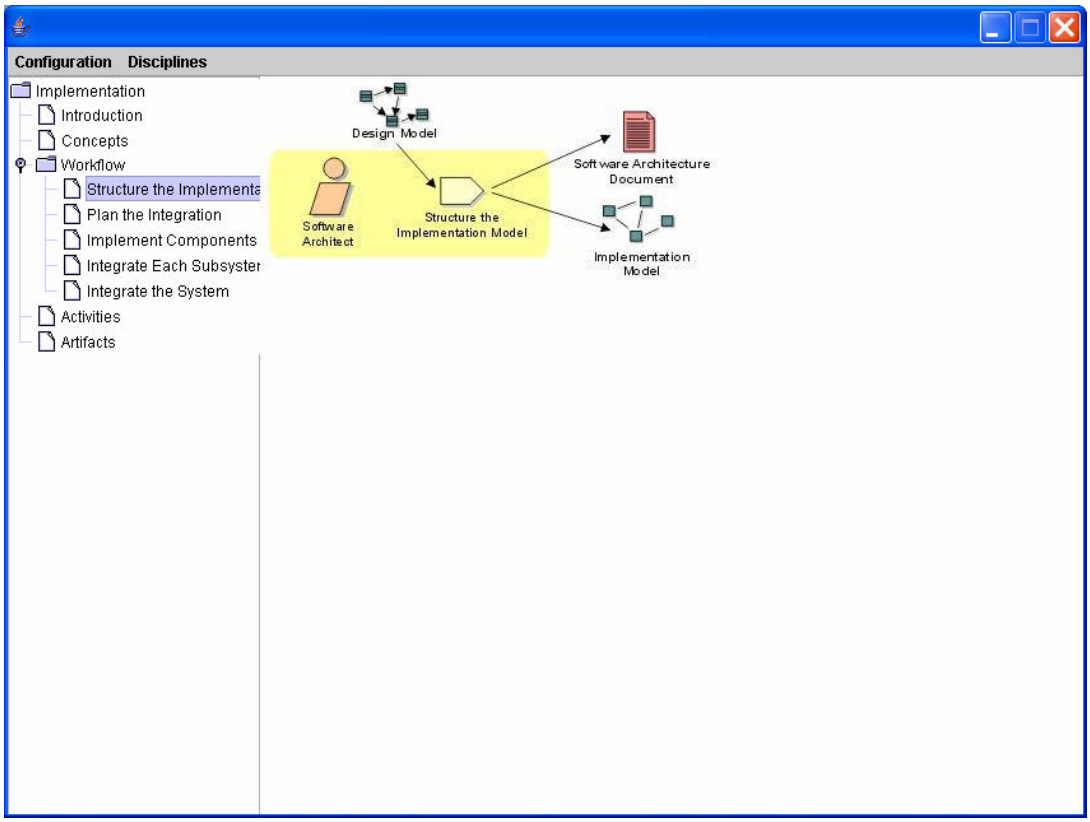

Figure 4.26: Implementation – Workflow - Structure the Implementation

Structuring the implementation model generally results in a set of Implementation Subsystems that can be developed relatively independently. A well-organized model will prevent configuration management problems and will allow the product to built-up from successively larger integration builds.

Structuring the implementation model should be done in parallel with the evolution of the other aspects of the architecture; failure to consider it early in the architecting process may lead to poor organization of the implementation and may impede the implementation and build process. In the worst case, a poorly organized implementation model will impede parallel development of software by the project team.

While the software architect has primary responsibility for the structure of the implementation model, the software architect's experience needs to include that of an integrator at the system level. They need experience in software build management, configuration management, and experience in the programming language in which

the components to be integrated are written. Because the automation of integration will be handled by the integrator, the software architect need not be an expert in scripting or integration automation, but some familiarity with the topic will often help the build process go more smoothly.

In Figure 4.27, "Plan the Integration" workflow details are shown.

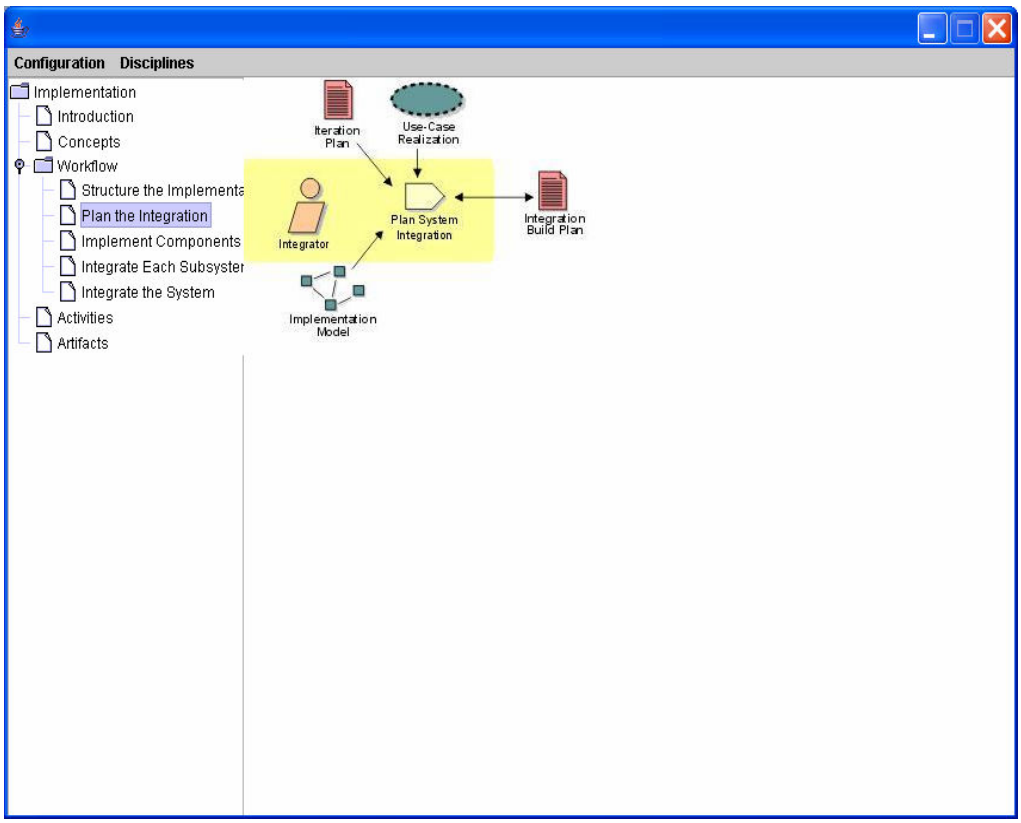

Figure 4.27: Implementation – Workflow – Plan the Integration

Planning the integration is focussed on which implementation subsystems should be implemented, and the order in which the implementation subsystems should be integrated in the current iteration. Integration is typically carried out by a single person (for a small project on which the build process in simple) or a small team (for a large project on which the build process is complex). The integrators need experience in software build management, configuration management, and experience in the programming language in which the components to be integrated are written. Because integration often involves a high degree of automation, expertise in operating system shell or scripting languages and tools like 'make' is also essential.

Planning the integration process should be done early, at least in rough form, when the architecture is baselined. As the architecture and design evolve, the integration plan should be examined and updated to ensure that the build plan does not become obsolete by changes in the architecture or the design.

In Figure 4.28, "Implementaiton Components" workflow details are shown.

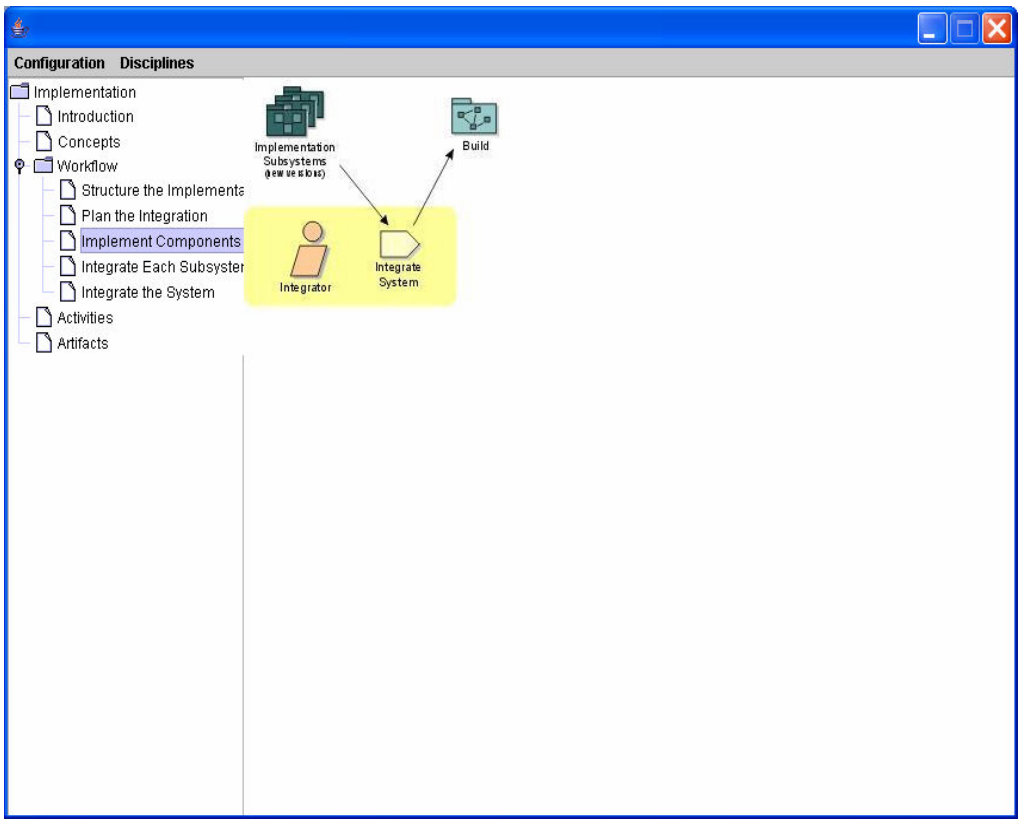

Figure 4.28: Implementation – Workflow – Implement Components

In this workflow detail:

- The implementers write source code, adapt existing source code, compile, link and perform unit tests, as they implement the elements in the design model. If defects in the design are discovered, the implementer submits rework feedback on the design.

- The implementers also fix code defects and perform unit tests to verify the changes. Then the code is reviewed to evaluate quality and compliance with the Programming Guidelines.

These activities carried out by the implementer tend to be done by a single person. The review activity is best carried out by a small team staffed by cross-functional team members, typically more senior members of technical staff with greater experience into common problems and pitfalls encountered in the programming language. Special expertise may be required in the problem domain, as is often the case in systems involving telephony or devices with special interfaces. Expertise in specific algorithms or programming techniques may also be required.

The review work is best done in several sessions, each focused on small sections of the system or on specific issues. The goal of these sessions is to identify specific problems in the code that need to be resolved, not to resolve them on the spot; resolution discussions should be postponed until after the review. More frequent reviews which are smaller in scope are more productive than less frequent sessions which are larger in scope.

In Figure 4.29, "Integrate Each SubSystem" workflow details are shown.

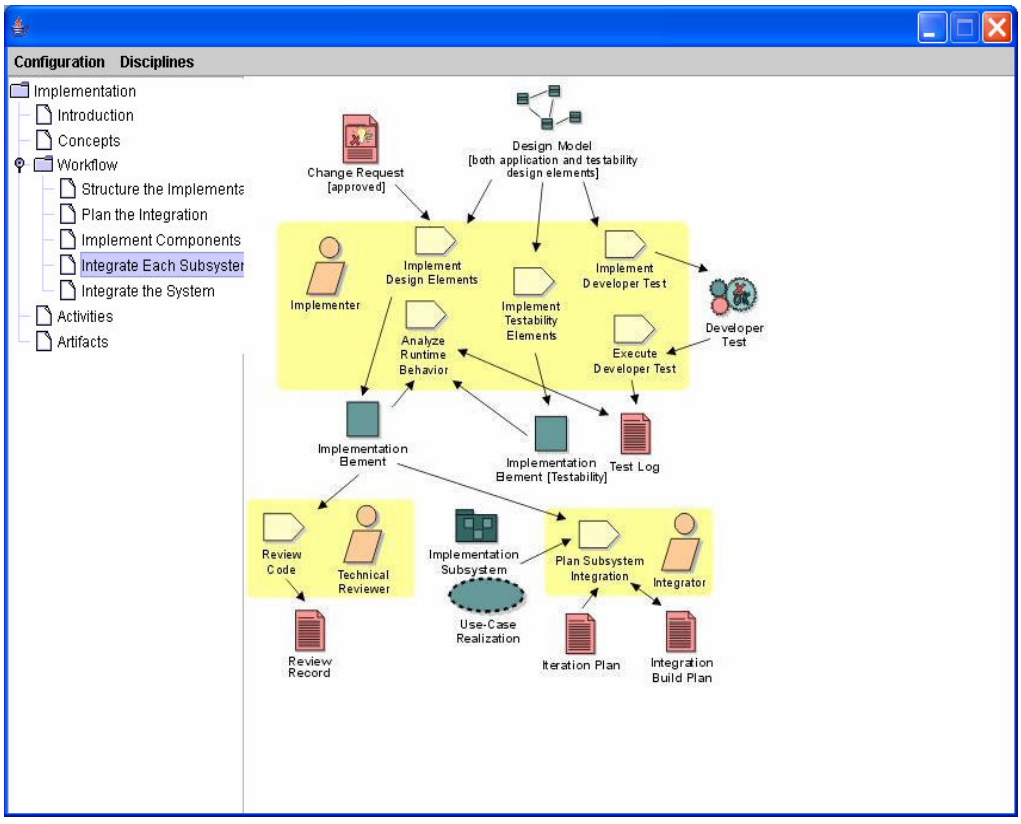

Figure 4.29: Implementation – Workflow - Integrate Each SubSystem

If several implementers work in the same implementation subsystem, the changes from the individual implementers need to be integrated to create a new consistent version of the implementation subsystem. The integration results in series of builds in a subsystem integration workspace. Each build is then integration tested by a tester and/or an implementer executing the developer tests. Following testing, the Implementation Subsystem is delivered into the system integration workspace.

Integration is typically carried out by a single person (for a small project on which the build process in simple) or a small team (for a large project on which the build process is complex). The integrators need experience in software build management, configuration management, and experience in the programming language in which the components to be integrated are written.

Integration work is typically automated to a large degree, with manual effort required when the build breaks. A frequent strategy is to perform automated nightly builds and some automated testing (usually at the unit level), allowing for frequent feedback from the build process.

In Figure 4.30, "Integrate the Subsystem" workflow details are shown.

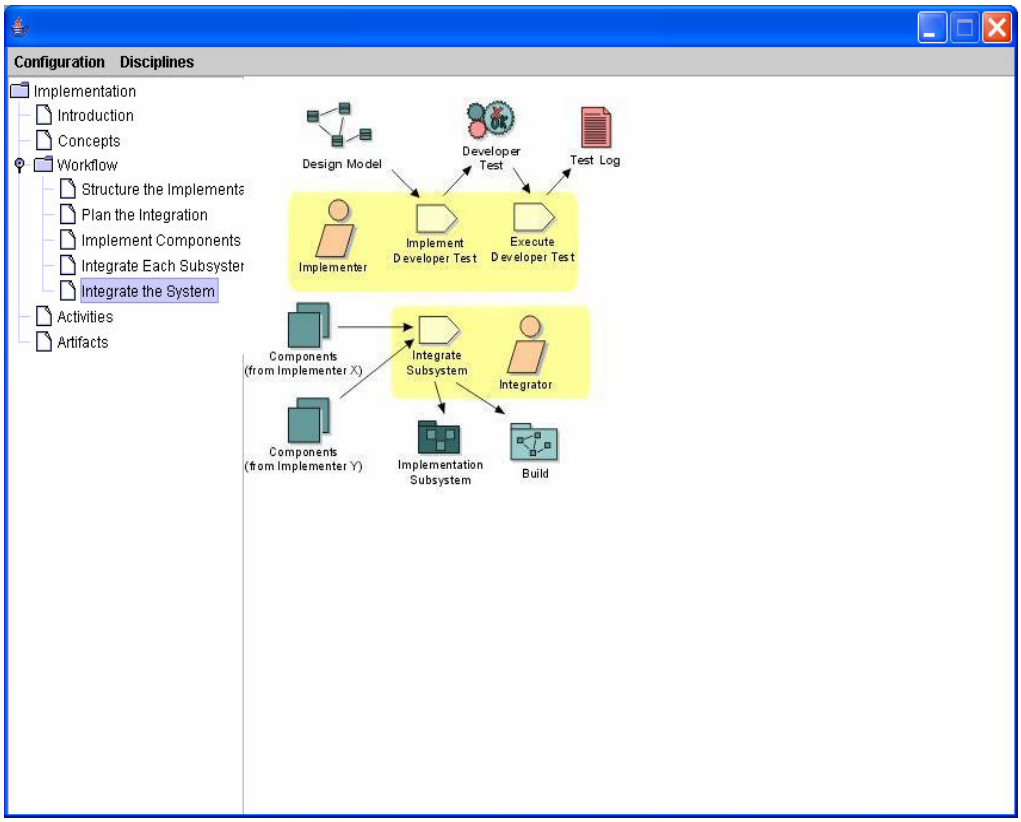

Figure 4.30: Implementation – Workflow - Integrate the System

The integrator integrates the system, in accordance with the integration build plan, by adding the delivered implementation subsystems into the system integration workspace and creating builds. Each build is then integration tested by a tester. After the last increment, the build can be completely system tested by a tester.

Integration work is typically automated to a large degree, with manual effort required when the build breaks. A frequent strategy is to perform automated nightly builds and some automated testing (usually at the unit level), allowing for frequent feedback from the build process.

Activities:

In Figure 4.31, acitivites of the "Implementation" discipline are shown. The details are explained in the workflow details meuitem. This view is added to the menu items to list the roles's responsibilities in a clear way.

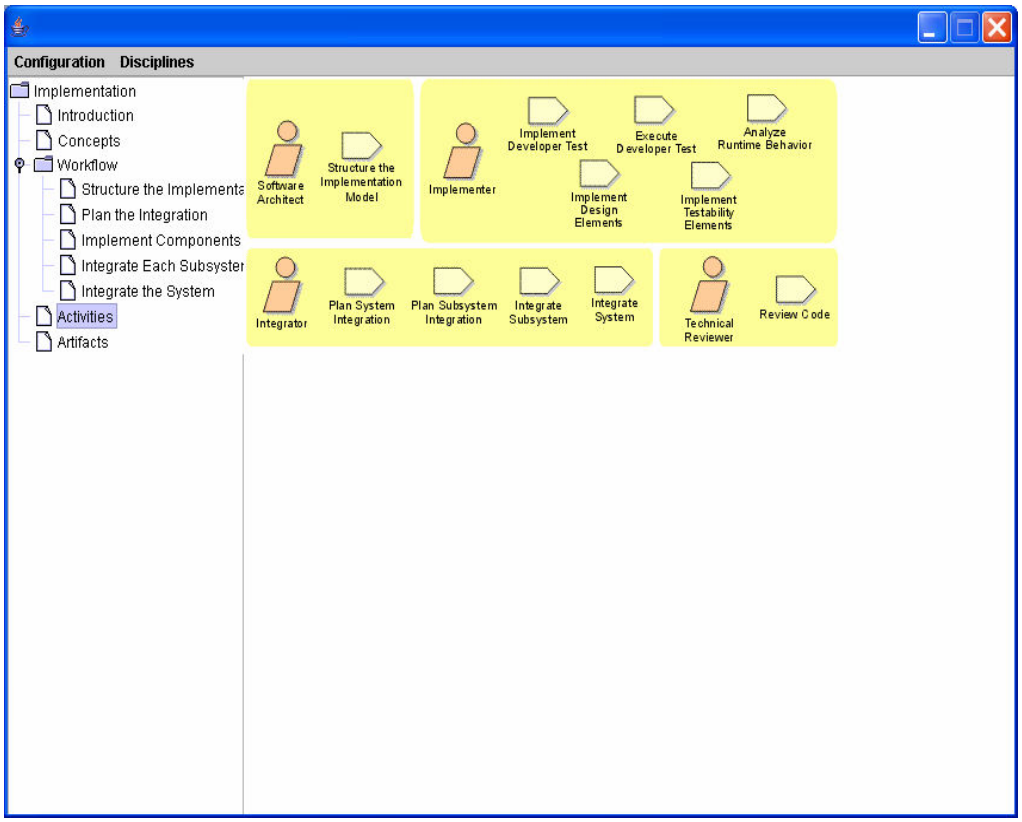

Figure 4.31: Implementation – Workflow - Activities

Artifacts:

In Figure 4.32, artifacts of the "Implementation" discipline are shown. The workflow details are explained in the workflow details meuitem. This view is added to the menuitems to list the produced artifacts.

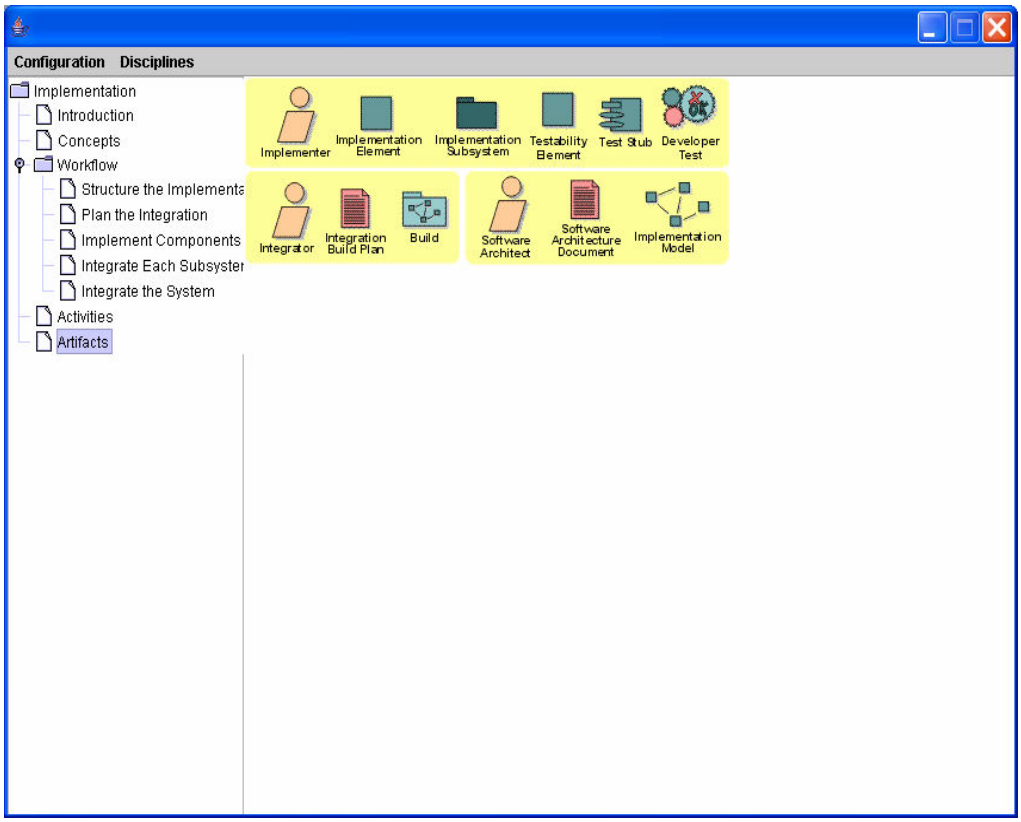

Figure 4.32: Implementation – Artifacts

4.1.3.4 Testing:

Introduction:

In Figure 4.33 "Test" discipline purpose is explained.

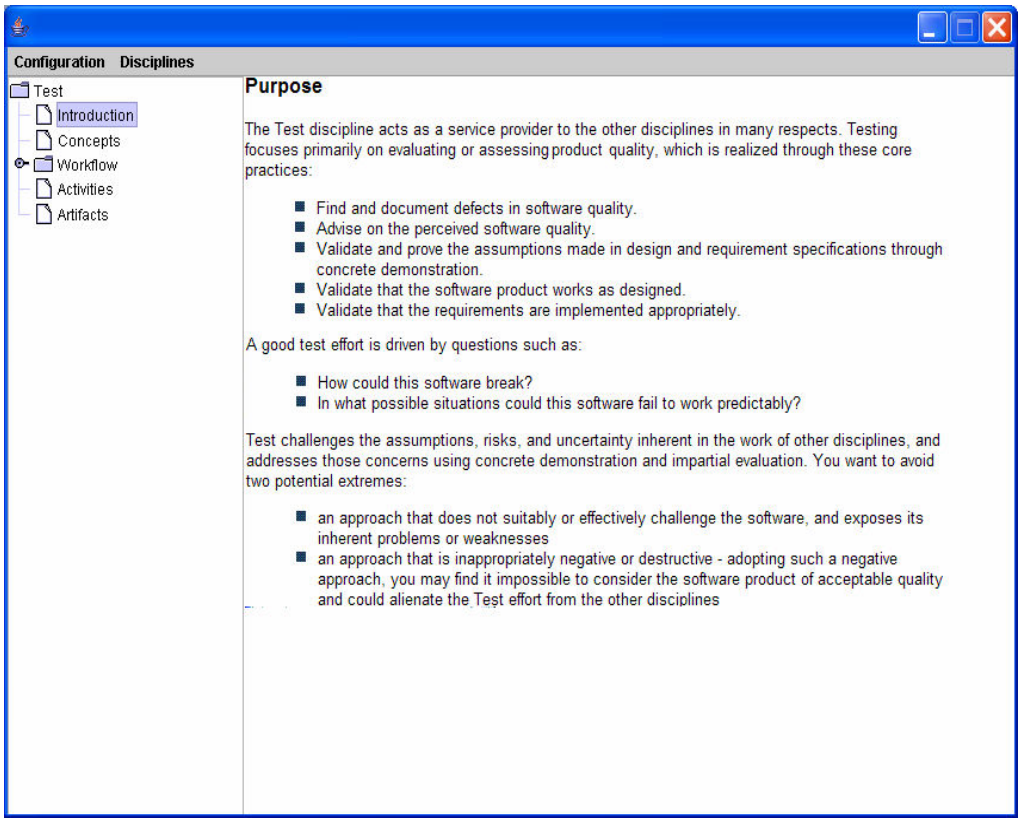

Figure 4.33: Test – Introduction

The test discipline acts as a service provider to the other disciplines in many respects. Testing focuses primarily on evaluating or assessing product quality, which is realized through these core practices:

- Find and document defects in software quality.
- Advise on the perceived software quality.
- Validate and prove the assumptions made in design and requirement specifications through concrete demonstration.
- Validate that the software product works as designed.
- Validate that the requirements are implemented appropriately.

A good test effort is driven by questions such as:

- How could this software break?

- In what possible situations could this software fail to work predictably?

Test challenges the assumptions, risks, and uncertainty inherent in the work of other disciplines, and addresses those concerns using concrete demonstration and impartial evaluation. To avoid two potential extremes:

- an approach that does not suitably or effectively challenge the software, and exposes its inherent problems or weaknesses

- an approach that is inappropriately negative or destructive - adopting such a negative approach, you may find it impossible to consider the software product of acceptable quality and could alienate the Test effort from the other disciplines

Information presented in various surveys and essays states that software testing accounts for 30 to 50 percent of total software development costs. It is, therefore, somewhat surprising to note that most people believe computer software is not well tested before it's delivered. This contradiction is rooted in a few key issues:

- Typically testing is done without a clear methodology, creating results that vary from project to project and from organization to organization. Success is primarily a factor of the quality and skills of the individuals.

- Productivity tools are used insufficiently, which makes the laborious aspects of testing unmanageable. In addition to the lack of automated test execution, many test efforts are conducted without tools that let you effectively manage extensive test data and test results. Flexibility of use and complexity of software make complete testing an impossible goal. Using a well-conceived methodology and state-of-the-art tools can improve both the productivity and effectiveness of software testing.

High-quality software is essential to the success of safety-critical systems - such as air-traffic control, missile guidance, or medical delivery systems - where a failure can harm people. The criticality of a typical system may not be as immediately obvious, but it's likely that the impact of a defect could cause the business using the software considerable expense in lost revenue and possibly legal costs. In this

information age, with increasing demands on providing electronically delivered services over the internet, many systems are now considered mission-critical; that is, companies cannot fulfill their functions and they experience massive losses when failures occur.

A continuous approach to quality, initiated early in the software lifecycle, can lower the cost of completing and maintaining your software significantly. This greatly reduces the risk associated with deploying poor quality software.

Concepts:

In Figure 4.34, "Test" concepts are shown.

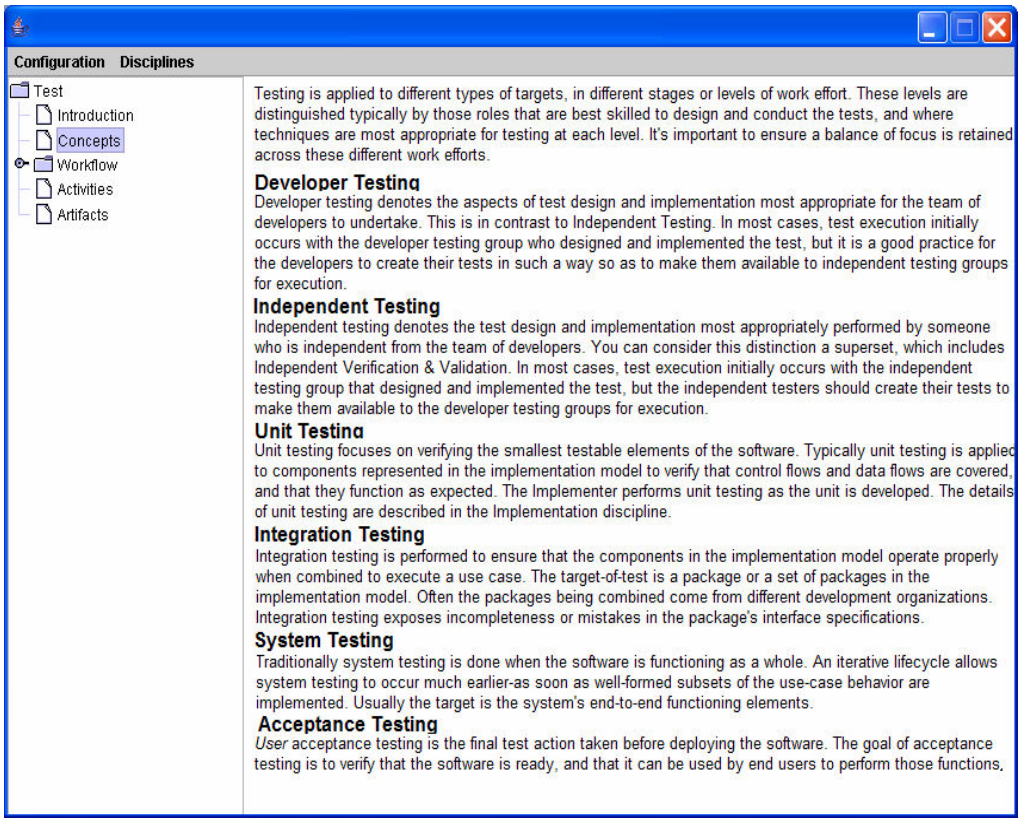

Figure 4.34: Test – Concepts

Workflow:

The main flowchart of this discipline is shown on this menu item. There are two ways to see the details. Project members could work on the flowchart by clicking on

the activities listed on the screen or selecting from the submenu. In Figure 4.35, wokflow details are shown.

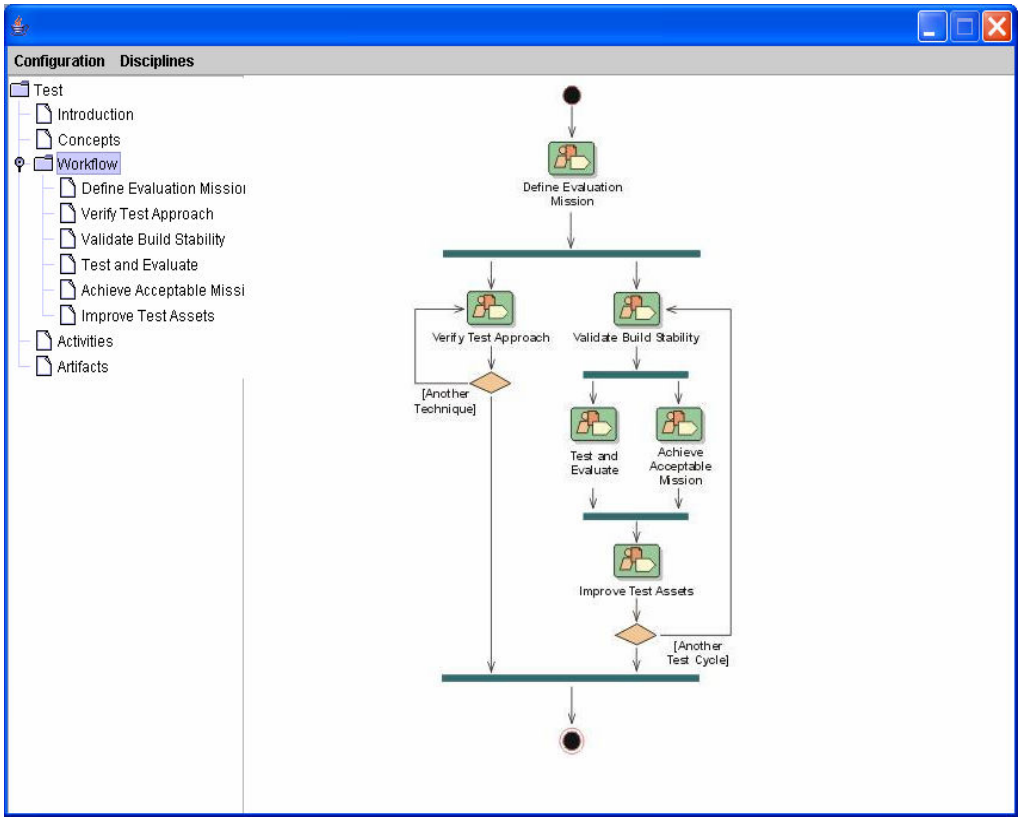

Figure 4.35: Test – Workflow

In Figure 4.36, "Define Evaluation Mission" wokflow details are shown.

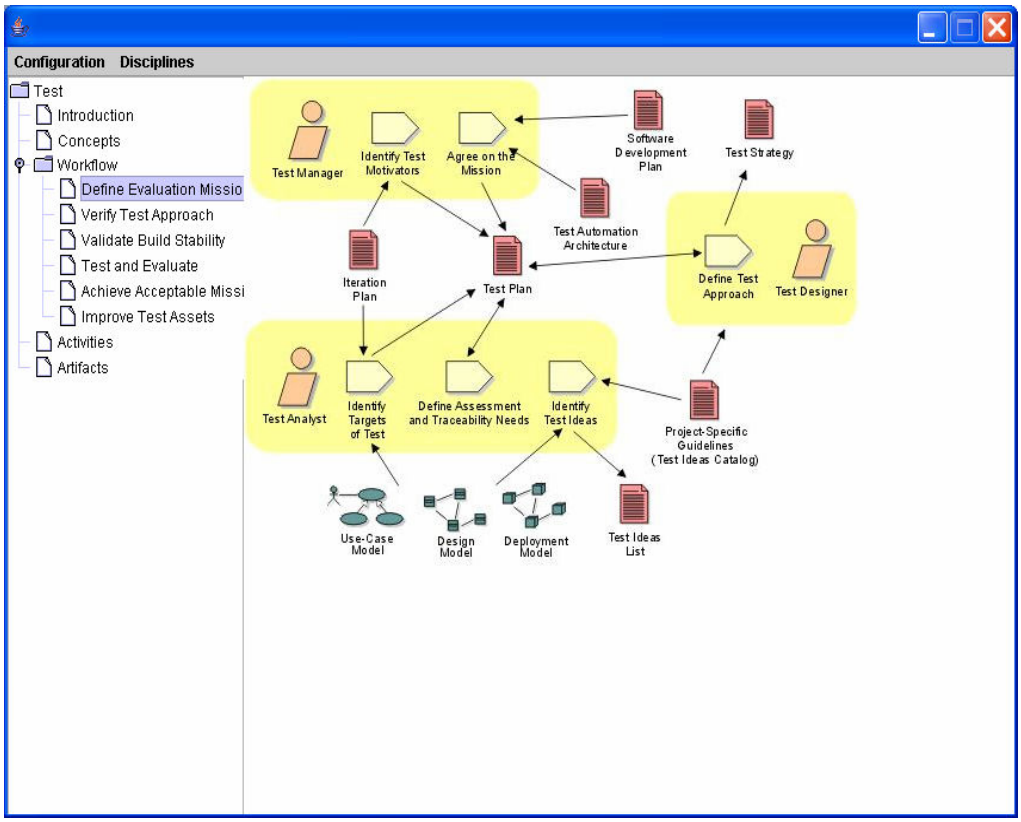

Figure 4.36: Test – Workflow – Define Evaluation Mission

For each iteration, this work is focused mainly on:

- Identifying the objectives for, and deliverables of, the testing effort
- Identifying a good resource utilization strategy
- Defining the appropriate scope and boundary for the test effort
- Outlining the approach that will be used
- Defining how progress will be monitored and assessed.

It should be noted that this work is performed in each iteration. The main value in performing this work is to think through the various concerns and issues that will impact testing over the course of the iteration, and consider the appropriate actions you should take. As a general rule, don't spend excessive amounts of time on the presentation of the documentation for these aspects of the test effort.

Although most of the roles involved in the Test discipline play a part in performing this work, the effort is primarily centered around the Test Manager and Test Analyst roles. The most important skills required for this work include negotiation, elicitation, strategy and planning.

While most of the resource for this work will be expended in Construction, significant resources will need to be allocated to this work from Inception to Transition.

As a relative indication of test resource use for this workflow detail by phase, typical percentages are: Inception - 50%, Elaboration - 25%, Construction - 10% and Transition - 10%.

In Figure 4.37, "Verify Test Approach" workflow details are shown.

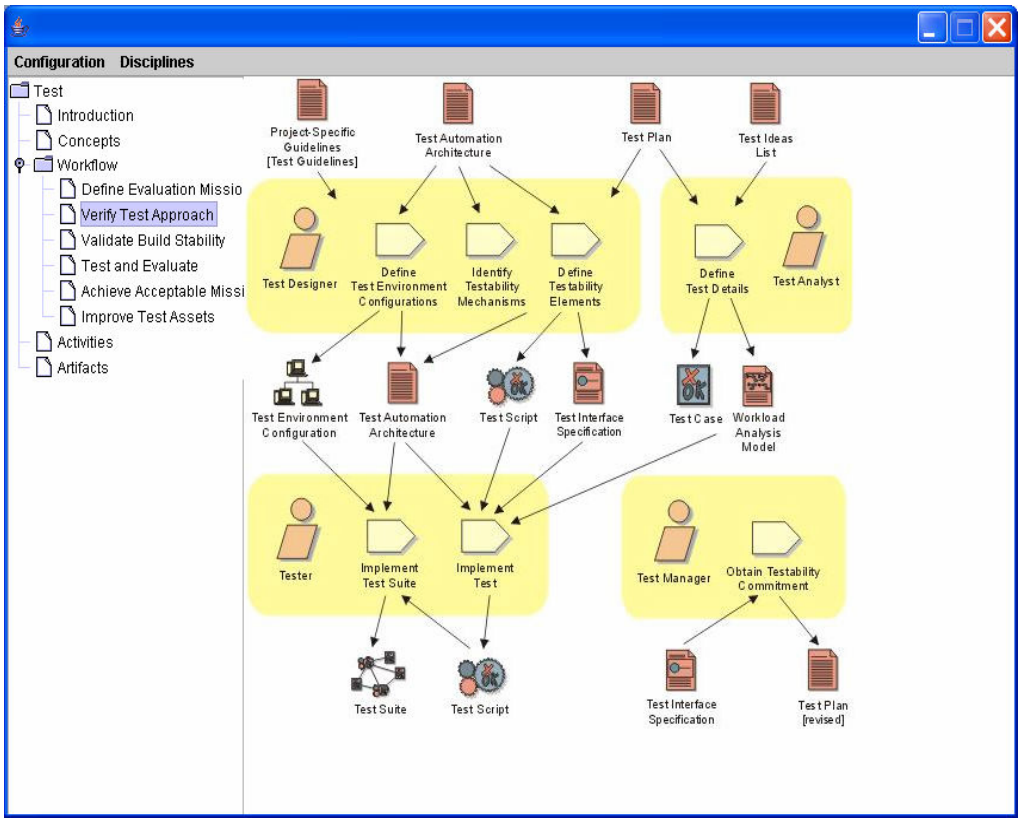

Figure 4.37: Test – Workflow - Verify Test Approach

The objective is to gain an understanding of the constraints and limitations of each technique as it will be applied in the given project context, and to either:

- find an appropriate implementation solution for each technique or
- find alternative techniques that can be used.

This helps to mitigate the risk of discovering too late in the project lifecycle that the test approach is unworkable. For each iteration, this work is focused mainly on:

- Early verification that the intended test strategy will work and produces results of value
- Establishing the basic infrastructure to enable and support the test strategy
- Obtaining commitment from the development team to develop the software to meet testability requirements necessary to achieve the test strategy, and to provide continued support for those testability requirements.
- Identifying the scope, boundaries, limitations and constraints of each technique

This work is somewhat independent of the test cycles, often involving the verification of techniques that will not be used until subsequent Iterations. This work normally begins after the evaluation mission has been defined for the current Iteration, although it can begin earlier. In some cases, finding the best implementation approach to a technique may take multiple Iterations.

The test implementation and execution activities that form a part of this work are performed for the purpose of obtaining demonstrable proof that the techniques being verified can actually work. As such, you should limit your selection of tests to a small, representative subset; typically focusing on areas with substantial quality risk. You should try to include a selection of tests that you expect to fail to confirm that the technique will successfully detect these failures.

While failures with the target test items will be identified and these incidents logged accordingly, this focus of this work is not directly on attempting to identify failures in the target test items as it's main objective. Again, the objective is to verify that the approach is appropriate (it produces results that complement the Iteration objectives),

is achievable (it can be implemented with given resource constraints), and that it will work.

For this work to produce timely results, it is often necessary to make use of incomplete, "unofficial" Builds, or to perform this work outside of a recognized Test Environment Configuration. Although these are appropriate compromises, be aware of the constraints, assumptions and risks involved in verifying your approach in under these conditions.

As the lifecycle progresses through its Phases, the focus of the test effort typically changes. Potentially this requires new or additional approaches, often requiring the introduction of new types of tests or new techniques to support the test effort.

In situations where the combination of domain, the test environment and other critical aspects of the strategy are unprecedented, you should allow more time and effort to complete this work. In some cases-especially where automation is a requirement-it may be more economic to obtain the use of resources with specialized skills that have proven experience in the unprecedented aspects of the strategy for a limited period of time (such as on contract) to define and verify the key technical needs of the test strategy.

Although most of the roles involved in the Test discipline play a part in performing this work, the effort is primarily centered around the Test Designer and Tester roles. The most important skill areas required for this work include software architecture, software design and problem solving.

It is typical for this work to require more resource in iterations from the late Inception to early Construction phases, often requiring minimal resource late in the Construction and in the Transition phases. However, be aware that as the project progresses, new objectives or deliverables may be identified that require new test strategies to be defined and verified.

As a heuristic for relative resource allocation by phase, typical percentages of test resource use for this workflow detail are: Inception - 30%, Elaboration - 20%, Construction - 10% and Transition - 05%.

In Figure 4.38, "Validate Build Stability" workflow details are shown.

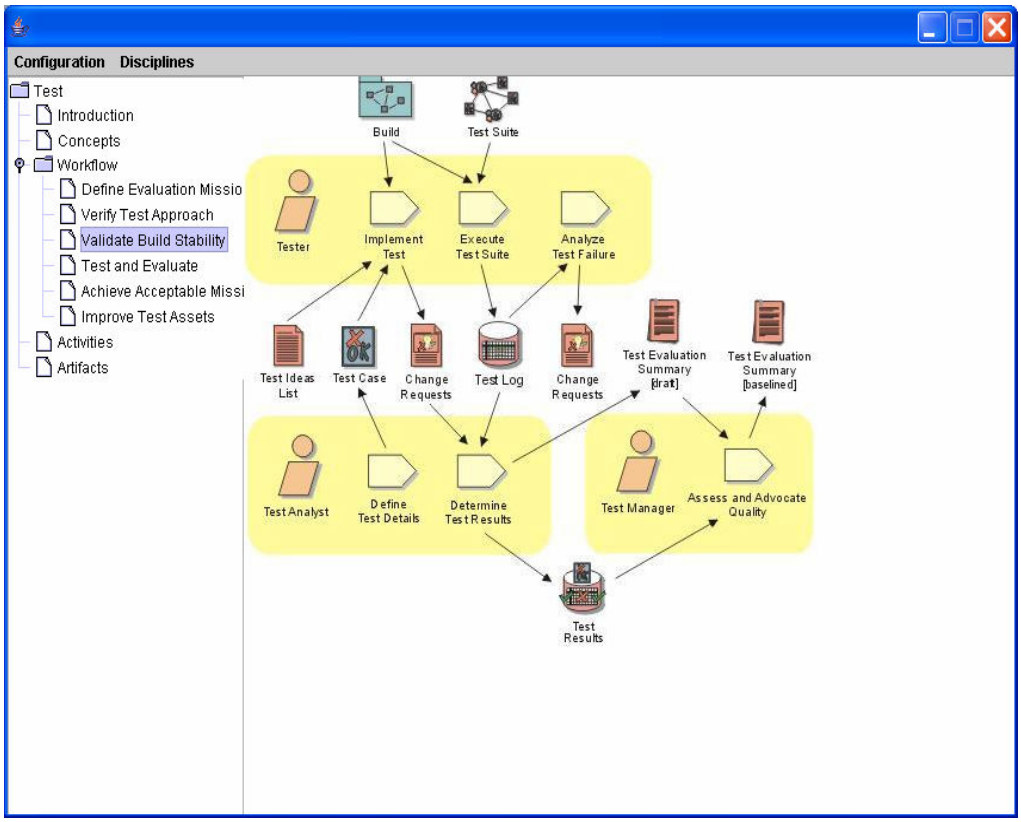

Figure 4.38: Test – Workflow – Validate Build Stability

For each build to be tested, this work is focused on:

- Making as assessment of the stability and testability of the Build
- Gaining an initial understanding-or confirming the expectation-of the development work delivered in the Build
- Making a decision to accept the Build as suitable for use-guided by the evaluation mission-in further testing, or to conduct further testing against a previous Build.

This work is potentially conducted once per Build-note however that it's typical not to test every Build. Once the Build is determined suitably stable, focus turns to "Workflow Detail: Test and Evaluate". Where it is determined that the build is

sufficiently unsuitable to conduct further testing against, Test and Evaluation work typically recommences against a previous suitable Build.

The work is primarily centered around the Tester and Test Analyst roles. The most important skills required for this work include providing timely results, thoroughness and applying reasonable judgment to assessing the usefulness of the Build for further testing.

It is appropriate to allocate a subset of the test team to perform this work; the other team members ignore the new build until it is validated as stable, devoting their efforts instead to either additional tests against the build from the previous test cycle, or improving test assets as appropriate.

The sophistication and availability of test automation tools and the necessary prerequisite skills to use them will have an impact on the resourcing of this work. Where automation tools are used, much of this work can be performed fast and efficiently: without automation significantly more effort is required.

In Figure 4.39, "Test and Evaluate" workflow details are shown.

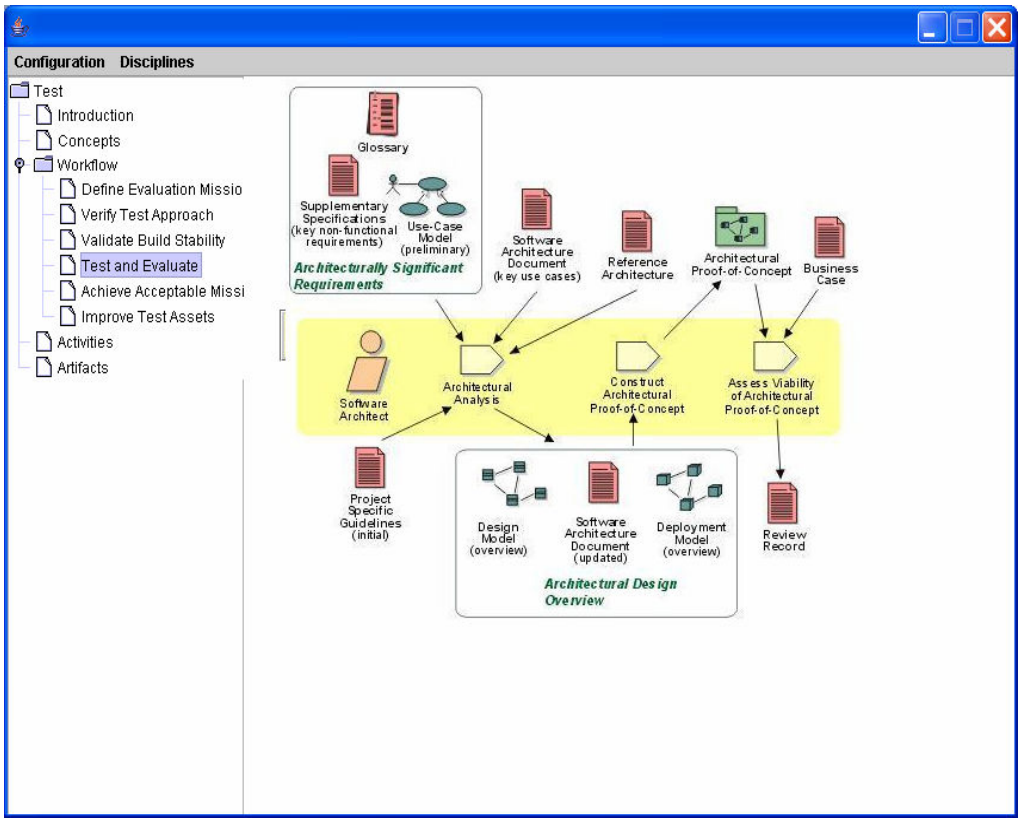

Figure 4.39: Test – Workflow – Test and Evaluate

Typically performed once per test cycle, this work involves performing the core tactical work of the test and evaluation effort: namely the implementation, execution and evaluation of specific tests and the corresponding reporting of incidents that are encountered [35].

For each test cycle, this work is focused mainly on:

- Providing ongoing evaluation and assessment of the Target Test Items
- Recording the appropriate information necessary to diagnose and resolve any identified Issues
- Achieving suitable breadth and depth in the test and evaluation work
- Providing feedback on the most likely areas of potential quality risk

As noted, this work is typically performed multiple times during an iteration; the actual number of times often equating to once per Build. It should be noted however

that it's typical not to test every Build. Build schedule will often result in this work increasing in frequency during the course of the iteration. The need for additional cycles is governed by assessing when appropriate breadth and depth of testing is achieved within a test cycle, which is the focus of the Workflow Detail: Achieve Acceptable Mission.

For iterations prior to and including those early in the Construction phase, additional effort is usually required to address tactical problems encountered for the first time during test implementation and execution. These issues often detract from the number of actual tests successfully implemented and executed and limit either the breadth or depth of the testing.

The sophistication and availability of test automation tools and the necessary prerequisite skills to use them effectively will have an impact on the resourcing of this work. It may be appropriate to strategically deploy specialized contract resource for some part of this work to improve the likelihood of success. It may also be more economical to lease the automation tools and contract appropriately skilled people to use the tools, especially to help mitigate the risks in getting started. The benefits of this approach with the necessity to develop in-house skills to maintain automation assets into the future should be balanced.

The work is primarily centered around the Tester and Test Analyst roles. The most important skills required for this work include investigative and analytical skills, tenacity, thoroughness, good technical knowledge and good verbal and written communication skills (documentation of incidents, change requests and so on).

As a heuristic for relative resource allocation by phase, typical percentages of test resource use for this workflow detail are: Inception - 05%, Elaboration - 25%, Construction - 40% and Transition - 35%.

Where the requirement for test automation is particularly important, it may be useful to assign the creation and maintenance of automation assets to a separate sub-team, allowing them to specialize on automation concerns. This allows the other team members to focus on the improvement of non-automation test assets.

In Figure 4.40, "Achieve Acceptable Mission" workflow details are shown.

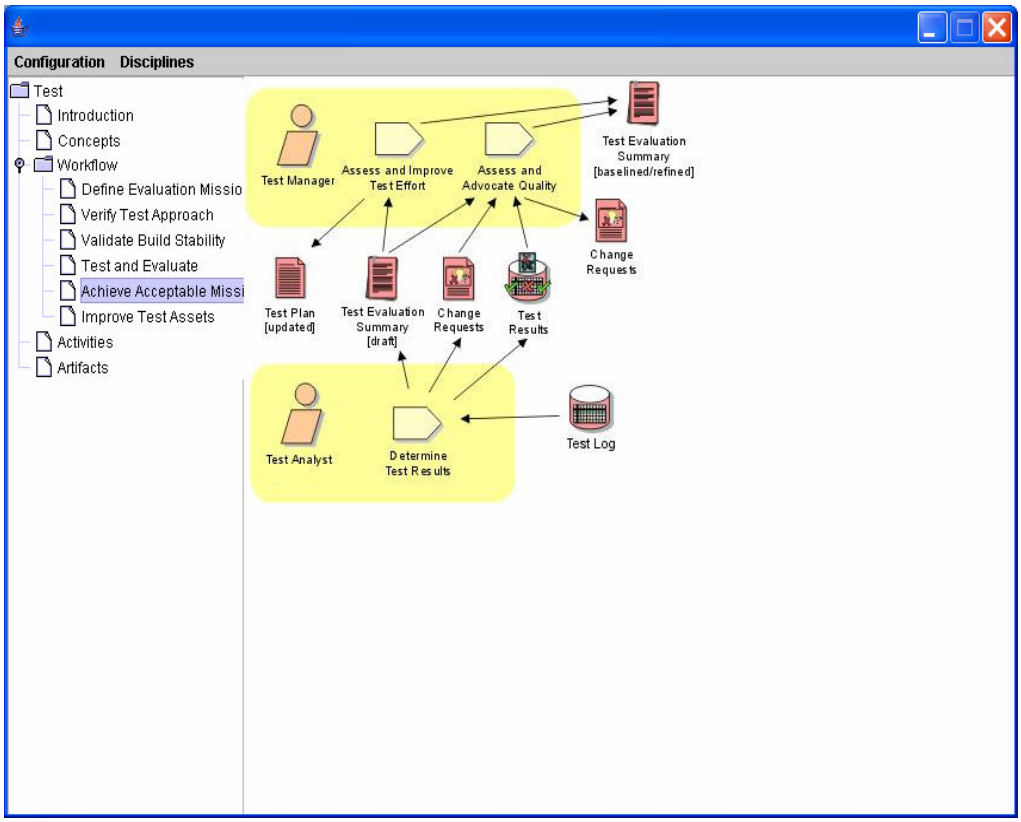

Figure 4.40: Test – Workflow – Achieve Acceptable Mission

For each test cycle, this work is focused mainly on:

- Actively prioritizing the minimal set of necessary tests that must be conducted to achieve the Evaluation Mission
- Advocating the resolution of important issues that have a significant negative impact on the Evaluation Mission
- Advocating appropriate quality
- Identifying regressions in quality introduced between test cycles
- Where appropriate, revising the Evaluation Mission in light of the evaluation findings so as to provide useful evaluation information to the project team

Given that providing focused evaluation feedback and achieving test-cycle closure are the objectives of this work, ongoing prioritization of the work and strategic management of the test resources is required. Focus continually on identifying and

executing the minimum set of specific tasks to achieve the evaluation mission. Ongoing involvement by the stakeholders in the test and evaluation effort is critical to ensure the appropriate focus is maintained and, ultimately, that the work is successful.

Notice that for some iterations it may not be possible to achieve the Evaluation Mission as originally defined. Rather than simply abandoning the test and evaluation effort, it is important to find an appropriate and agreeable revision of the original Evaluation Mission based on the current situation, and attempt to provide useful evaluation information to the stakeholders of the test effort.

This work typically starts toward the end of each test cycle as suitable breadth and depth is achieved in the testing effort. For test cycles earlier in the project lifecycle, there is typically less work to be managed, therefore less effort is required to address this workflow detail. In later iterations-especially those toward the end of the Elaboration phase and throughout the Construction phase-this work becomes more important and typically requires more focused effort.

The availability of analysis tools that provide accurate and timely results has an impact on resourcing this work. Without the use of appropriate tools, this task quickly becomes unmanageable as the test effort progresses and increasingly more detail needs to be analyzed and assessed manually.

This work is primarily centered around the Test Manager and Test Analyst roles, although success relies heavily on the work of the Tester. The most important skills required for this work include problem and results analysis, communication and negotiation, as well as the ability to identify and focus on the most important items.

As a heuristic for relative resource allocation by phase, typical percentages of test resource use for this workflow detail are: Inception - 10%, Elaboration - 00%, Construction - 20% and Transition - 30%.

In Figure 4.41, “Improve Test Assests” workflow details are shown.

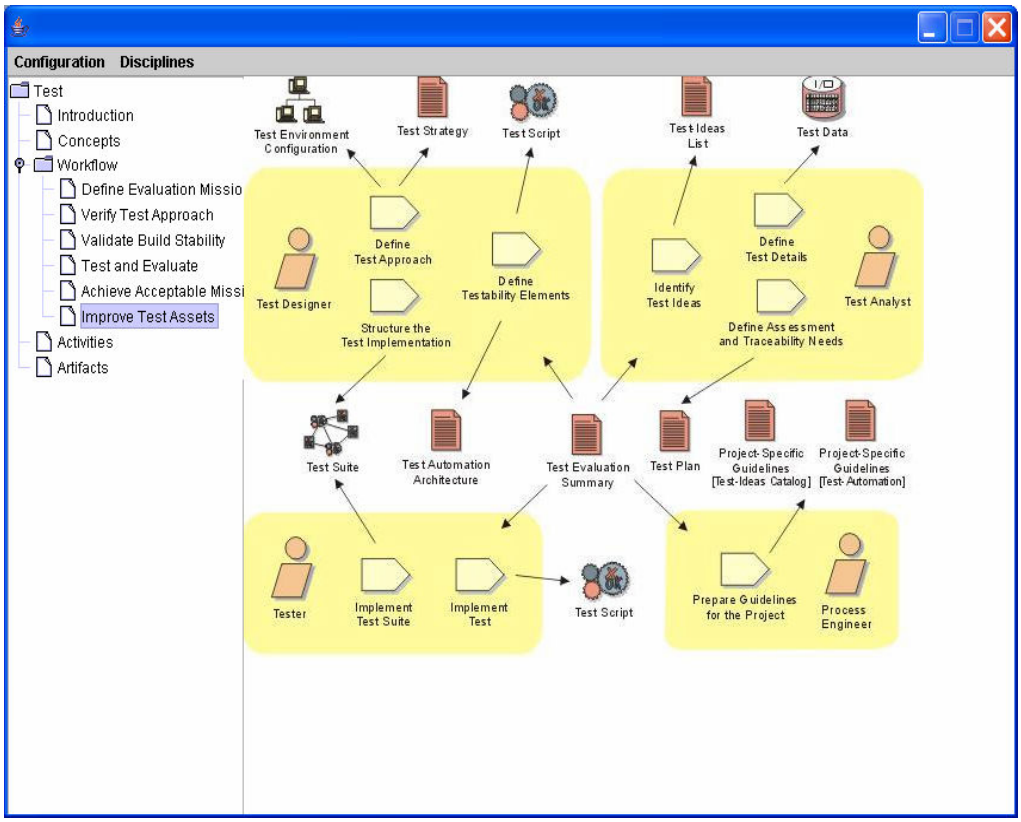

Figure 4.41: Test – Workflow – Improve Test Assets

For each test cycle, this work is focused mainly on:

- Adding the minimal set of additional tests to validate the stability of subsequent Builds
- Removing test assets that no longer serve a useful purpose or have become uneconomic to maintain
- Conducting general maintenance of and making improvements to the maintainability of test automation assets
- Assembling test scripts into additional appropriate test suites
- Exploring opportunities for reuse and productivity improvements
- Maintaining test environment configurations and test data sets

- Documenting lessons learned-both good and bad practices discovered during the test cycle.

This work typically occurs at the end of each test cycle, however some teams perform aspects of this work only once per Iteration. A common practice is to focus the work in each test cycle on adding and maintaining only those tests necessary to assess the stability for the build in the subsequent test cycle. After the final build for the iteration has been tested, other aspects of test asset improvement may also be explored.

Although most of the roles in the Test discipline play a part in performing this work, the effort is primarily centered around the Test Designer and Tester roles. The most important skills required for this work include focus on test asset coverage, an eye for potential reuse, consistency of test assets and an appreciation of architectural issues.

As a heuristic for relative resource allocation by phase, typical percentages of test resource use for this workflow detail are: Inception - 05%, Elaboration - 20%, Construction - 10% and Transition - 10%.

Where the requirement for test automation is particularly important, this work may take more effort and, therefore, more time or more resource. In some cases it may be useful to assign the creation and maintenance of automation assets to a separate sub-team, allowing them to specialize on automation concerns. This allows the other team members to focus on the improvement of non-automation test assets.

Activities:

In Figure 4.42, acitivites of the “Test” discipline are shown. The details are explained in the workflow details meuitem. This view is added to the menuitems to list the roles’s responsibilities in a clear way.

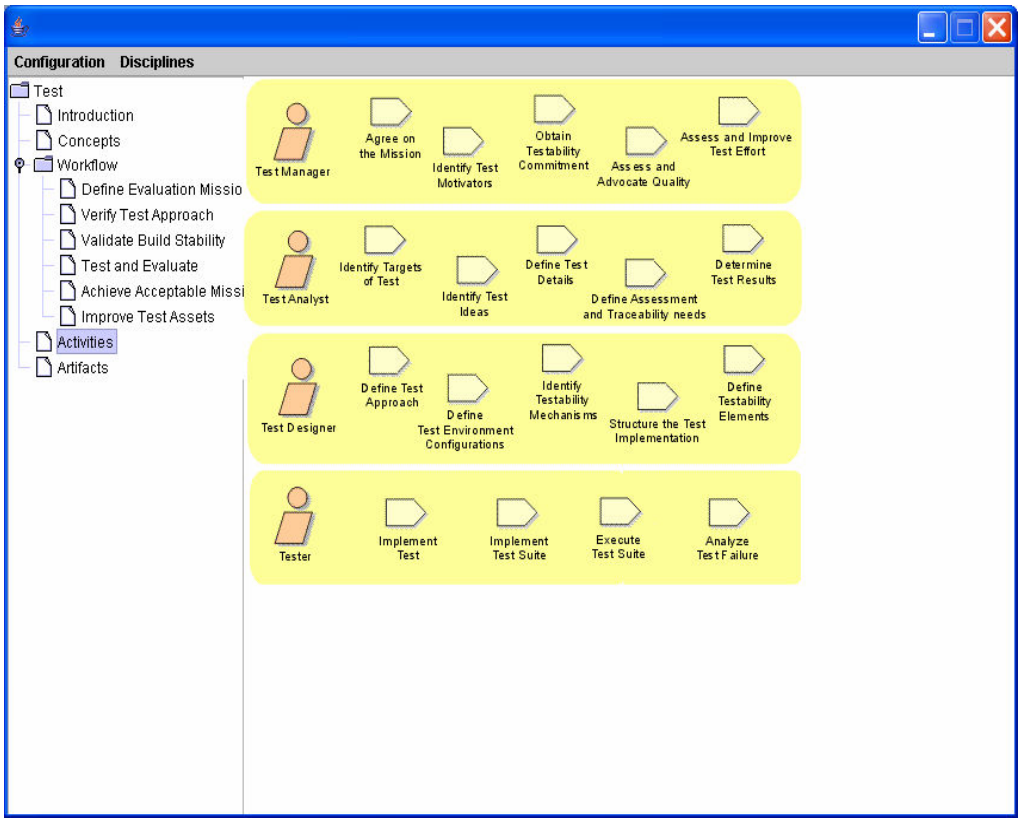

Figure 4.42: Test – Activities

Artifacts:

In Figure 4.43, artifacts of the "Test" discipline are shown. The workflow details are explained in the workflow details menu item. This view is added to the menu items to list the produced artifacts.

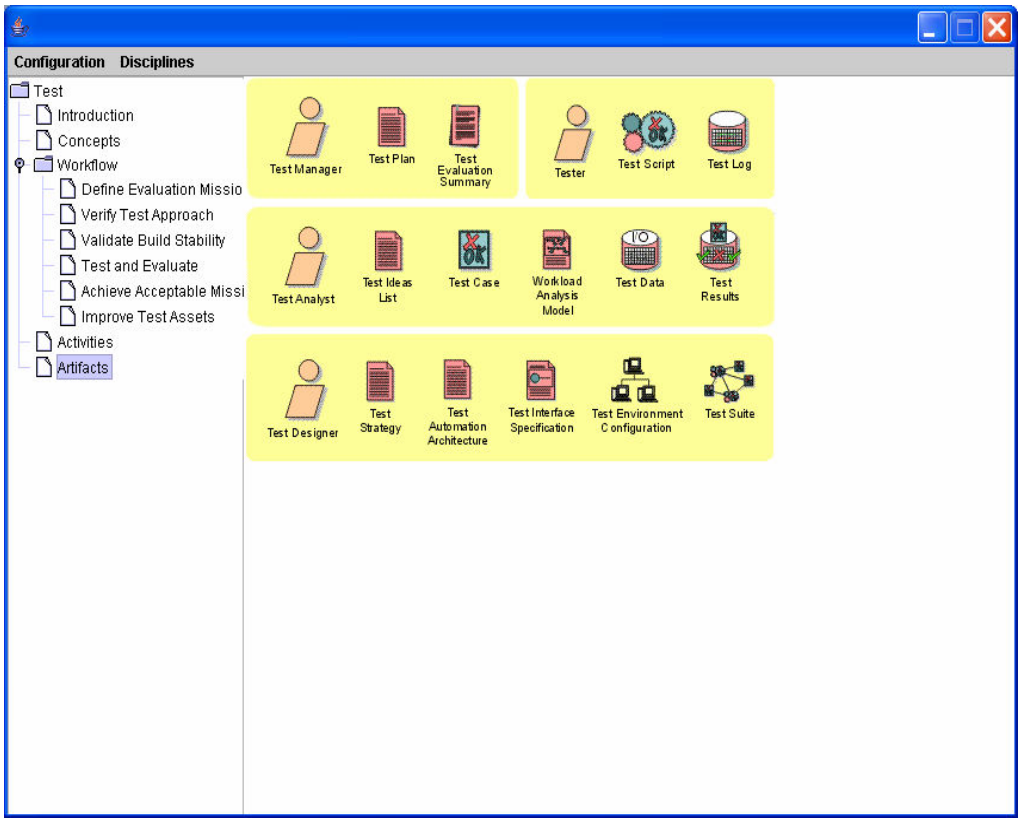

Figure 4.43: Test – Workflow - Artifacts

4.1.3.5 Deployment:

Introduction:

In Figure 4.44 "Deployment" discipline purpose is explained.

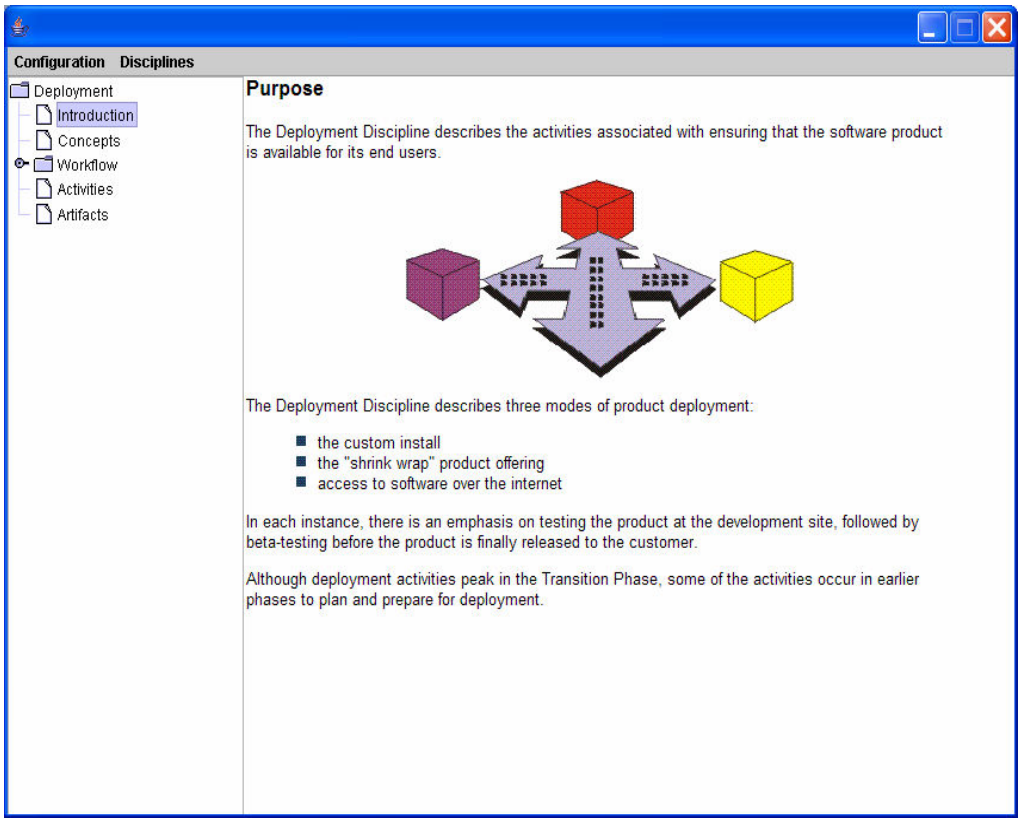

Figure 4.44: Deployment - Introduction

The “Deployment” discipline describes the activities associated with ensuring that the software product is available for its end users.

The “Deployment” discipline describes three modes of product deployment:

- the custom install
- the product offering
- access to software over the internet

In each instance, there is an emphasis on testing the product at the development site, followed by beta-testing before the product is finally released to the customer.

Although deployment activities peak in the transition phase, some of the activities occur in earlier phases to plan and prepare for deployment.

Concepts:

In Figure 4.45, "Deployment" concepts are shown.

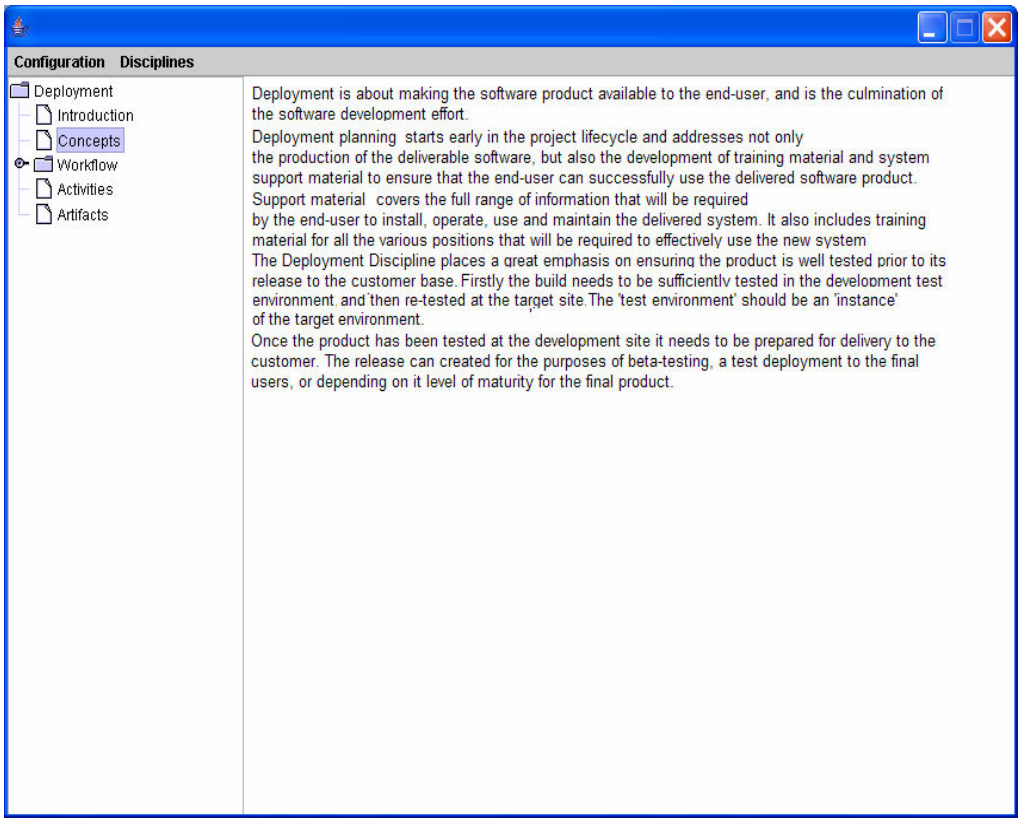

Figure 4.45: Deployment - Concepts

Workflow:

The main flowchart of this discipline is shown on this menuitem. There are two ways to see the details. Project members could work on the flowchart by clicking on the activities listed on the screen or selecting from the submenu. In Figure 4.43 workflow is shown.

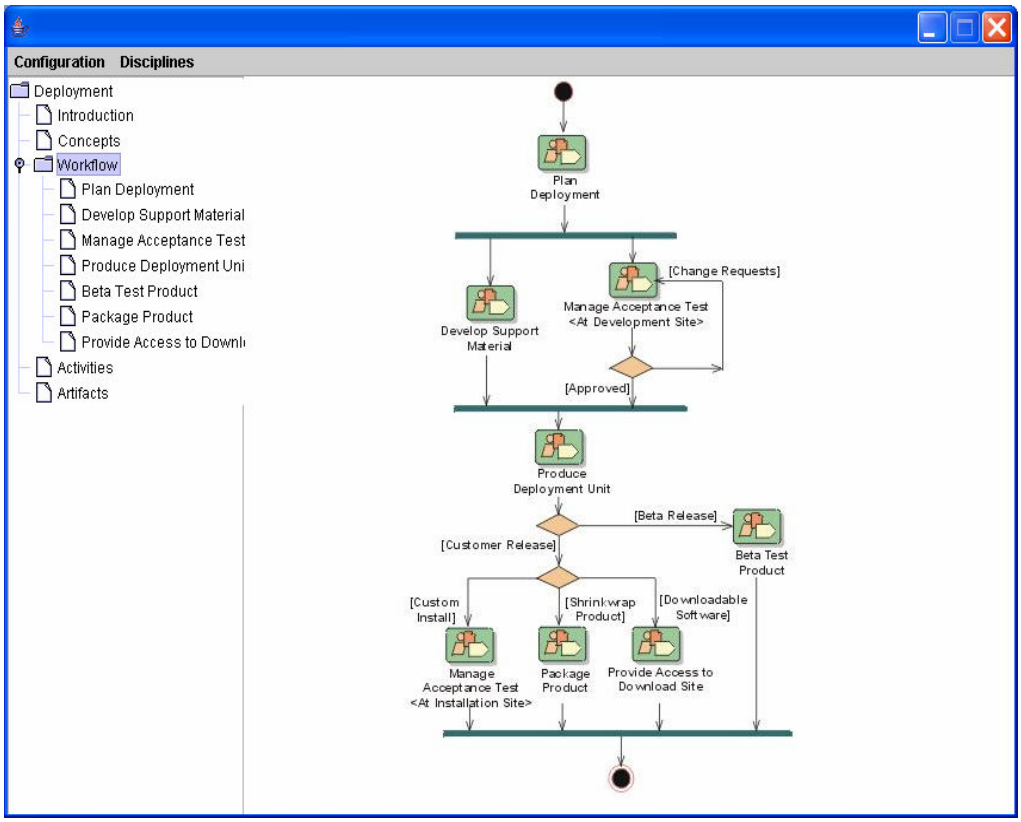

Figure 4.46: Deployment – Workflow

In Figure 4.47, "Plan Deployment" workflow details are shown.

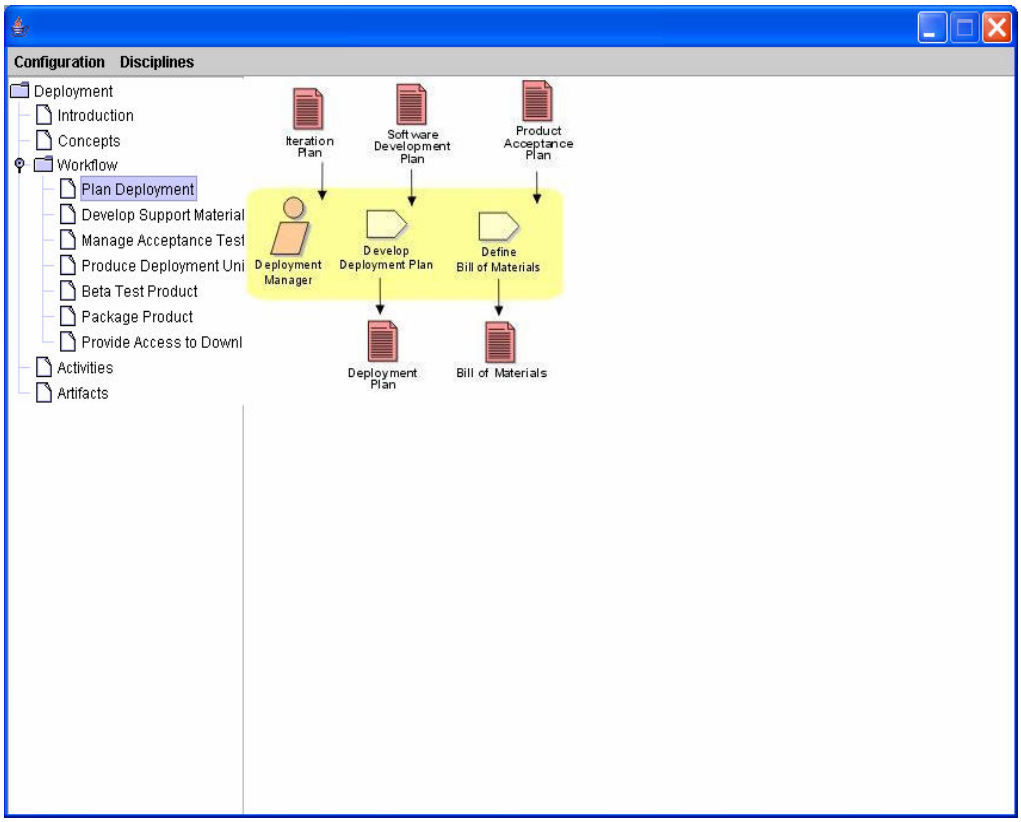

Figure 4.47: Deployment – Workflow – Plan Deployment

Deployment planning requires a high degree of customer collaboration and preparation. A successful conclusion to a software project can be severely impacted by factors outside the scope of software development such as the building, hardware infrastructure not being in place, and the staff being ill-prepared for cut-over to the new system.

To ensure successful deployment, and transition to the new system and ways of doing business, the Deployment Plan needs to address not only the deliverable software, but also the development of training material and system support material to ensure that end users can successfully use the delivered software product.

A deployment manager needs to be someone who is aware of the operational needs of the end user and capable of pulling together all the items that go into making the product. The deployment manager runs the beta test and, in the case of "shrink wrap" products, deals with the manufacturers to ensure that adequate quality is achieved in the product.

The deployment manager "gets the product out there" and, as such, needs to be well versed in the required infrastructure, and user needs, to ensure that the product is available for the users.

In Figure 4.48, "Develop Support Material" workflow details are shown.

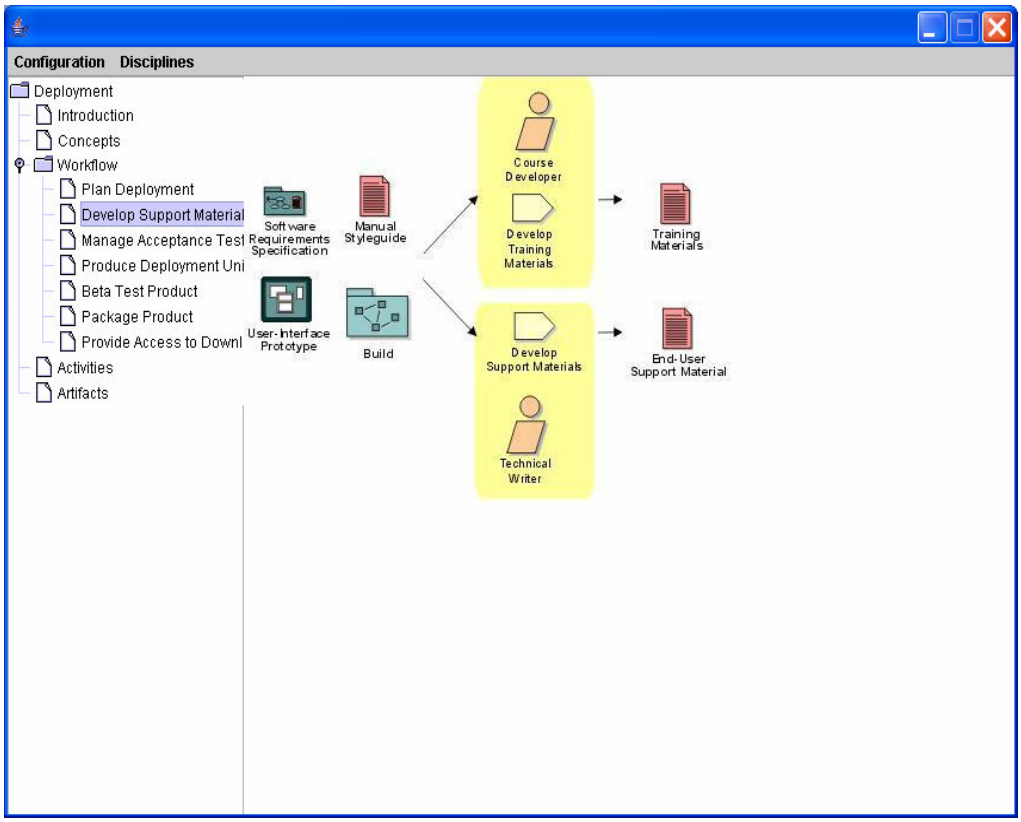

Figure 4.48: Deployment – Workflow – Develop Support Material

Support material covers the full range of information that will be required by the end-user to install, operate, use, and maintain the delivered system. It also includes training material for all of the various positions that will be required to effectively use the new system.

Both the Technical Writer and Course Developer need to be articulate and adept at creating information, written or otherwise, that is relevant from an end-user perspective.

In Figure 4.49, "Manage Acceptance Test" workflow details are shown.

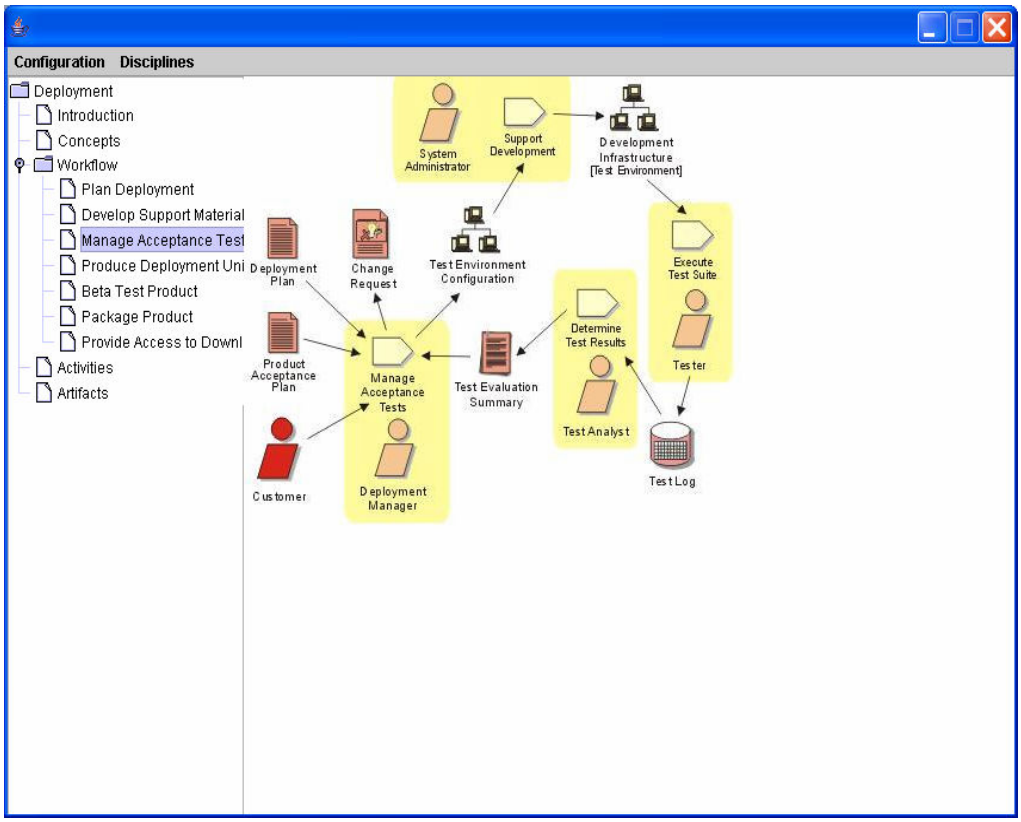

Figure 4.49: Deployment – Workflow – Manage Acceptance Test

The Deployment Manager organizes the installation of the product on one or more Test Environment Configurations that represents an environment acceptable to the customer as specified in the Product Acceptance Plan. In some cases, this environment will actually be the production deployment environment itself.

In some cases, the installation process itself may involve be subject to an acceptance test, as may any preceding hardware upgrades and configurations.

Once installed, the Tester typically runs through a preselected set of tests-usually based on a selected subset of the existing Test Suites-and determines the Test Results. The Deployment Manager and other stakeholders review the Test Results for anomalies. If there are "show stoppers", the Deployment Manager raises Change Requests that require immediate attention and resolution, and may delay or postpone subsequent plans for deployment to a wider user base.

In Figure 4.50, "Produce Deployment Unit" workflow details are shown.

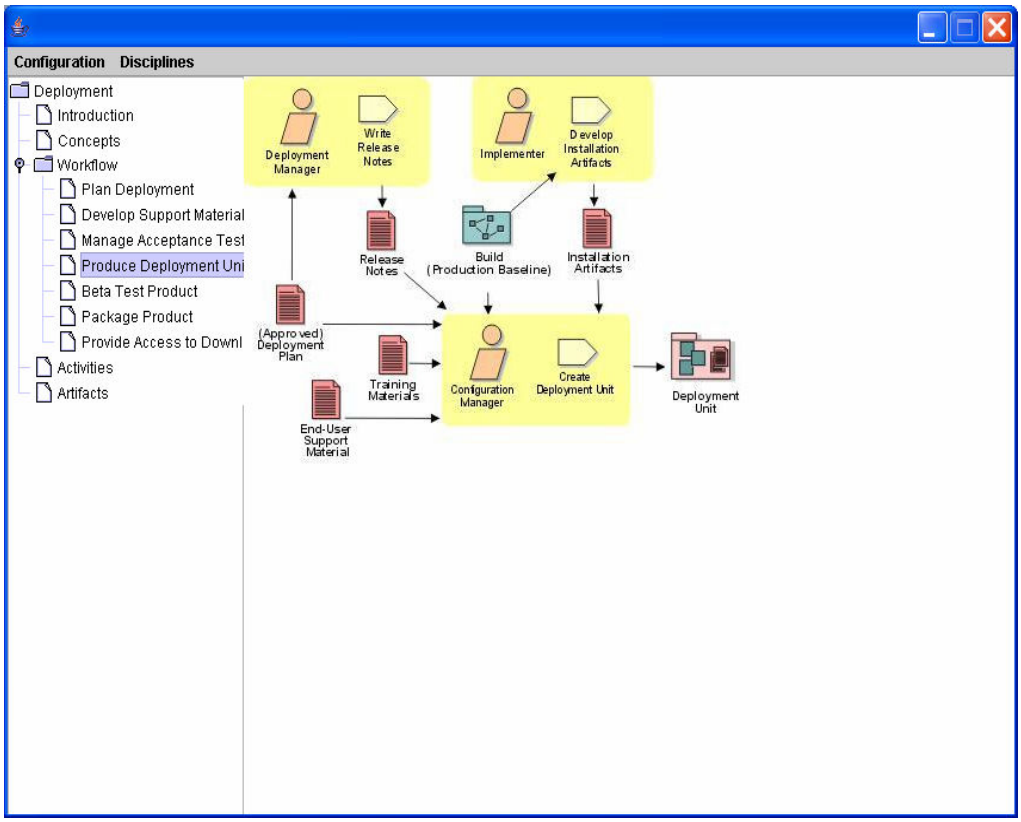

Figure 4.50: Deployment – Workflow – Produce Deployment Unit

The purpose of this workflow detail is to:

- Create a deployment unit that consists of the software, and the necessary accompanying artifacts required to effectively install and use it.
- The deployment unit can be created for the purposes of beta testing a test deployment to the final users or, depending on it level of maturity, for the final product.

This workflow detail relies on the skill set of described roles to create the product, installation scripts, and associated user support material, in a form that can be effectively delivered to the end users.

In Figure 4.51, "Beta Test Product" workflow details are shown.

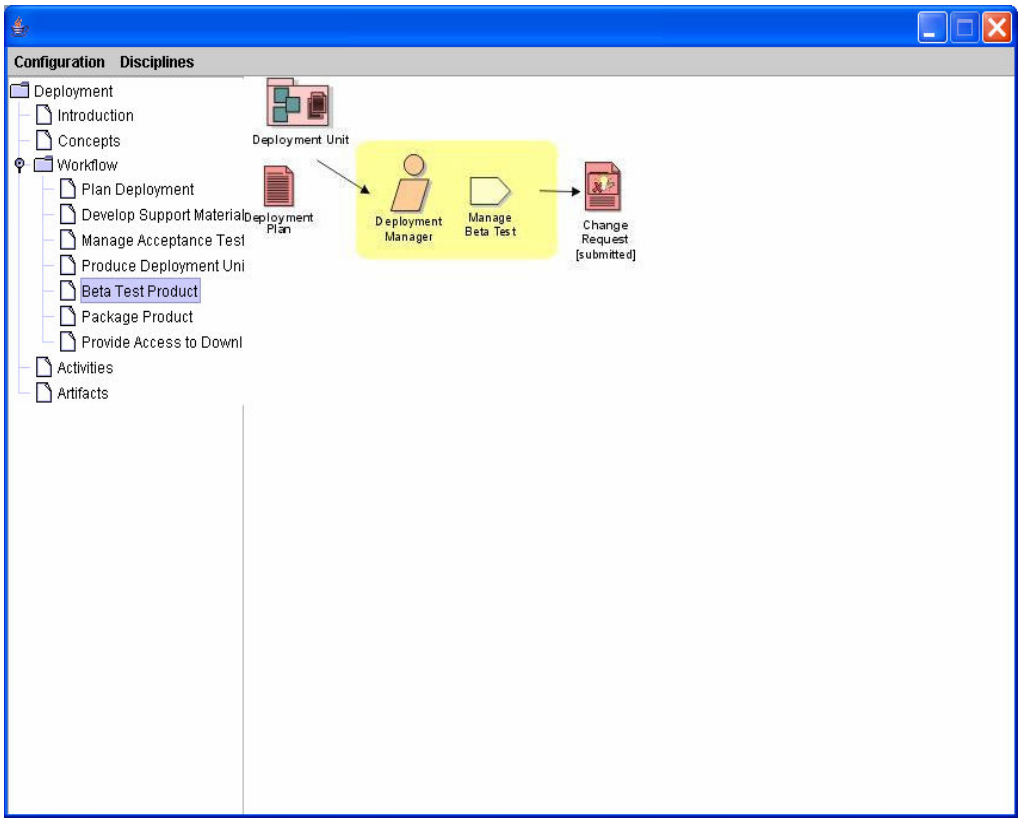

Figure 4.51: Deployment – Workflow - Beta Test Product

Feedback from the Beta Program is treated as Stakeholder Requests and factored into the developing product features in subsequent iterations.

In Figure 4.52, "Package Product" workflow details are shown.

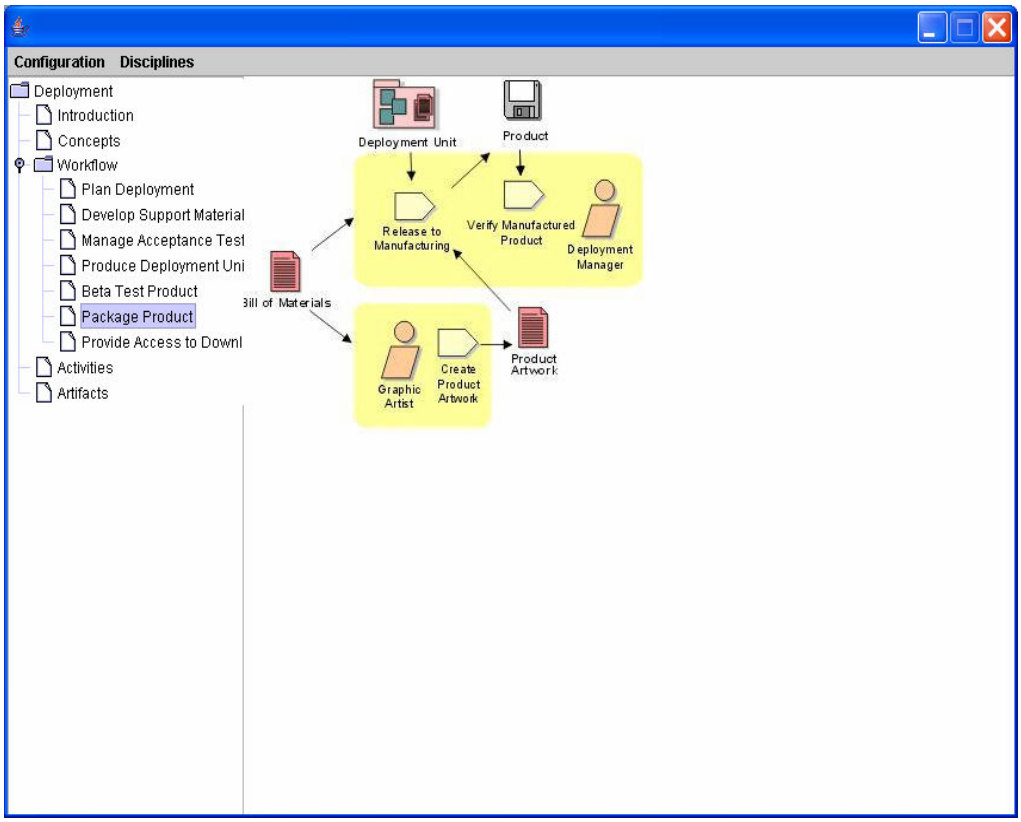

Figure 4.52: Deployment – Workflow – Package Product

The idea is to take the deployment unit, installation scripts, and user manuals, then package them for mass production-as in a consumer product.

Apart from the software logistics people like the Deployment Manager, this workflow detail calls for the product image-makers such as the technical "copy" writers and graphic artists to lend their talents to add to the product's visual appeal as it competes for consumer attention. Also required is handing off of the product to manufacturing, who will produce the product in massive quantities.

In Figure 4.53, "Provide Access to Download Site" workflow details are shown.

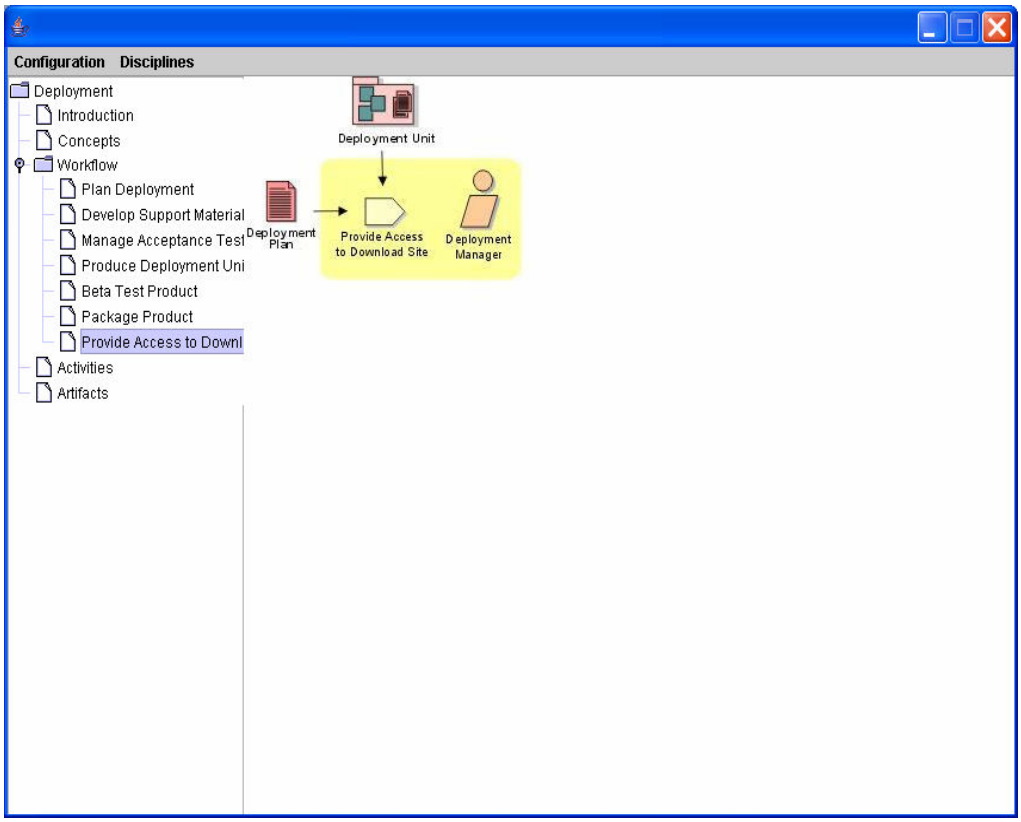

Figure 4.53: Deployment – Workflow – Provide Access to Download Site

The appeal of the Internet as a software distribution channel is obvious. The product is entirely accessible through the software environment via browsers and web-sites. The challenge for the provider is to make sure the product is reliably available at all times to a global marketplace, even through varying that could choke the host hardware and communication bandwidths.

Setting up the hardware infrastructure to host the corporate web presence is beyond the scope of a software development process. However, the deployment manager needs to know how to add the product offering to the list of products available over the web and that the product is available for purchase and delivery on demand.

Activities:

In Figure 4.54, acitivites of the "Deployment" discipline are shown. The details are explained in the workflow details meuitem. This view is added to the menuitems to list the roles's responsibilities in a clear way.

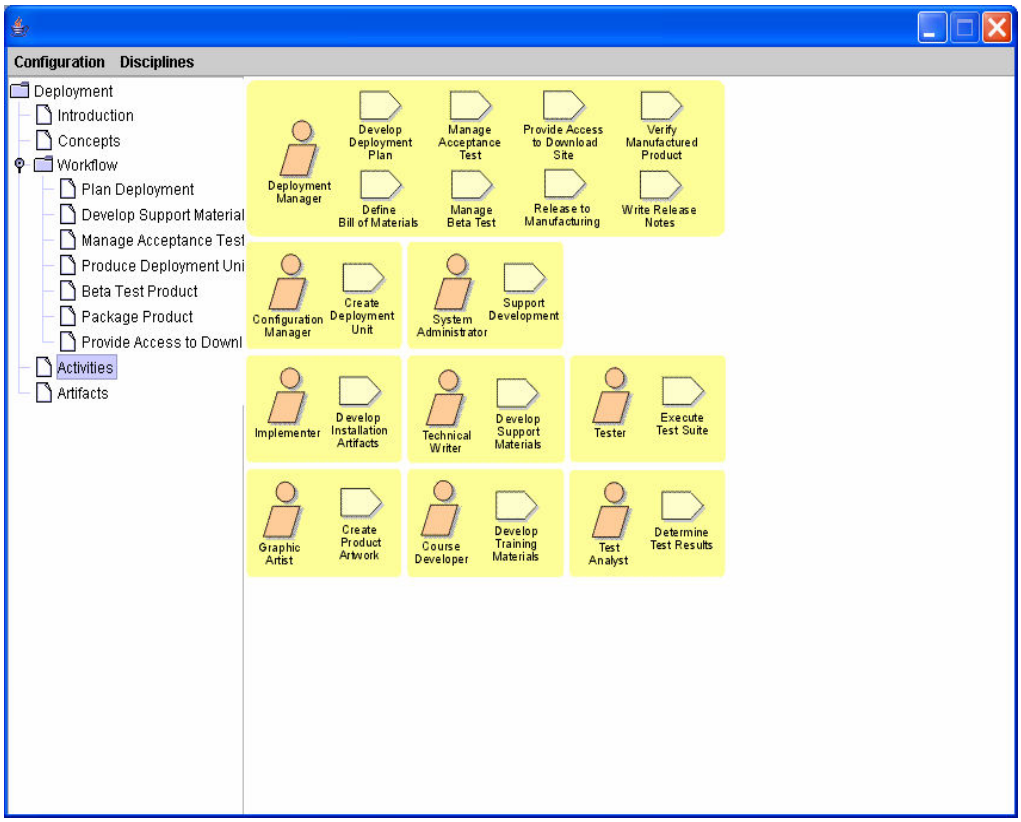

Figure 4.54: Deployment – Activities

Artifacts:

In Figure 4.55, artifacts of the "Deployment" discipline are shown. The workflow details are explained in the workflow details meuitem. This view is added to the menuitems to list the produced artifacts.

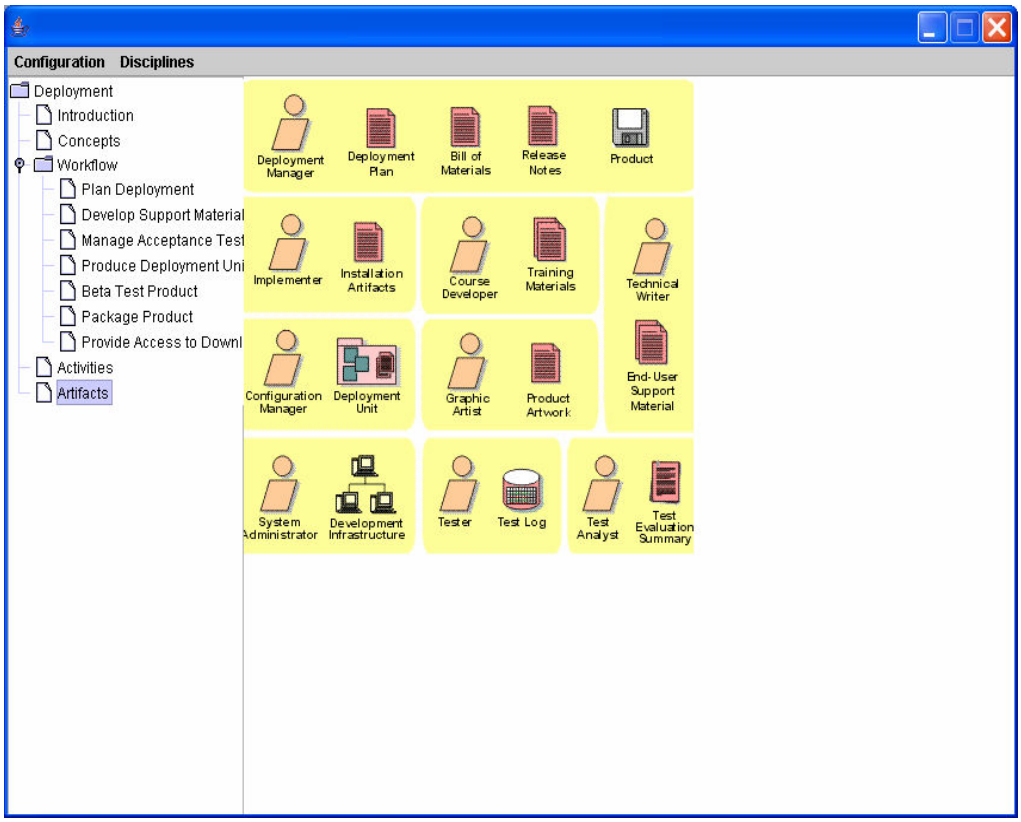

Figure 4.55: Deployment – Artifacts

4.1.3.6 Configuration Management:

Introduction:

In Figure 4.56 “Change Management” discipline purpose are explained.

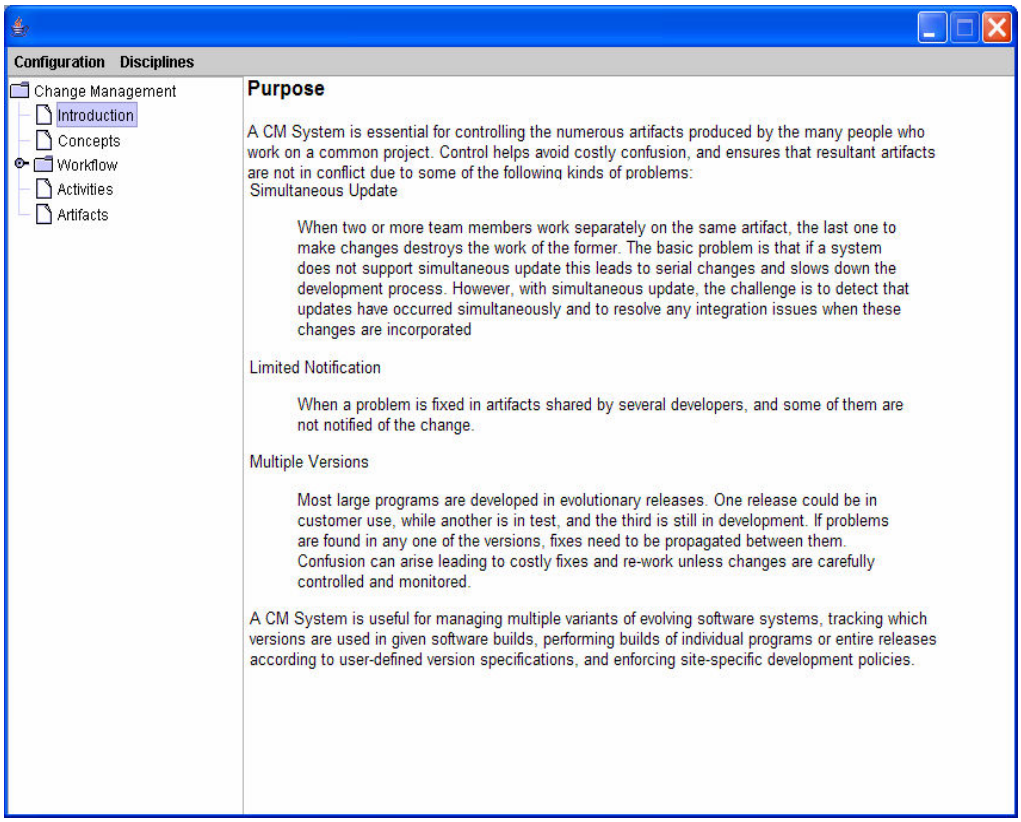

Figure 4.56: Change Management - Introduction

Change Management(CM) involves:

- identifying configuration items,
- restricting changes to those items,
- auditing changes made to those items, and
- defining and managing configurations of those items.

The methods, processes, and tools used to provide change and configuration management for an organization can be considered as the organization's CM System.

An organization's CM System holds key information about its product development, promotion , deployment and maintenance processes, and retains the asset base of potentially re-usable artifacts resulting from the execution of these processes.

The CM System is an essential and integral part of the overall development processes.

A CM System is essential for controlling the numerous artifacts produced by the many people who work on a common project. Control helps avoid costly confusion, and ensures that resultant artifacts are not in conflict due to some of the following kinds of problems [35]:

- Simultaneous Update: When two or more team members work separately on the same artifact, the last one to make changes destroys the work of the former. The basic problem is that if a system does not support simultaneous update this leads to serial changes and slows down the development process. However, with simultaneous update, the challenge is to detect that updates have occurred simultaneously and to resolve any integration issues when these changes are incorporated

- Limited Notification: When a problem is fixed in artifacts shared by several developers, and some of them are not notified of the change.

- Multiple Versions: Most large programs are developed in evolutionary releases. One release could be in customer use, while another is in test, and the third is still in development. If problems are found in any one of the versions, fixes need to be propagated between them. Confusion can arise leading to costly fixes and re-work unless changes are carefully controlled and monitored.

A CM System is useful for managing multiple variants of evolving software systems, tracking which versions are used in given software builds, performing builds of individual programs or entire releases according to user-defined version specifications, and enforcing site-specific development policies.

Some of the direct benefits provided by a CM System are that it:

- supports development methods,

- maintains product integrity,

- ensures completeness and correctness of the configured product,
- provides a stable environment within which to develop the product,
- restricts changes to artifacts based on project policies, and
- provides an audit trail on why, when and by whom any artifact was changed.

In addition, a CM System stores detailed 'accounting' data on the development process itself: who created a particular version (and when, and why), what versions of sources went into a particular build, and other relevant information.

Concepts:

In Figure 4.57, "Change Management" concepts are shown.

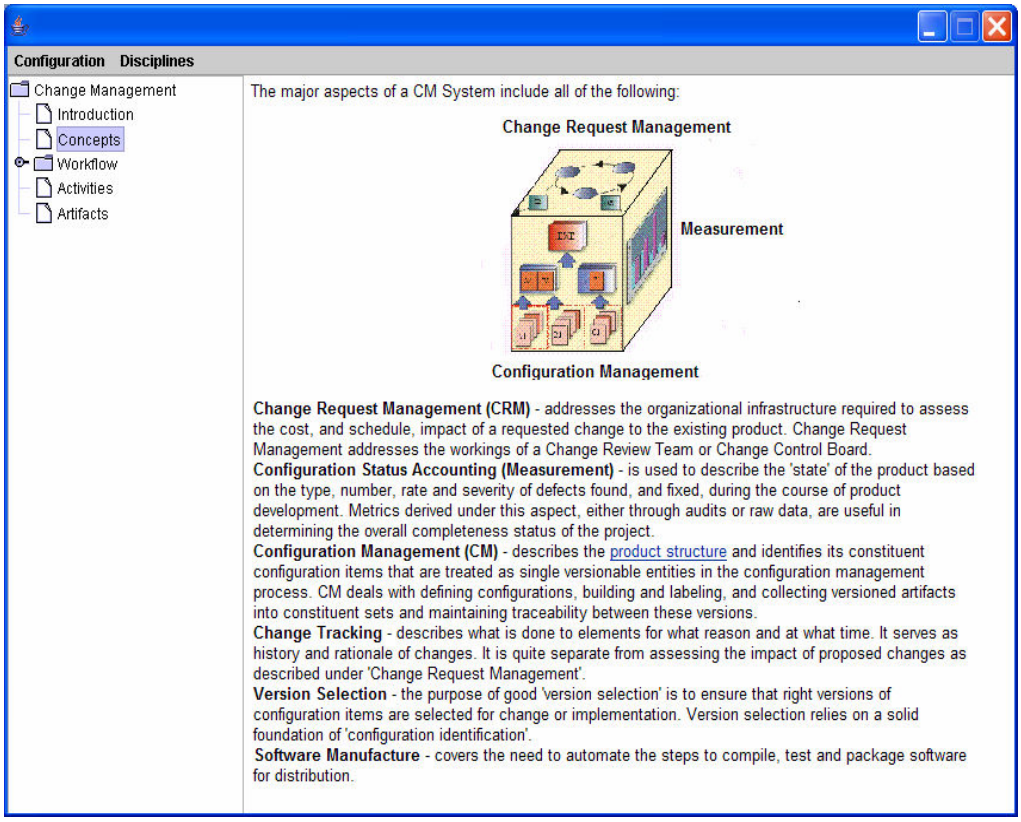

Figure 4.57: Change Management – Concepts

Workflow:

The main flowchart of this discipline is shown on this menu item. There are two ways to see the details. Project members could work on the flowchart by clicking on the activities listed on the screen or selecting from the submenu.

In Figure 4.58, "Change Management" workflow is shown.

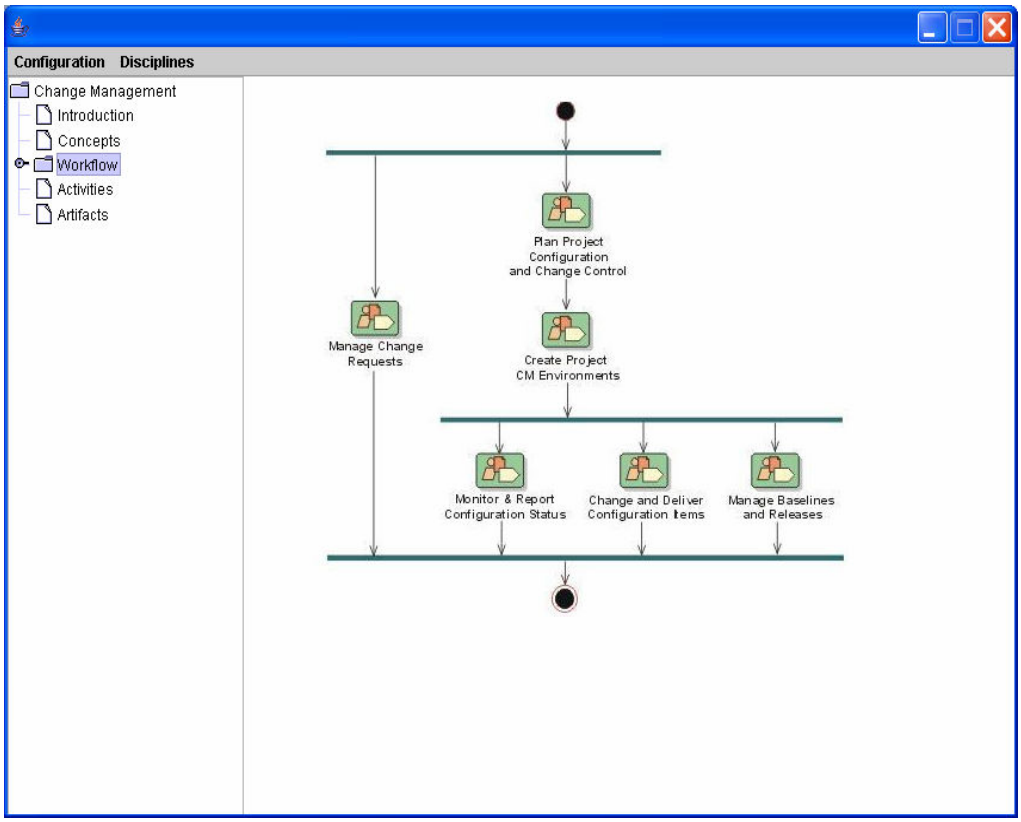

Figure 4.58: Change Management– Workflow

In Figure 4.59, "Plan Project Configuration & Change Control" workflow details are shown.

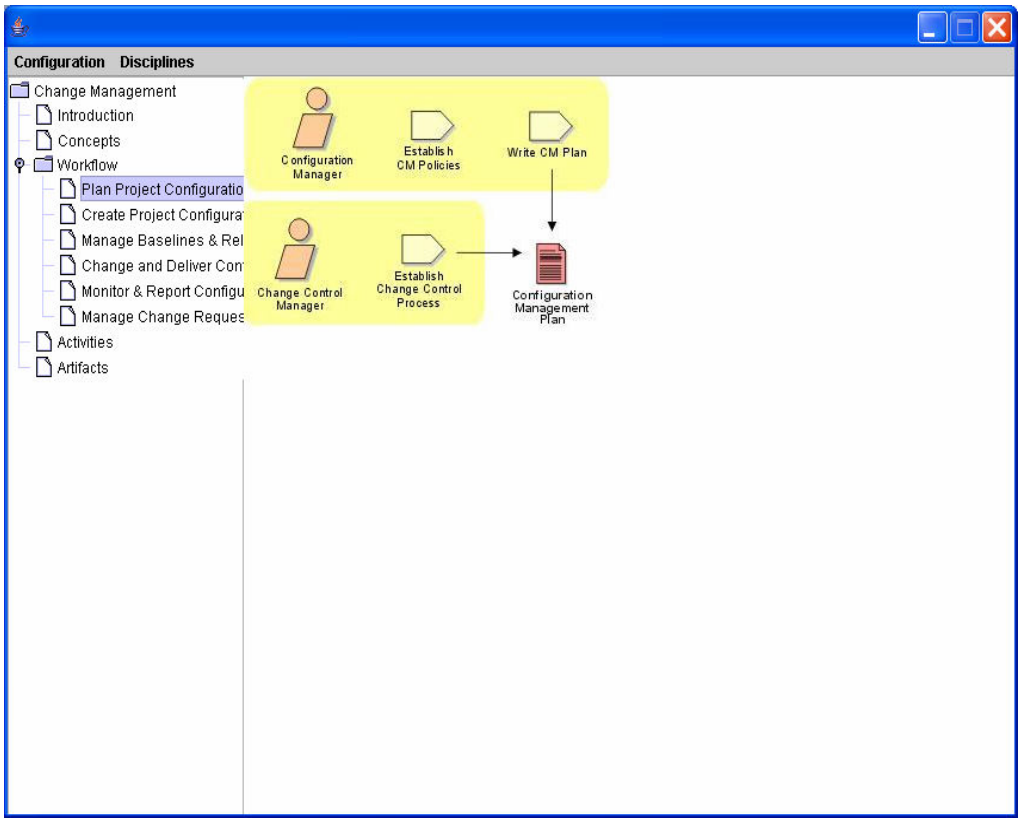

Figure 4.59: Change Management– Workflow – Plan Project Configuration

The workflow detail focuses on:

- Establishing project configuration management policies
- Establishing policies and processes for controlling product change
- Documenting this information in the configuration management plan

"Configuration Management" policies refer to the ability to identify and report on the artifacts that have been approved for use in a project. Identification is simplified and enabled through the use of proper tools to control project artifacts, and the systematic labeling of those artifacts over time to identify their relative maturity and their relationships with each other at given points in time. Systematic identification practices are a key enabler for the safeguarding of project artifacts through archiving and baselining techniques.

Standard, documented change control processes help to ensure that changes are made within a project in a consistent manner, and the appropriate stakeholders are informed of the current state of the product, requested changes to it and the impact of these changes on cost, schedule and so forth.

The configuration management plan documents how product related activities are to be planned, implemented controlled and organized.

A person playing the configuration manager role needs to be organized by nature, yet flexible enough to plan configuration and change control to suit the needs of the project team. The configuration manager role supports the team by ensuring that the project change policies are reflected within the projects change management tools, enabling software developers to easily transition artifacts through state changes in accordance with the defined development and approval practices. The configuration manager role is required to put measures in place to monitor that the CM Plan is being followed as intended, that audit reporting is occurring on a regular basis, and to work with the System Administrator role to ensure that backups of CM assets are in safekeeping.

The change control manager is a key arbitration role. In this capacity, the decision for the inclusion of any given change in a software build is ultimately made by the change control manager on a project. In practice, only those changes of significant potential impact typically warrant monitoring, and any potential impact on the inclusion-or exclusion-of changes to the product should be carefully considered with regard to project factors such as the political climate, the need to establish trust between developer and customer and so forth.

In Figure 4.60, "Create Project Configuration Management (CM) Environments" workflow details are shown.

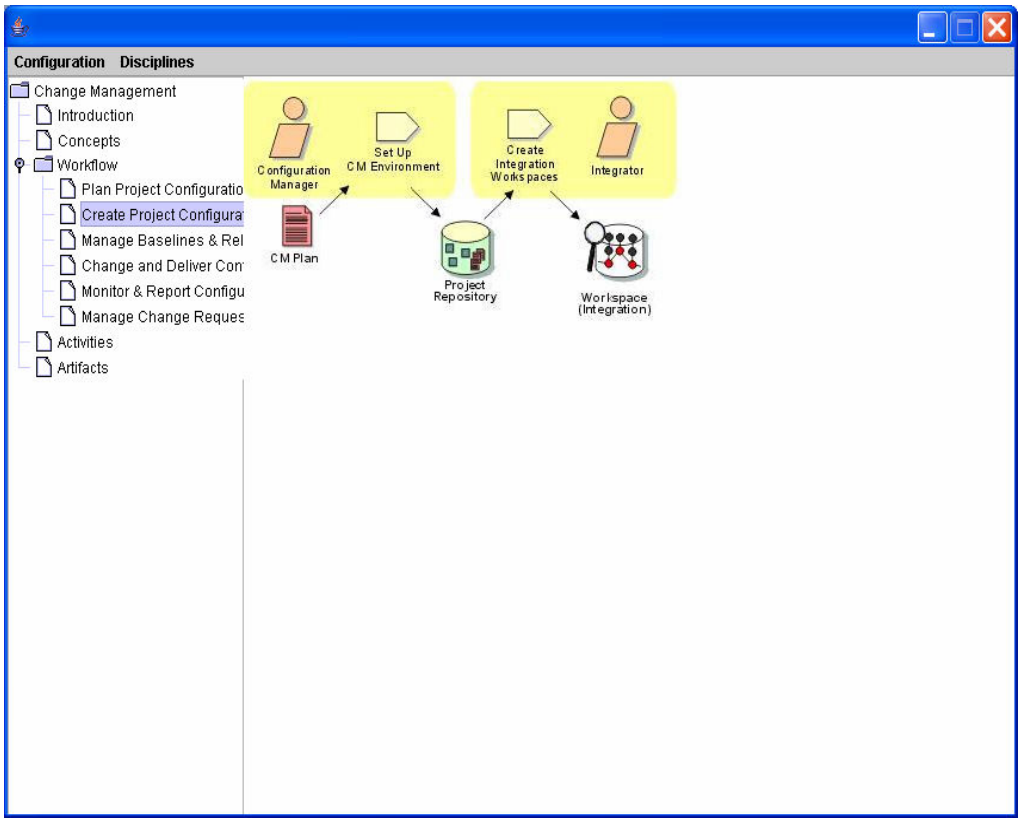

Figure 4.60: Change Management– Workflow – Create Project Configuration

This work is done by making sure the essential artifacts are available to developers and integrators in the various private and public workspaces as they need them, and then are adequately baselined and stored for subsequent use. Setting up the CM environment involves creating the product directory structure, repositories, workspaces (developer and integration) and allocating machine resources (servers and disk space).

To set up an appropriate environment, a person playing the configuration manager role needs to have a good understanding of the component structures of the overall product, and will need to work closely with the software architect to ensure that adequate "place holder" CM items are established.

A person playing the integrator role in this work needs to ensure that artifacts delivered from the developer workspaces are sufficiently tested such that they can be incorporated into a testable build. The integrator role needs to be familiar with project configuration management policies, build and test practices.

In Figure 4.61, "Manage Baselines & Releases" workflow details are shown.

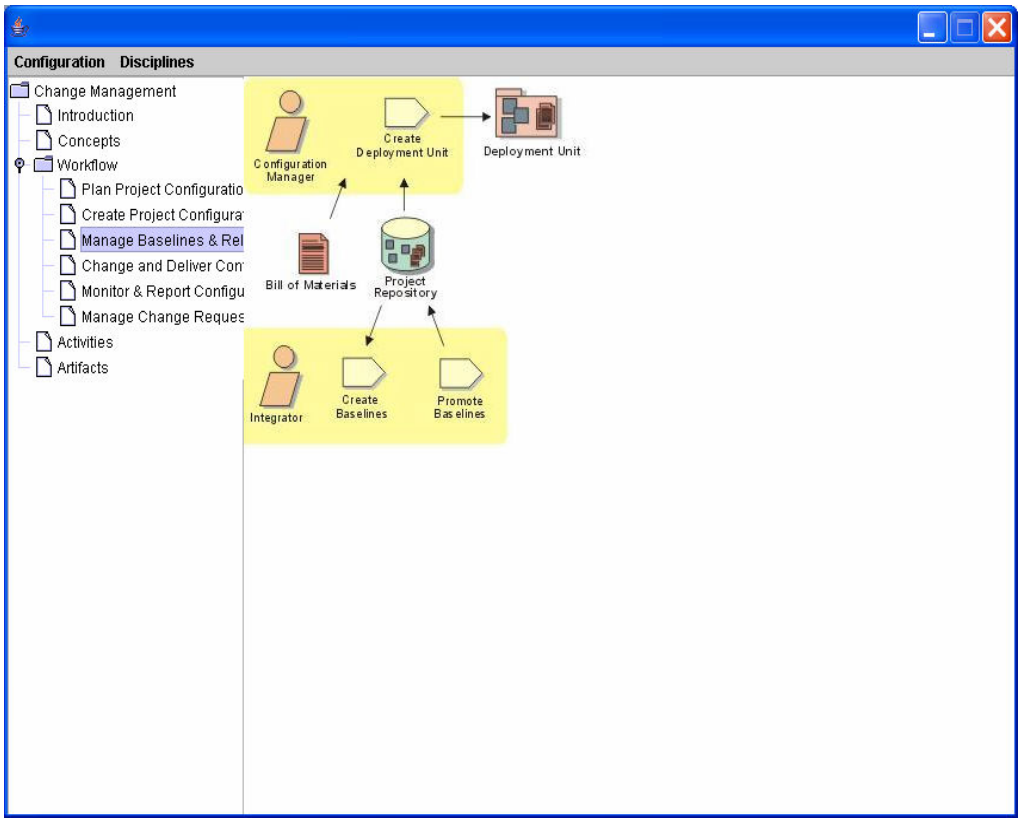

Figure 4.61: Change Management– Workflow – Manage Baselines and Releases

The frequency and formality in which baselines are created are described in the CM Plan. The degree of formality is clearly much higher for a product being released to a customer than for executable releases within the internal project team. When the combined set of artifacts reach certain stages or levels of maturity, baselines are created to assist managing availability for release, reuse and so forth.

This work is primarily driven by the configuration manager role, where the typical need is to be able to assemble a product for release. The released product requires a Bill of Materials (BOM) that serves as a complete checklist of what is to be delivered to the customer. The released product will inevitably require release notes and training material as described in the deployment activities.

The integrator role contributes to this work by ensuring that artifacts delivered from the developer workspaces are integrated such that they can be incorporated into a independently testable build.

In Figure 4.62, "Change and Deliver Configuration Items" workflow details are shown.

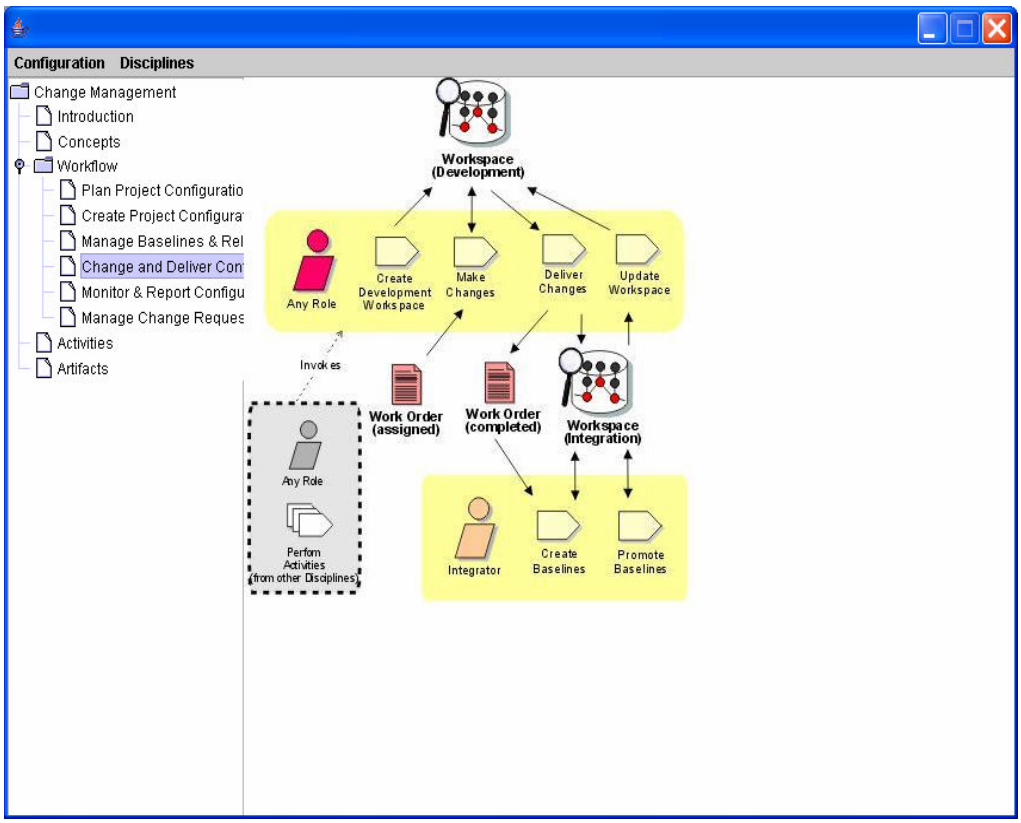

Figure 4.62: Change Management– Workflow – Change and Deliver Configuration Items

This workflow detail is focused on:

- The creation of workspaces, accessing project artifacts, making changes to those artifacts, delivering the changes for inclusion in the overall product, by any role in the project team.
- The building of the product, creation of baselines and promotion of the baselines for availability to the rest of the development team.

In Figure 4.63, "Monitor & Report Configuration Status" workflow details are shown.

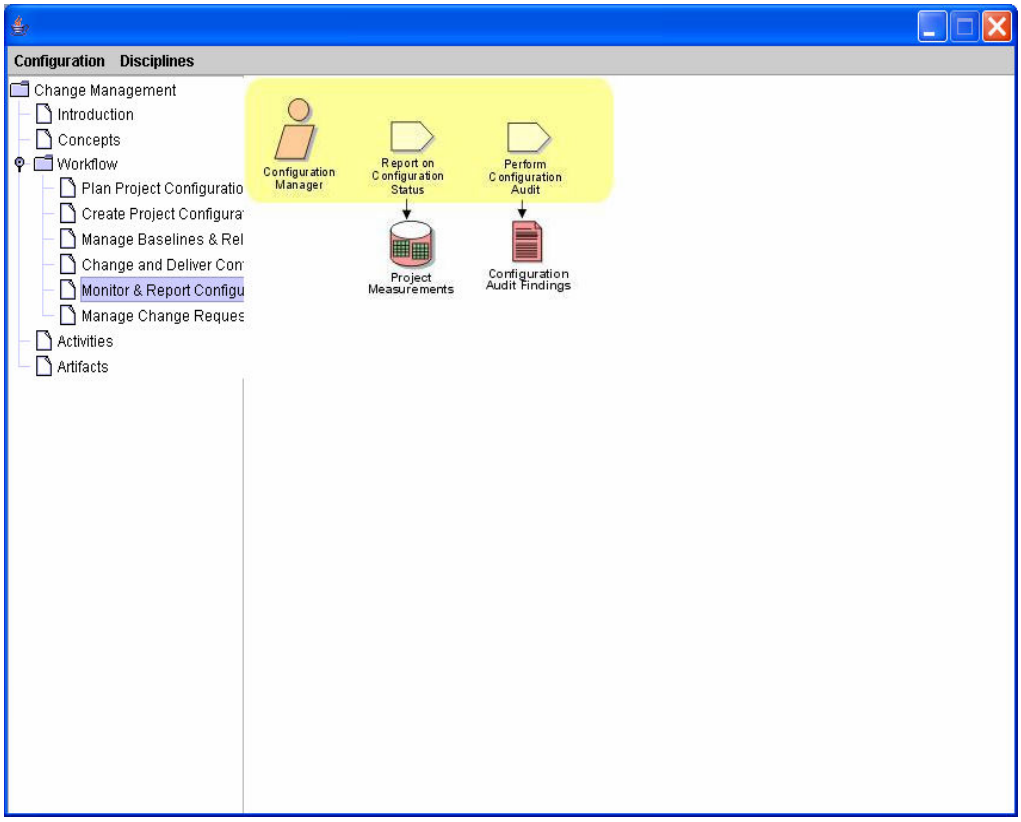

Figure 4.63: Change Management– Workflow – Monitor & Report Configuration

This workflow detail is focused on:

- Ensuring that artifacts and their associated baselines are available.
- Determining if required artifacts are stored in a controlled library and baselined.
- Supporting project Configuration Status Accounting activities.
- Facilitating reporting of change request information, especially the activities related to work performed against defect and enhancement requests.
- Ensuring that data is "rolled-up" and reported for the purposes of tracking progress and trends.

In Figure 4.64, "Manage Change Requests" workflow details are shown.

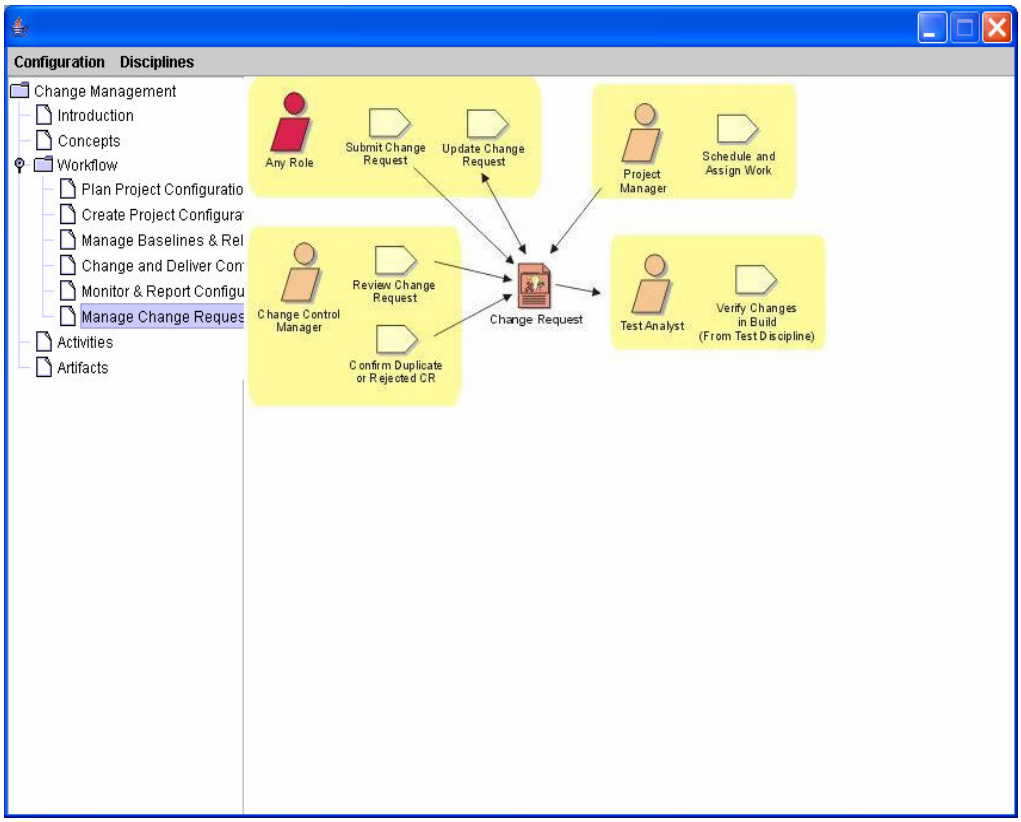

Figure 4.64: Change Management– Workflow – Manage Change Request

Having a standard, documented change control process ensures that changes are made within a project in a consistent manner and the appropriate stakeholders are informed of the state of the product, changes to it and the cost and schedule impact of these changes.

Activities:

In Figure 4.65, activites of the "Change Management" discipline are shown. The details are explained in the workflow details menu item. This view is added to the menuitems to list the roles's responsibilities in a clear way.

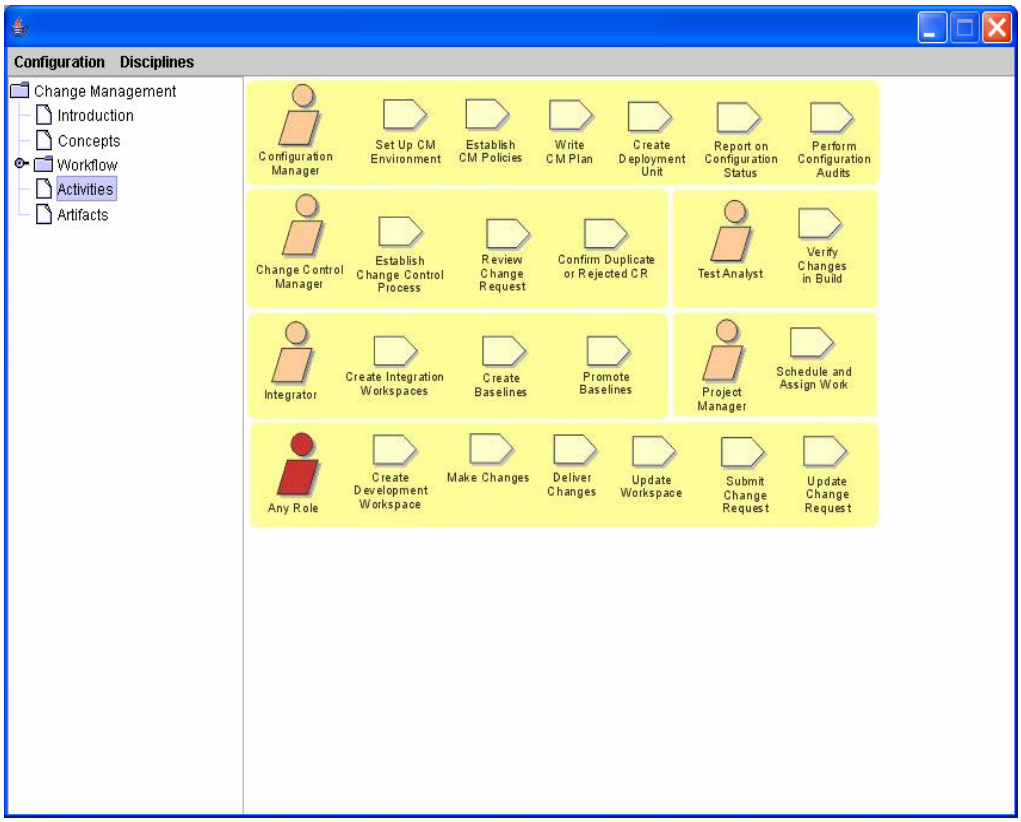

Figure 4.65: Change Managemen - Activities

Artifacts:

In Figure 4.66, artifacts of the "Change Management" discipline are shown. The workflow details are explained in the workflow details menu item. This view is added to the menu items to list the produced artifacts.

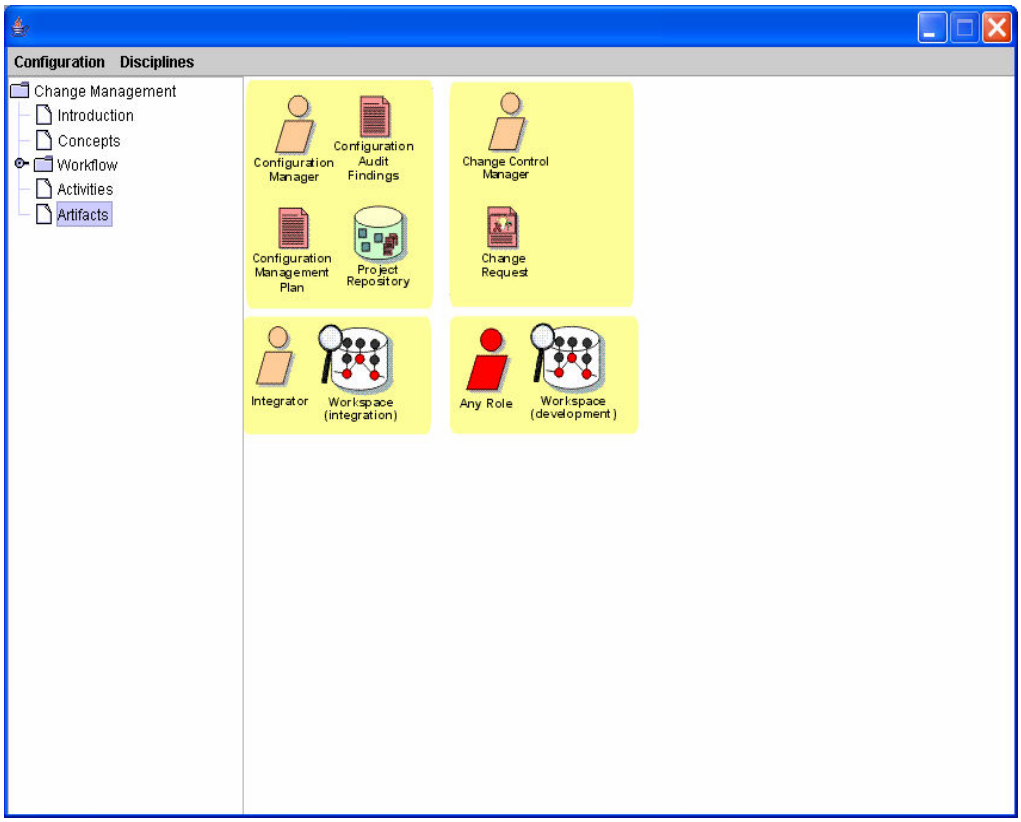

Figure 4.66: Change Management– Artifacts

4.1.3.7 Project Management:

Introduction:

In Figure 4.67 “Project Management” discipline purpose is explained.

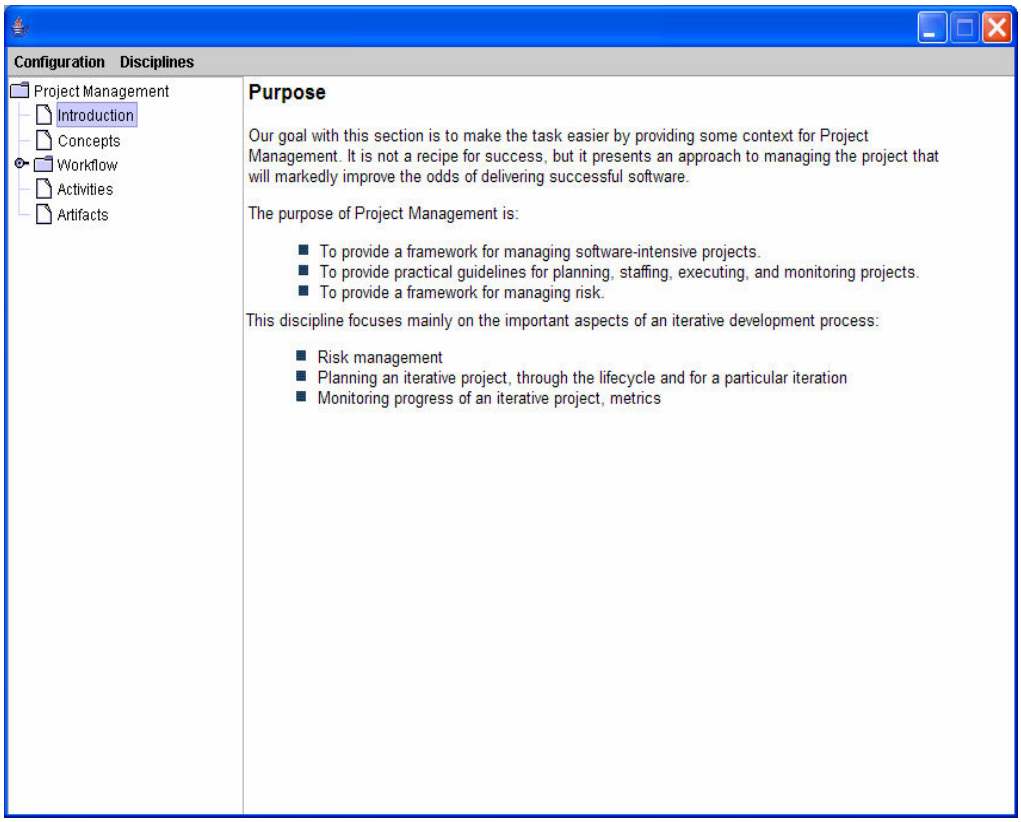

Figure 4.67: Project Management - Introduction

Software project management is the art of balancing competing objectives, managing risk, and overcoming constraints to successfully deliver a product which meets the needs of both customers (the payers of bills) and the users. The fact that so few projects are unarguably successful is comment enough on the difficulty of the task.

Our goal with this section is to make the task easier by providing some context for project management. It is not a recipe for success, but it presents an approach to managing the project that will markedly improve the odds of delivering successful software.

The purpose of "Project Management" is:

- To provide a framework for managing software-intensive projects.
- To provide practical guidelines for planning, staffing, executing, and monitoring projects.

- To provide a framework for managing risk.

This discipline focuses mainly on the important aspects of an iterative development process:

- Risk management
- Planning an iterative project, through the lifecycle and for a particular iteration
- Monitoring progress of an iterative project, metrics

Concepts:

In Figure 4.68, "Project Management" concepts are shown.

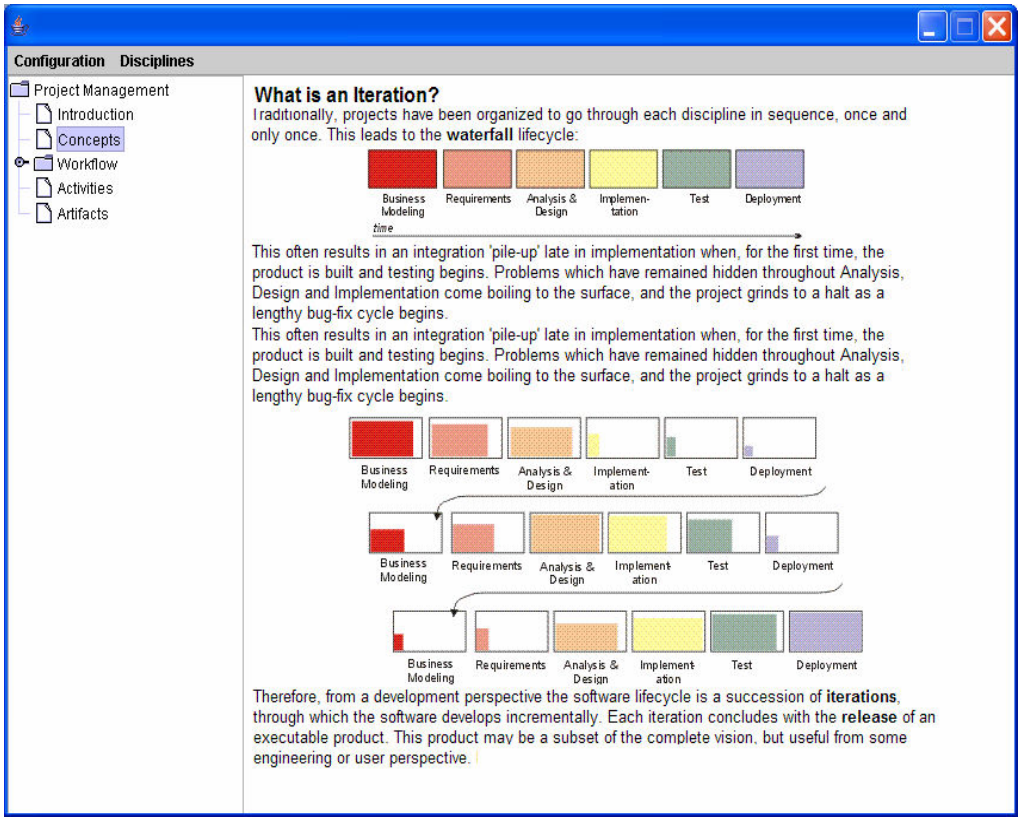

Figure 4.68: Project Management - Concepts

Workflow:

In Figure 4.69, "Project Management" workflow is shown.

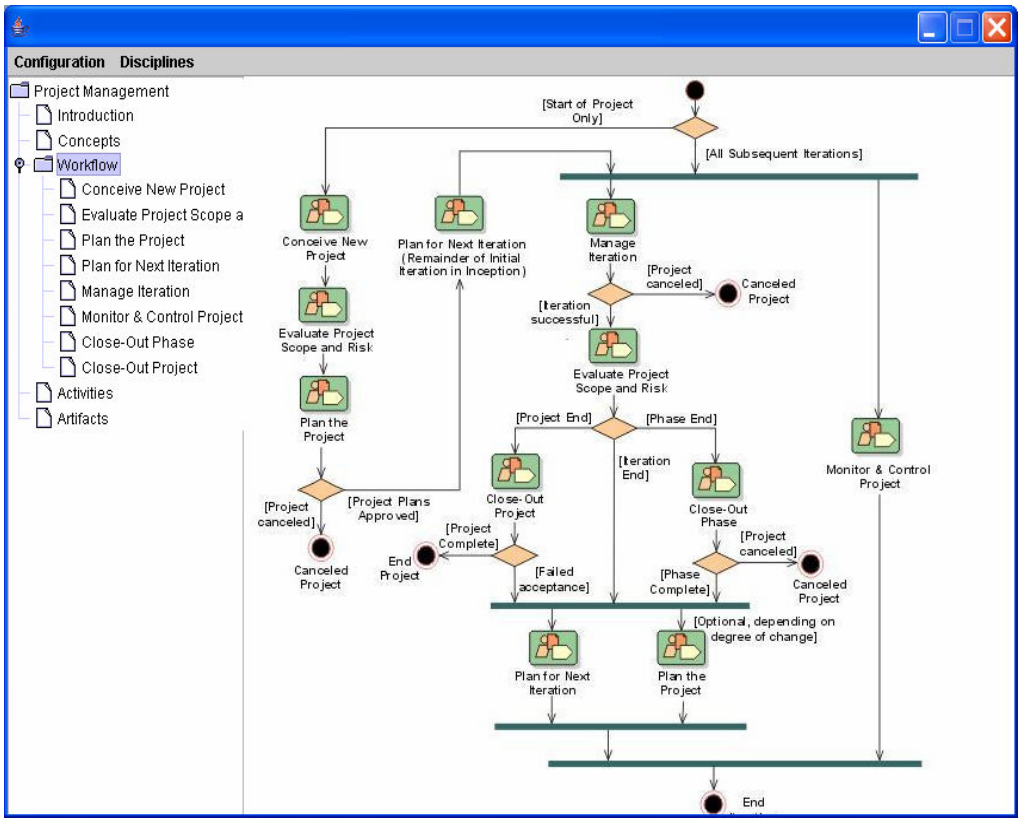

Figure 4.69: Project Management– Workflow

In Figure 4.70, "Conceive New Projects" workflow details are shown.

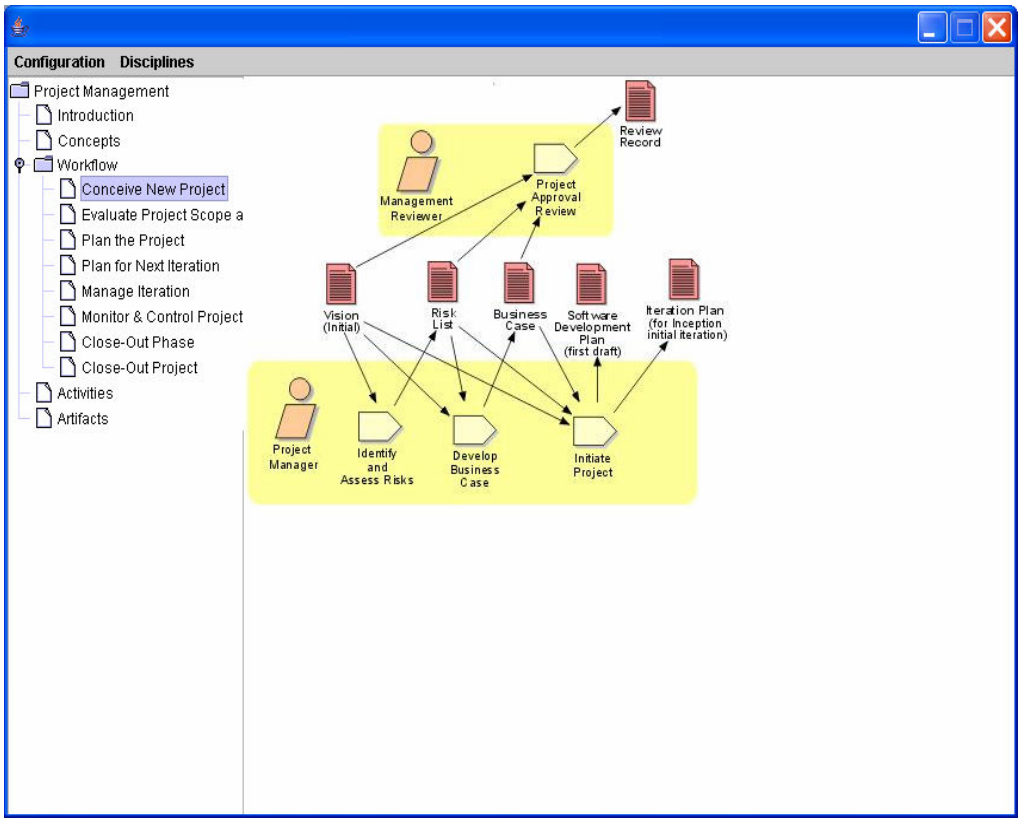

Figure 4.70: Project Management– Workflow – Conceive New Project

On the basis of the initial vision, risks are assessed and an economic analysis, the business case, is produced. If the "Activity:Project Approval Review" finds these satisfactory, the project is formally set up (in Activity: Initiate Project), and given limited sanction (and budget) to begin a complete planning effort. This latter activity adds substance to the initial Vision, validates and refines it.

In the Business Case, the Project Manager should describe at least two approaches to realizing the Vision, and analyze these in terms of risk impact, and economic outcomes. During the "Activity:Project Approval Review", one of the offered choices will be selected, if the project is to continue. There is a considerable body of management knowledge and theory to assist the Project Manager and the Project Reviewer in risk and decision analysis, and it is valuable to have a few of the project management and review staff well versed in these techniques - especially if the project is large, unprecedented, complex or otherwise risky.

In Figure 4.71, "Evaluate Project Scope and Risk" workflow details are shown.

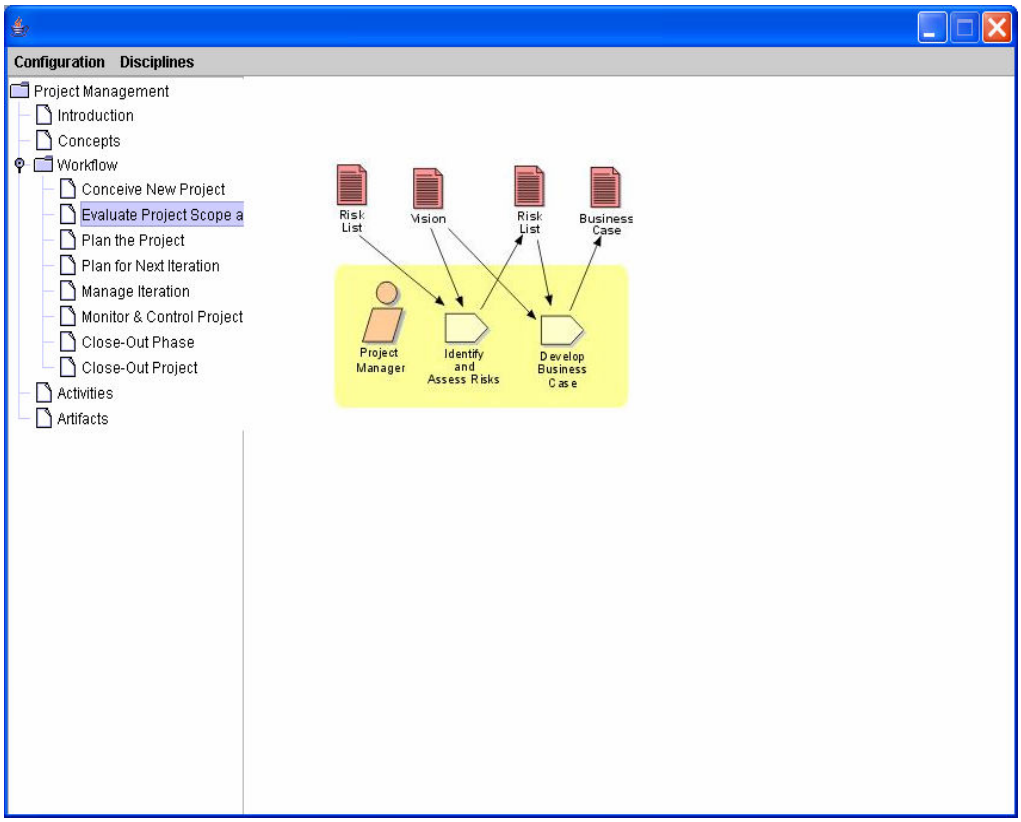

Figure 4.71: Project Management– Workflow – Evaluate Project Scope and Risk

The purpose of this workflow detail is to reappraise the project's intended capabilities and characteristics, and the risks associated with achieving them. As the capabilities and risks are better understood, the business case should be updated, to ensure that the project continues to be worth investing in, in its current form, or if a change in direction is needed.

This workflow detail updates and refines the Risk List and Business Case. Techniques such as those described in Workflow Detail: Conceive New Project: Guidelines may be used for risk and decision analysis. The Risk List and Business Case should be subjected to internal walkthroughs and reviews to ensure there is a general consensus, before the next round of detailed planning is begun.

In Figure 4.72, "Plan the Project" workflow details are shown.

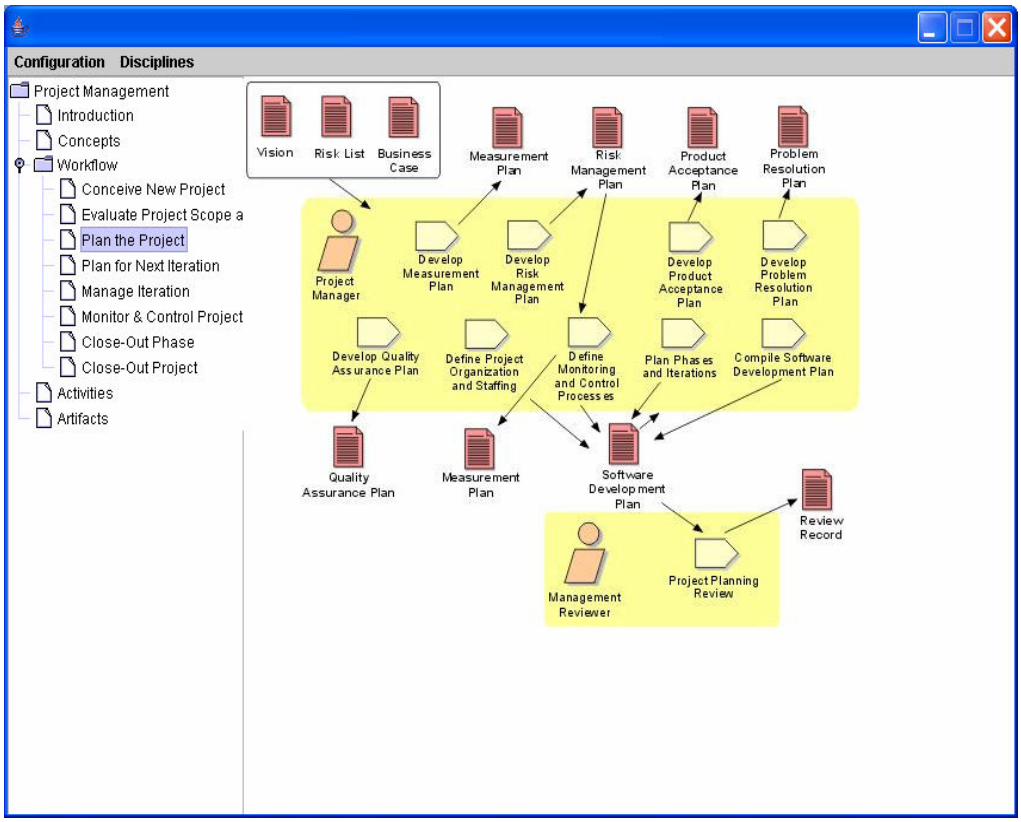

Figure 4.72: Project Management– Workflow – Plan the Project

The major effort in creating these artifacts comes early in the inception phase; thereafter, when this workflow detail is invoked at the beginning of each iteration, it is to revise the Software Development Plan (and its enclosures) on the basis of the previous iteration's experience and the Iteration Plan for the next. The Project Manager will also collate all other contributions to the Software Development Plan and assemble them in "Activity: Compile Software Development Plan".

Estimation should ideally be based in the organization's own experience, which is then used to calibrate an estimation model, such as COCOMO. If the Project Manager is starting from scratch, using default values for model coefficients, it will be important to use other methods to validate the estimates. Just as important is to obtain staff and other stakeholder agreement that the estimates are realistic and achievable. However, the Project Manager has to take into account the experience of staff giving feedback about estimates. More junior staff may be just guessing numbers and then adding large margins for error; conversely, their effort estimates may be naively low. The Project Manager must be circumspect when dealing with

estimates from junior staff, and be prepared to counsel them when necessary, and offer the assistance of a more experienced peer.

All enclosed plans and sections of the Software Development Plan should be evaluated through internal walkthroughs and reviews before the "Activity:Project Planning Review" occurs.

In Figure 4.73, "Plan for Nex Iteration" workflow details are shown.

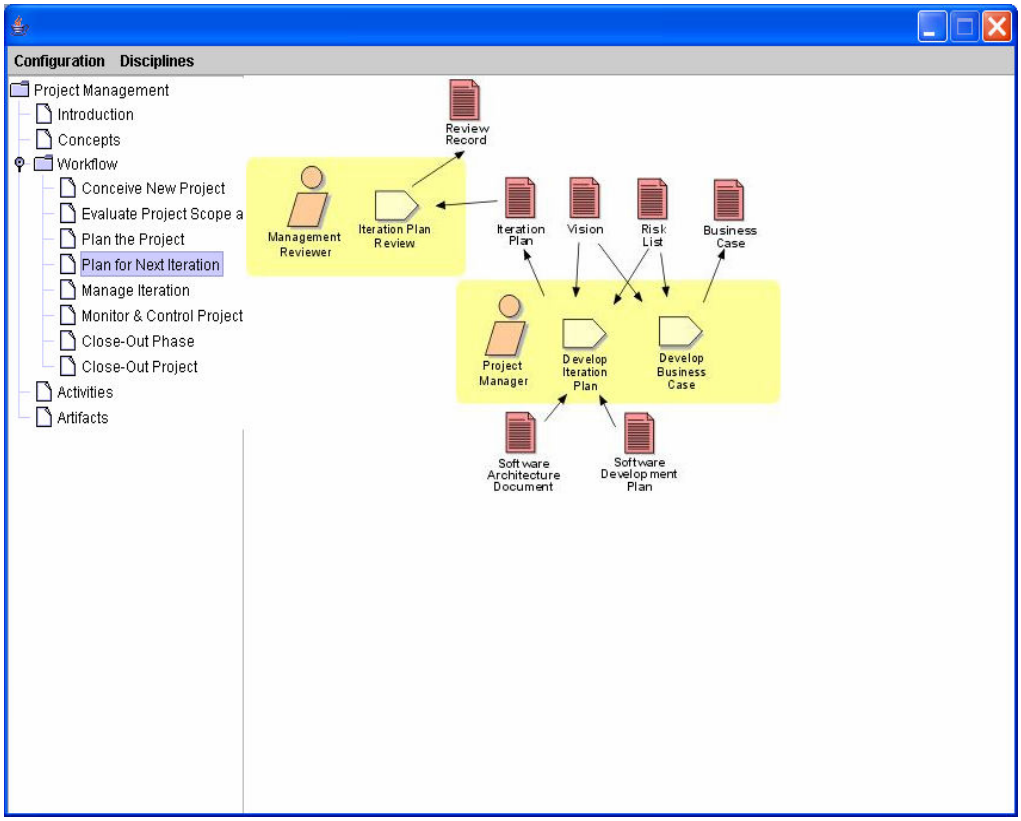

Figure 4.73: Project Management– Workflow – Plan for Next Iteration

The Iteration Plan should be reviewed by the customer and other stakeholders, and, if satisfactory, should be approved through the "Activity:Iteration Plan Review". This review also gives the customer visibility of the project's expectations of customer participation and resources-particularly if the iteration is intended to deliver artifacts or deploy software-so the customer can make appropriate plans.

The Project Manager should work closely with the Software Architect to define the iteration's contents. The Iteration Plan should be evaluated internally, through

walkthrough and review, before being presented for the "Activity:Iteration Plan Review", in particular:

- to assess the clarity of expression of the evaluation criteria for the iteration
- to reach agreement internally that the planned artifacts can be built with the effort and time available
- to ensure that the results of the iteration will be testable or otherwise demonstrable; that is, the iteration will have a tangible outcome

In Figure 4.74, "Manage Iteration" workflow details are shown.

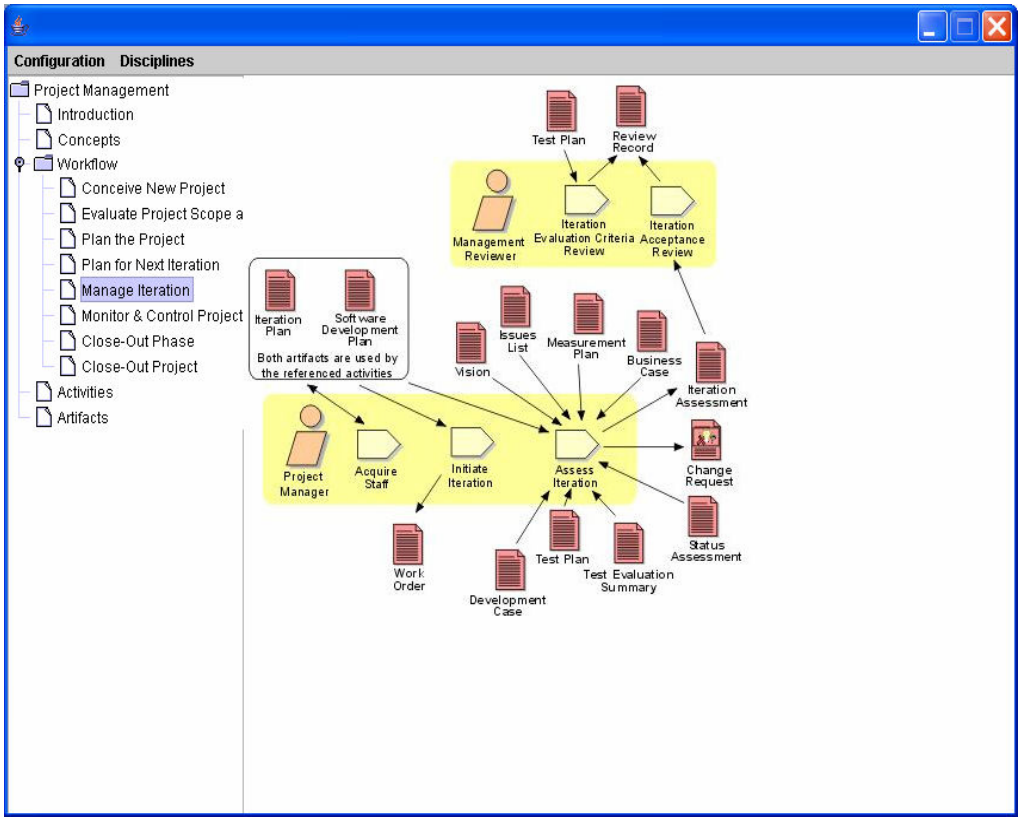

Figure 4.74: Project Management– Workflow – Manage Iteration

This workflow detail contains the activities that begin, end and review an iteration. The purpose is to acquire the necessary resources to perform the iteration (in "Activity: Acquire Staff" and "Activity: Initiate Iteration"), allocate the work to be done (in "Activity: Initiate Iteration"), and finally, to assess the results of the

iteration in “Activity: Assess Iteration”. An iteration concludes with an “Activity:Iteration Acceptance Review” which determines, from the “Activity:Iteration Assessment”, whether the objectives of the iteration were met.

Optionally, in a lengthy iteration, the project manager may think it prudent to resynchronize the expectations of management, technical staff, customer and other stakeholders, by holding an “Activity:Iteration Evaluation Criteria Review” mid-way through the iteration. At this review, which is based mainly around the test plan, the project reveals the planned contents of the iteration in a very concrete way. This gives an opportunity for a 'mid-course correction', should misunderstandings have arisen over the intent of the iteration plan.

The evaluation criteria for an iteration should have been set objectively and clearly, so the assessment of an iteration requires the project manager to be analytic and equally objective.

Failing the iteration on this count alone would not be sensible. Far better for the project manager and management reviewer to agree to relax this requirement, and as compensation, to add capability elsewhere. The management reviewer (and Pproject manager) need the experience and confidence to make these kinds of trades, which do not compromise the Vision for the product.

In figure 4.75, “Monitor and Control Project” workflow details are shown.

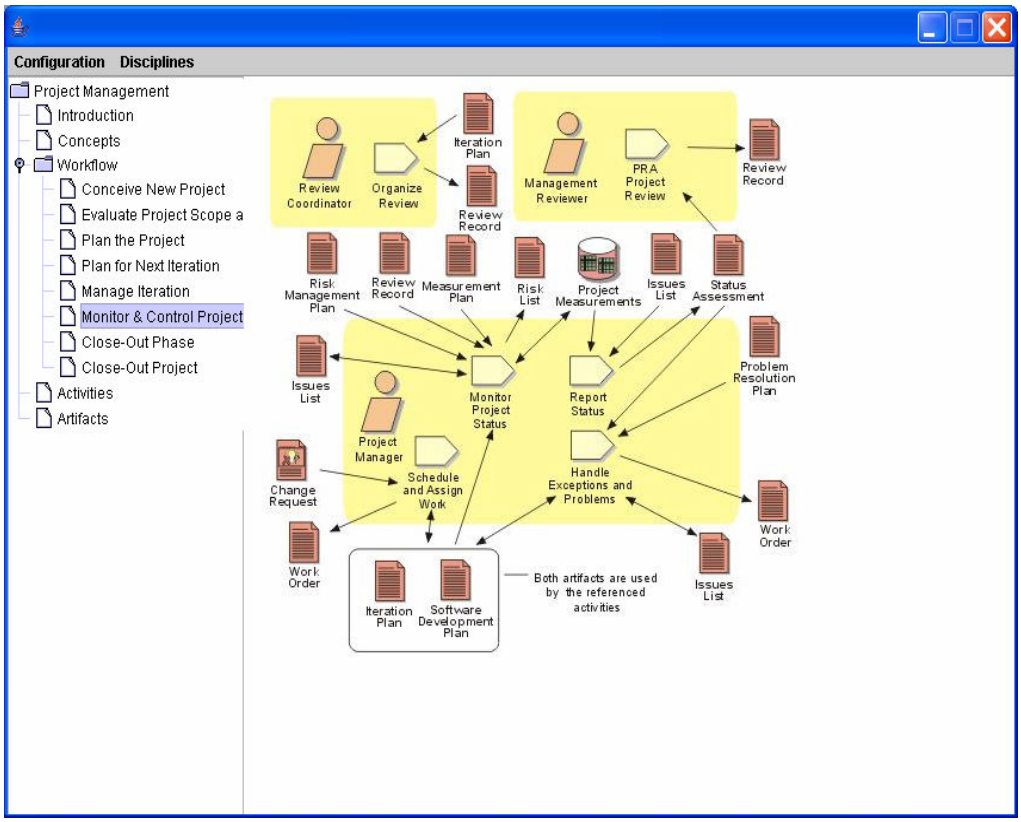

Figure 4.75: Project Management– Workflow – Monitor and Control Project

This workflow detail captures the daily, continuing, work of the project manager, covering:

- dealing with change requests that have been sanctioned by the change control manager, and scheduling these for the current or future iterations;
- continuously monitoring the project in terms of active risks and objective measurements of progress and quality;
- regular reporting of project status, in the status assessment, to the project review authority (PRA), which is the organizational entity to which the project manager is accountable;
- dealing with issues and problems as they are discovered, through the "Activity: Monitor Project Status" or otherwise, and driving these to closure according to the Problem Resolution Plan. This may require that change

requests be issued for work that cannot be authorized by the project manager alone.

The project manager should put in place mechanisms to automate, as far as possible, the collection and reduction of information (metrics, for example) about the project. Time should be spent in analyzing trends, not in collection and calculation. The responsibility for solution of problems that arise on a project obviously ultimately rests with the project manager. However, there is a class of technical problems that should be delegated to the software architect, for example, for solution. The project manager's role is then to implement the suggested solution - which may give rise to a secondary problem, say, lack of resources, which does have to be solved by the project manager. This demonstrates the kind of trust that must exist between the project manager and the technical staff - the project manager expects the software architect to devise sound technical solutions, and the software architect expects the project manager to put in place the infrastructure and resources to implement them, contractual and financial constraints permitting.

In Figure 4.76, "Close-Out Phase" workflow details are shown.

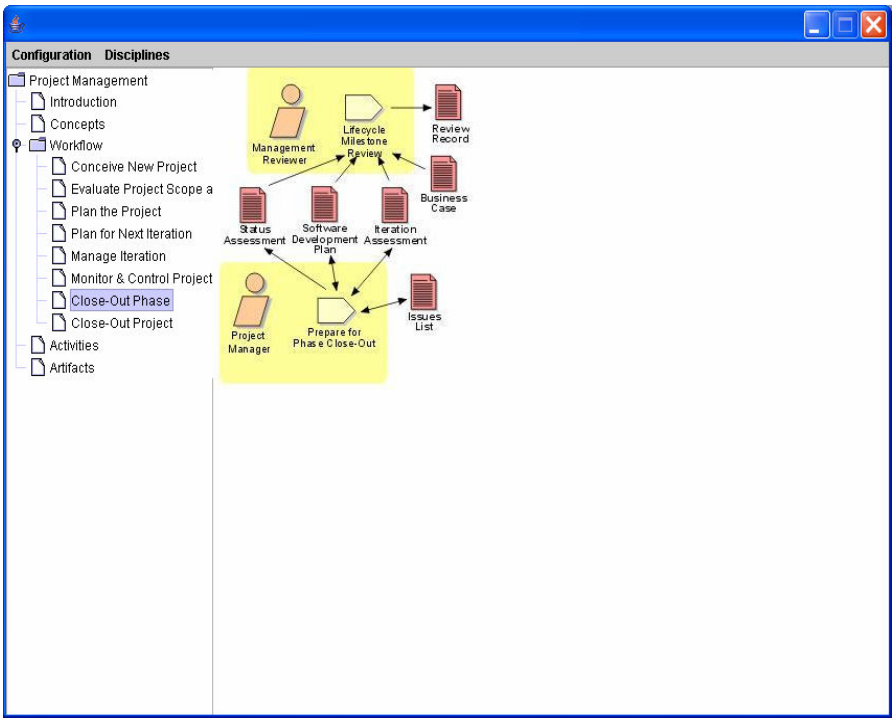

Figure 4.76: Project Management– Workflow – Close Out Phase

In this workflow detail, the Project Manager brings the phase to closure by ensuring that:

- all major issues from the previous iteration are resolved
- the state of all artifacts is known (through configuration audit)
- required artifacts have been distributed to stakeholders
- any deployment (for example, installation, transition, training), problems are addressed
- the project's finances are settled, if the current contract is ending (with the intent to recontract for the next phase)

A final phase status assessment is prepared for the lifecycle milestone review, at which point the phase artifacts are reviewed and, if the project state is satisfactory, sanction is given to proceed to the next phase.

In Figure 4.77, "Close Out Project" workflow details are shown.

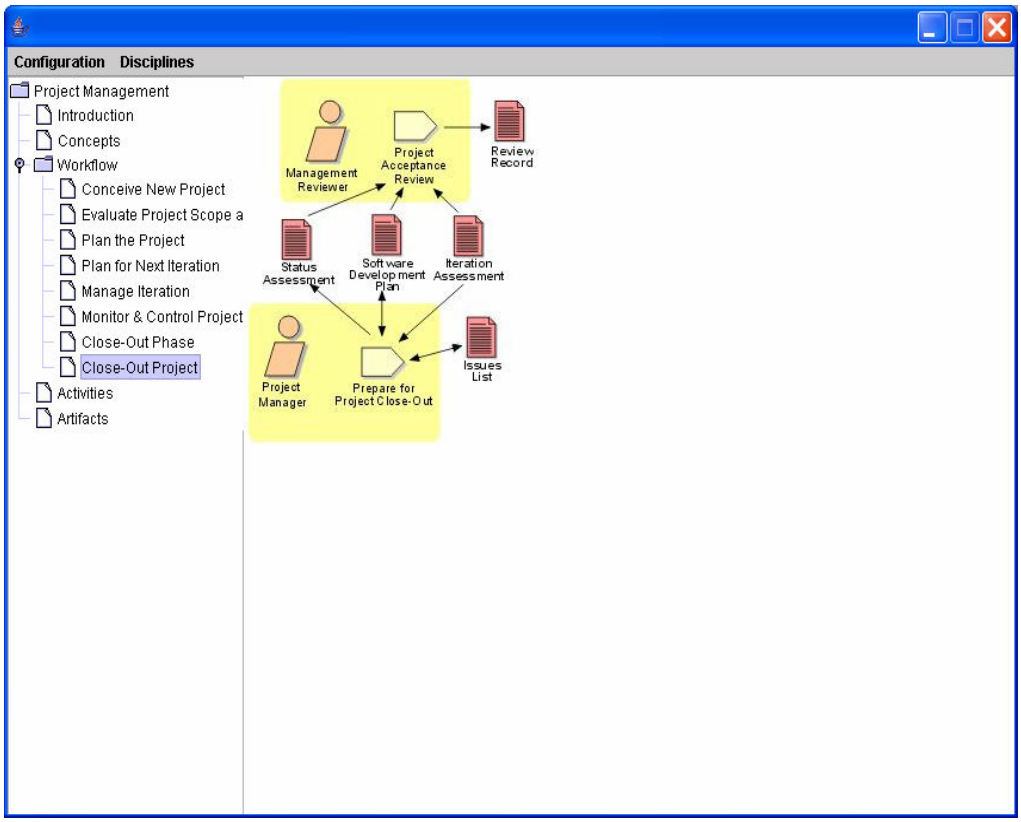

Figure 4.77: Project Management– Workflow – Close Out Project

A final status assessment is prepared for the "Activity: Project Acceptance Review", which, if successful, marks the point at which the customer formally accepts ownership of the software product. The Project Manager then completes the close-out of the project by disposing of the remaining assets and reassigning the remaining staff.

Activities:

In Figure 4.78, acitivites of the "Project Management" discipline are shown. The details are explained in the workflow details menu item. This view is added to the menuitems to list the roles's responsibilities in a clear way.

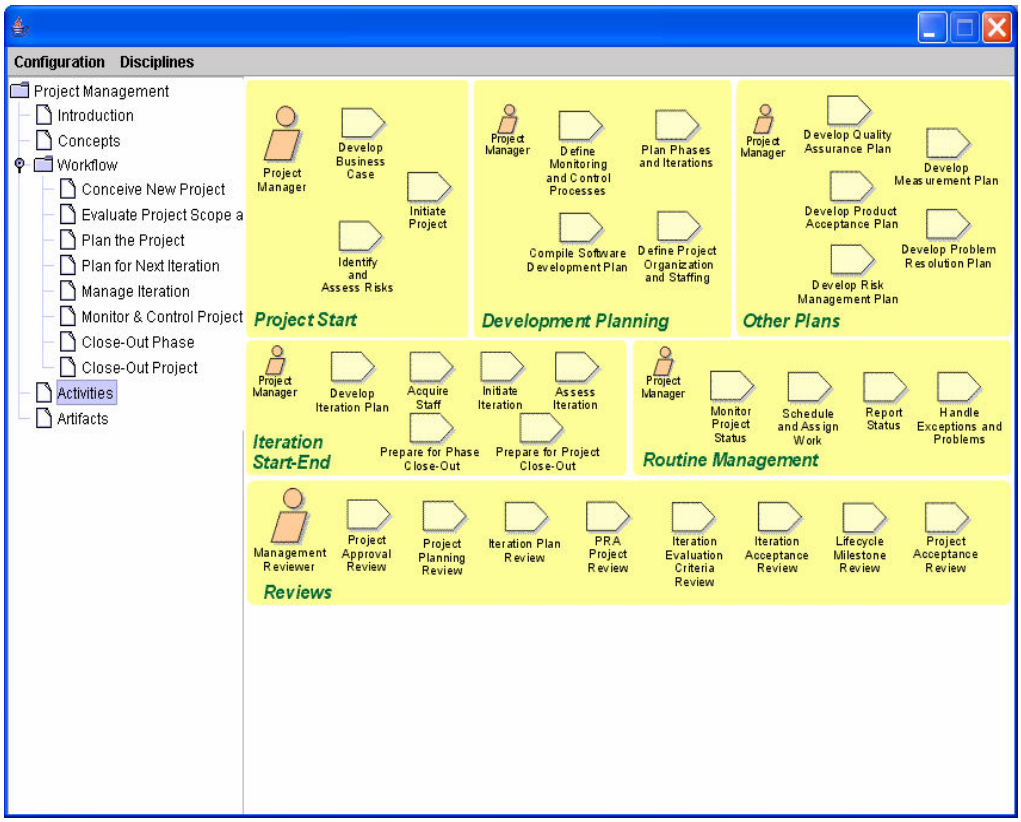

Figure 4.78: Project Management– Activities

Artifacts:

In Figure 4.79, artifacts of the "Project Management" discipline are shown. The workflow details are explained in the workflow details menu item. This view is added to the menu items to list the produced artifacts.

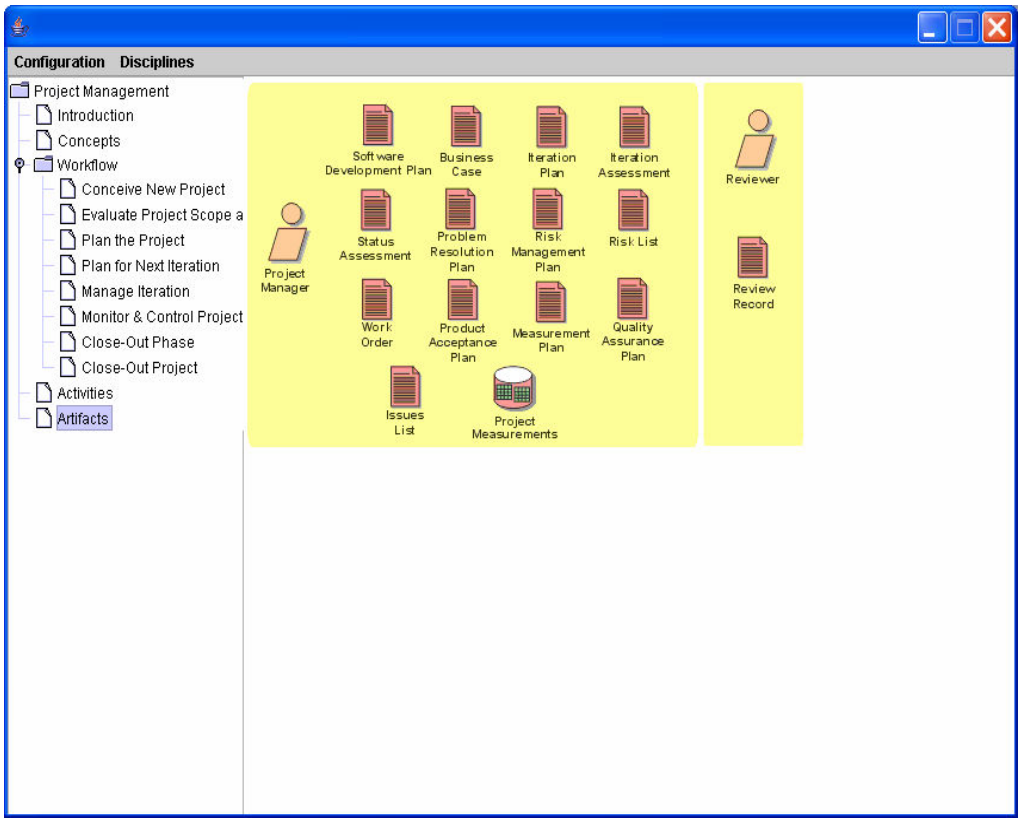

Figure 4.79: Project Management– Artifacts

4.2 Traceability Relations

The implementation tool sould not to be seen just as a document management tool. There are three main areas, where the tool is very strong and be a pioneer.

First, the implementation tool acts a tutorial, which has a deep knowledge on sofftware engineering. It tolds the user, how a process area work, what the main concepts are, which artifacts should be prepared. It also gives teplate and guidelines of those artifacts.

Second, it has a strong configuration management framework. More then just a a fileserver, it keeps track of all the artifacts. It has a self version control mechanism to fulfill this CMMI requirement.

Third, the tracebility issue on main artifacts are the strongets part of this tool. There are lots of discipline spesific tools, which have a deep expertise on the spesific area. But CMMI impresses, that the power of software engineering comes from the overall integrity of all the disciplines. The implementation tool has the both way tracebility, and control properties on the seven disciplines.

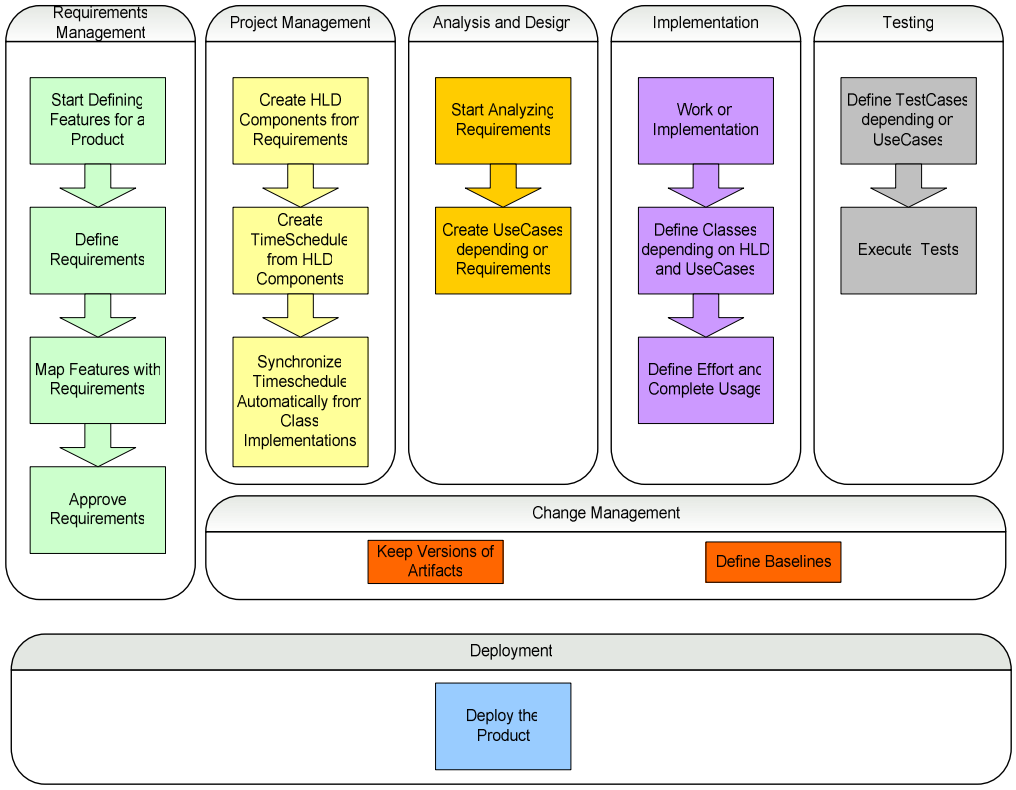

Figure 4.80: Traceability Relation

The traceability starts with requirements management discipline as seen in Figure 4.80. In requirements management discipline, it is required to define requirements. First of all, there should be main ideas, which have to be collected. They are not the real requirements but features. The traceability begins with features. After features are defined, it is time to transfer them as real requirements for products. There could be one to one relationship between features and requiremets, or the feature will not be implementedon this product, or the last scenario there could be many to many relationship. All the relation types are supported from the implementation tool. After approving requirements from development team, the team started to create High Level Design components. These components will also be main steps for activities in implementation phase. The implementation tool has the ability to prepare automatically a project plan using HLD components. The Automation has a lot of advantages like, time saving from preparing the artifact,no mismatches or human error, both way traceability. The plan can be updated on MSProjet side or the HLD can be updated inside the implementation tool. From both ways, the implementation tool could make synchronisation to be consistent.

This plan will also be updated when developers start to begin implementation. They give inputs to the implementation tool, how much effort do they use before committing a class or a component. The implementation tool gives the project manager the information by updating their plan, and putting completing percentages into the plan. So every person, especially project manager could be aware from the flow of the project , whenever they want.

Usecases and testcases are also directly in a relationship with requirements. Every functional requirement is also a usecase in implementation tool. Again a time saving and tracebility advantange appears here between Requirements Management discipline and Test discipline. In Analysis and Design discpline, the classes also defined based on usecase realizations.

The detailed explanation on the spesific disciplines and artifacts are in Discipline Step of this document.

Traceability Flow:

The usage cycle begins with creating a new project in implementation tool. Entering an unique project name and a description is enough information for project authentication.

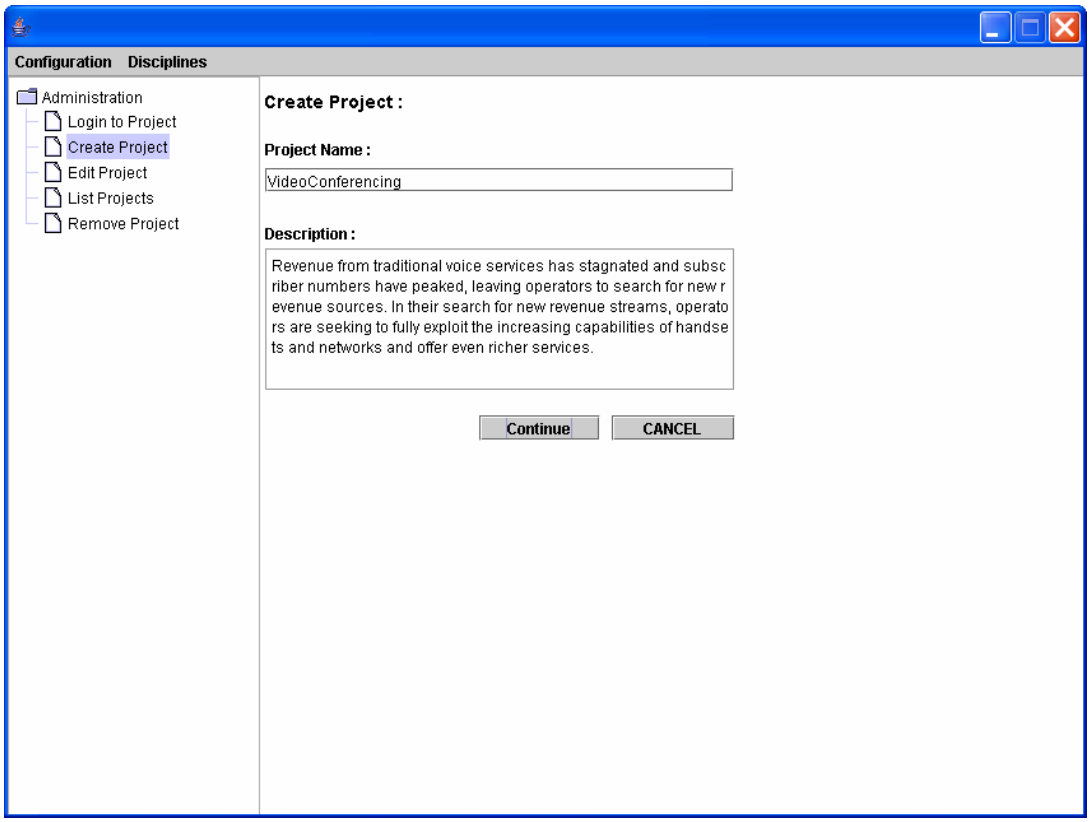

Figure 4.81: Traceability – Create Project

After clicking "Continue" button on the window in Figure 4.79, the implementation tool continues creating users for your project. An user could be in more then one role in the project. It is mandatory to fullfill all the roles for a project.

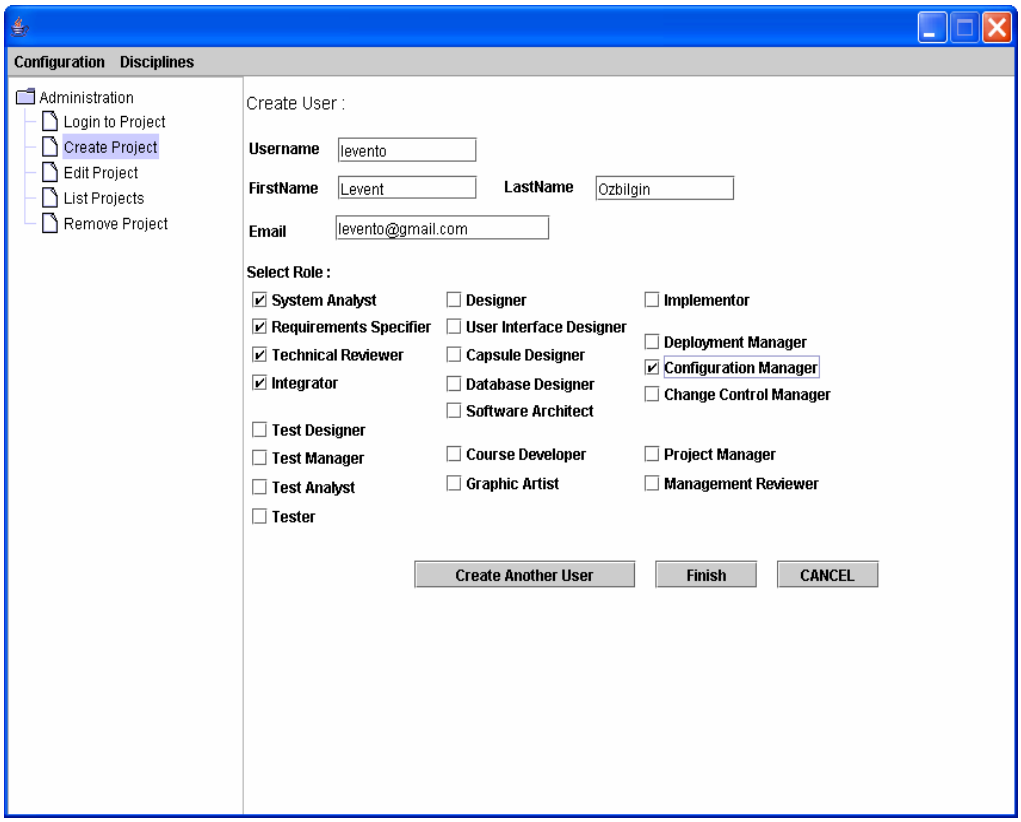

Figure 4.82: Traceability – Create Project - User

To complete creating the user and continue with a new user, it must be clicked the "Create Another User " button o the window shown in Figure 4.82. To complete and finish the user, it must be clicked "Finish" buton. To cancel the creation, it must be clicked "CANCEL" button. It could be then continuing creating user using "Edit Project" menu.

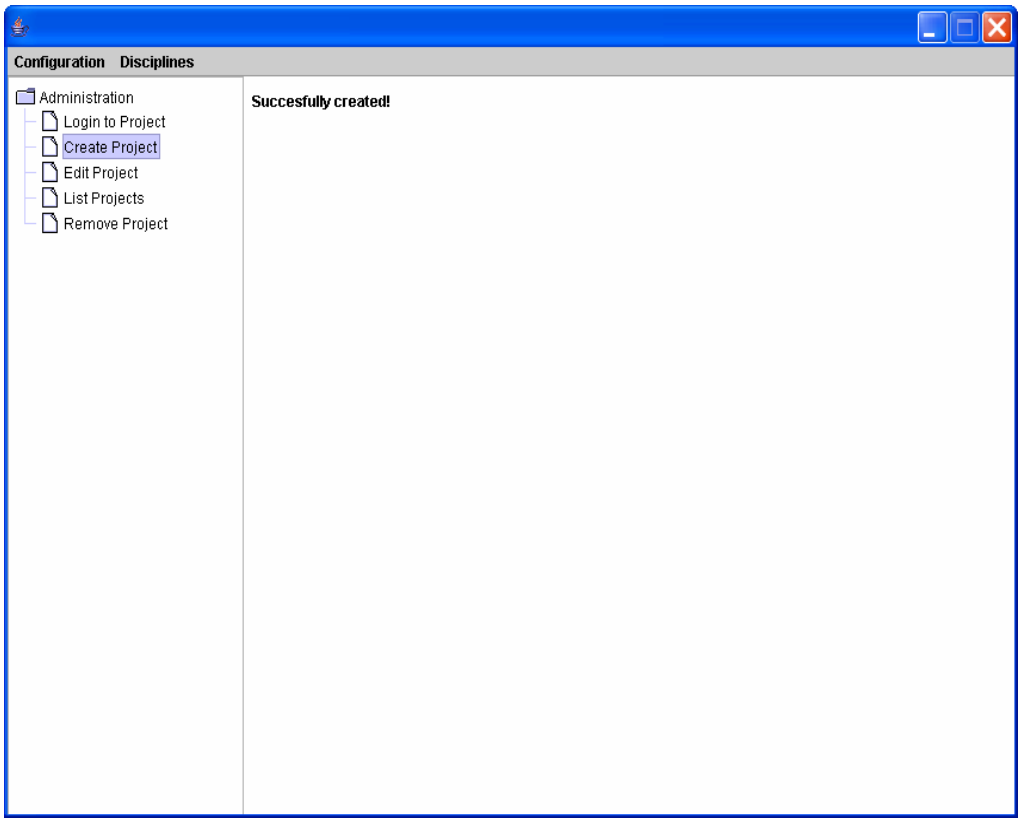

Figure 4.83: Traceability – Create Project - Confirmation

The implementation tool displays the confirmation message ”Succesfully created”, when the project is created on the database side. It is shown on Figure 4.83. To list projects created before, “List Projects” menuitem could be selected. If a project is no longer alive, and need not to be listed on the “List Project” menuitem, then it could be removed using “Remove Project” menuitem.

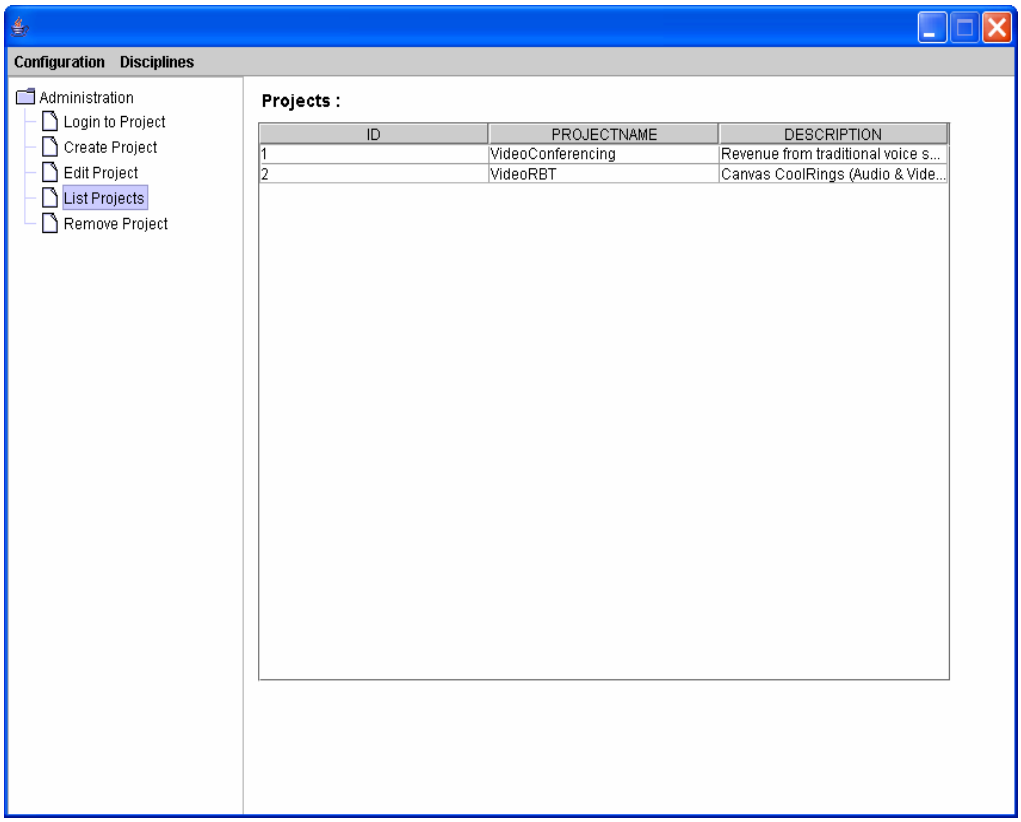

Figure 4.84: Traceability – List Projects

To begin working on a project, users have to login to the project. From "Login to Project" menu item, users select the project using dropdown menu and click "LOGIN" button.

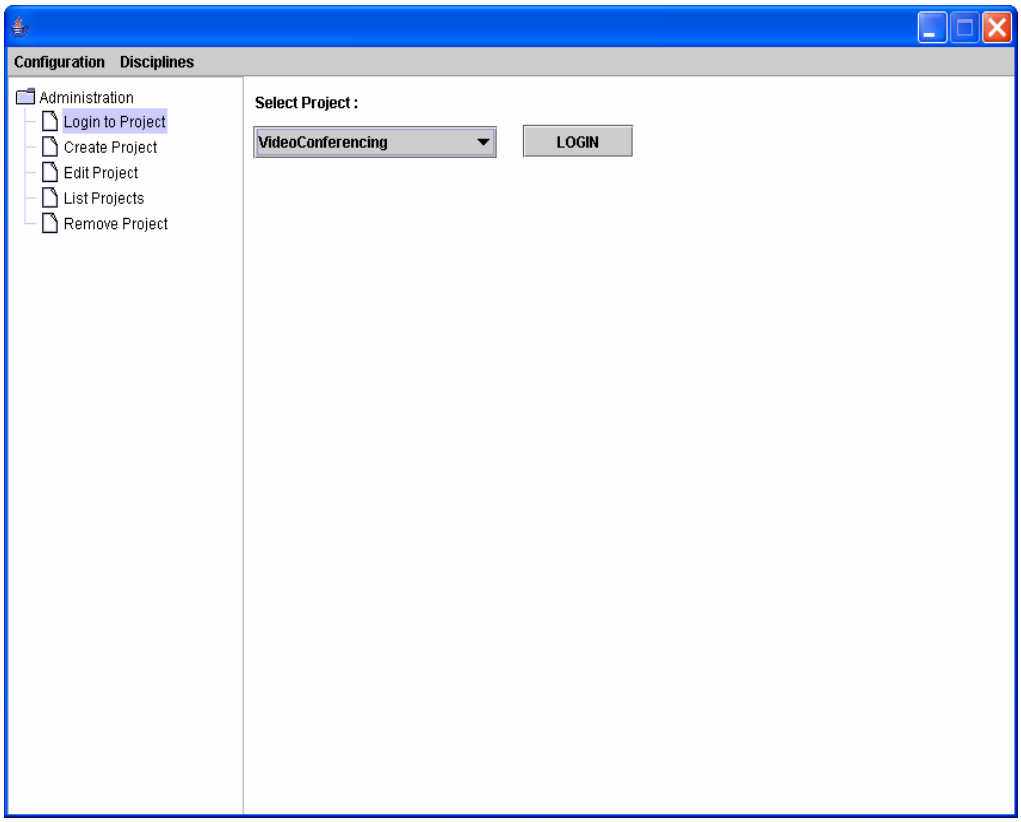

Figure 4.85: Traceability – Login

After loging to the project shown in Figure 4.85, project members can begin working on disciplines selecting from “Disciplines”menu. There are seven disciplines listed on the “Disciplines” menu shown in Figure 4.86. The traceability cycle begins with the concept “defining requirements”. Requirements Management will be the first discipline therefore.

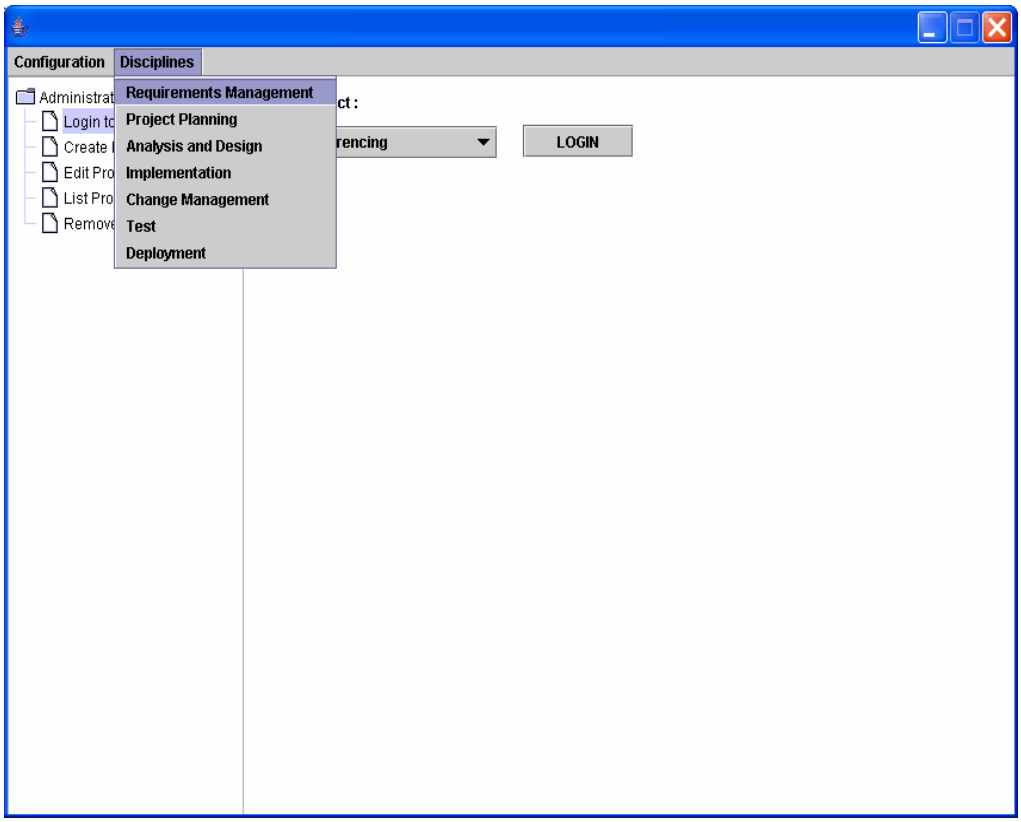

Figure 4.86: Traceability – Disciplines

The discipline detail will be explained later. Every discipline has five menuitems, which are “Introduction”, “Concepts”, “Workflow”, “Activities” and “Artifacts”. The main work on disciplines is to produce some artifacts to be more predictable. The first tracebility property is to keep track of this artifacts. Using “Workflow Details” menuitems or “Artifacts” menuitem, artifacts will be produced or seen. Every artifact document will be kept on the server in a versioned way.

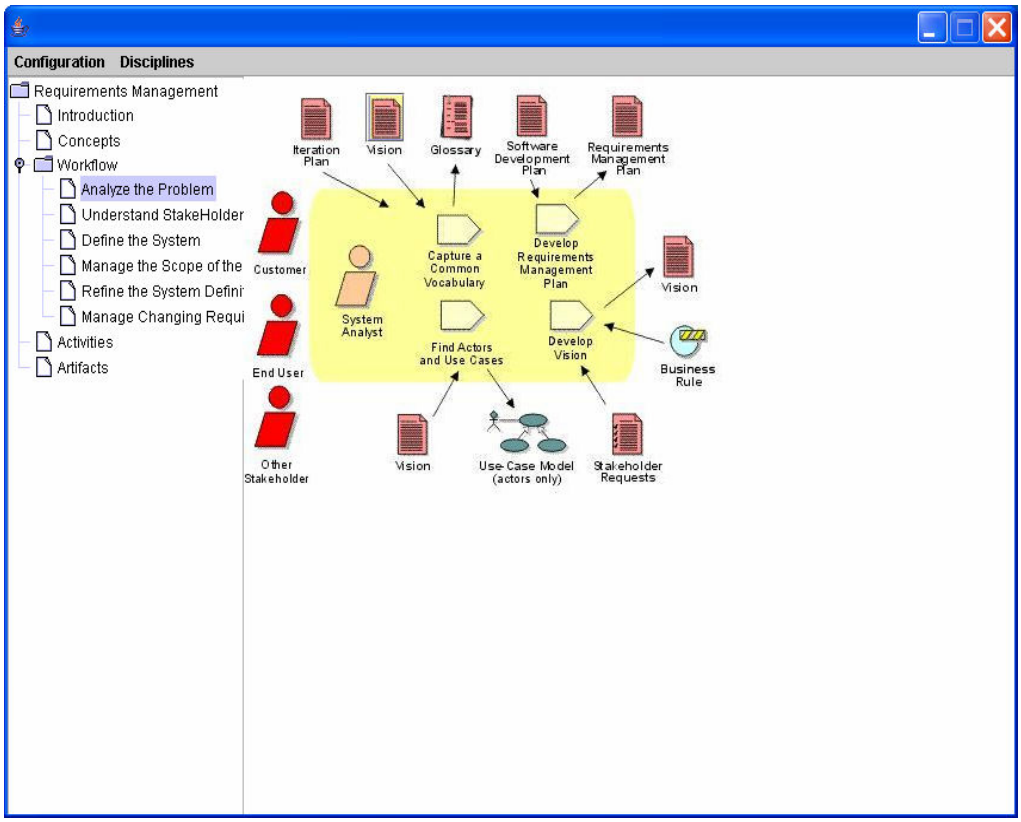

Figure 4.87: Traceability – Artifact

Clicking on a document shown on the Figure 4.87 will guide to another window, where a new document or a new version of a document will be created. Documents latest version, document template could be seen on this window shown in Figure 4.86. If it is the first time, "Latest Version" will be "v0". The user could view the template (shown in 4.87) document download it and fill the template depending on his project.

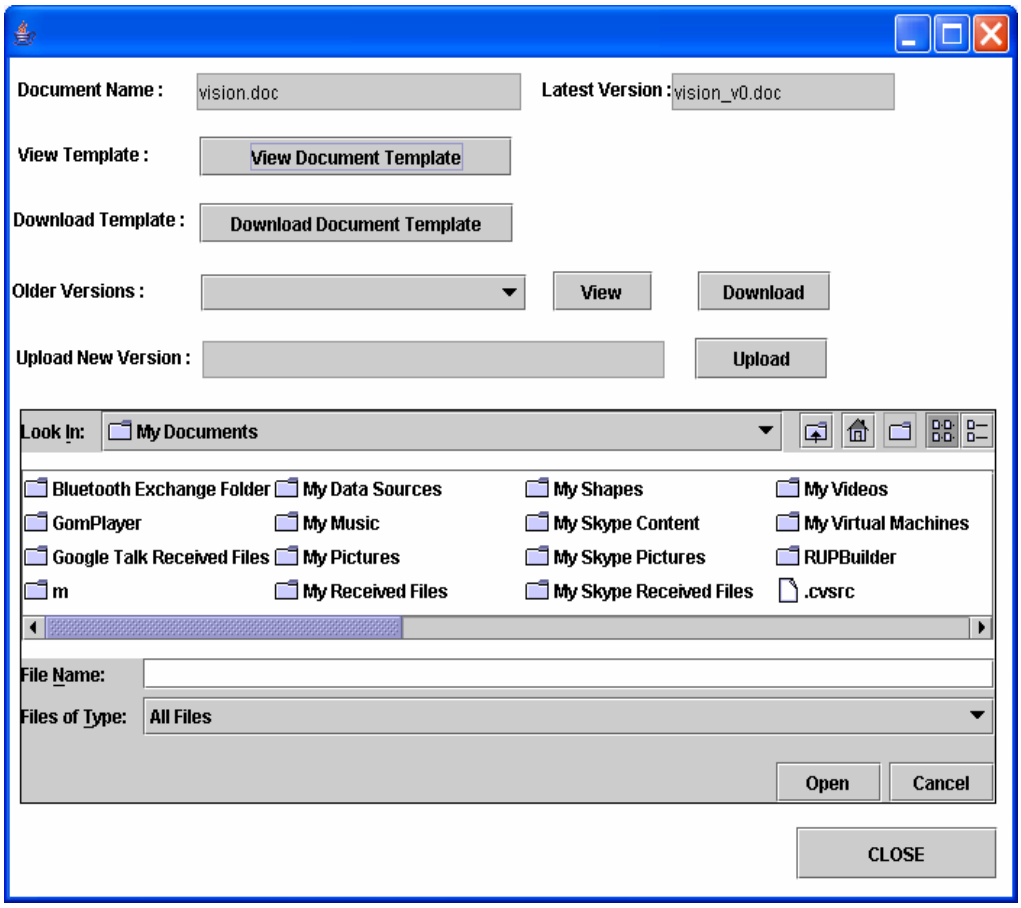

Figure 4.88: Traceability – Versioning

All the view activities opens documents using their original programs.

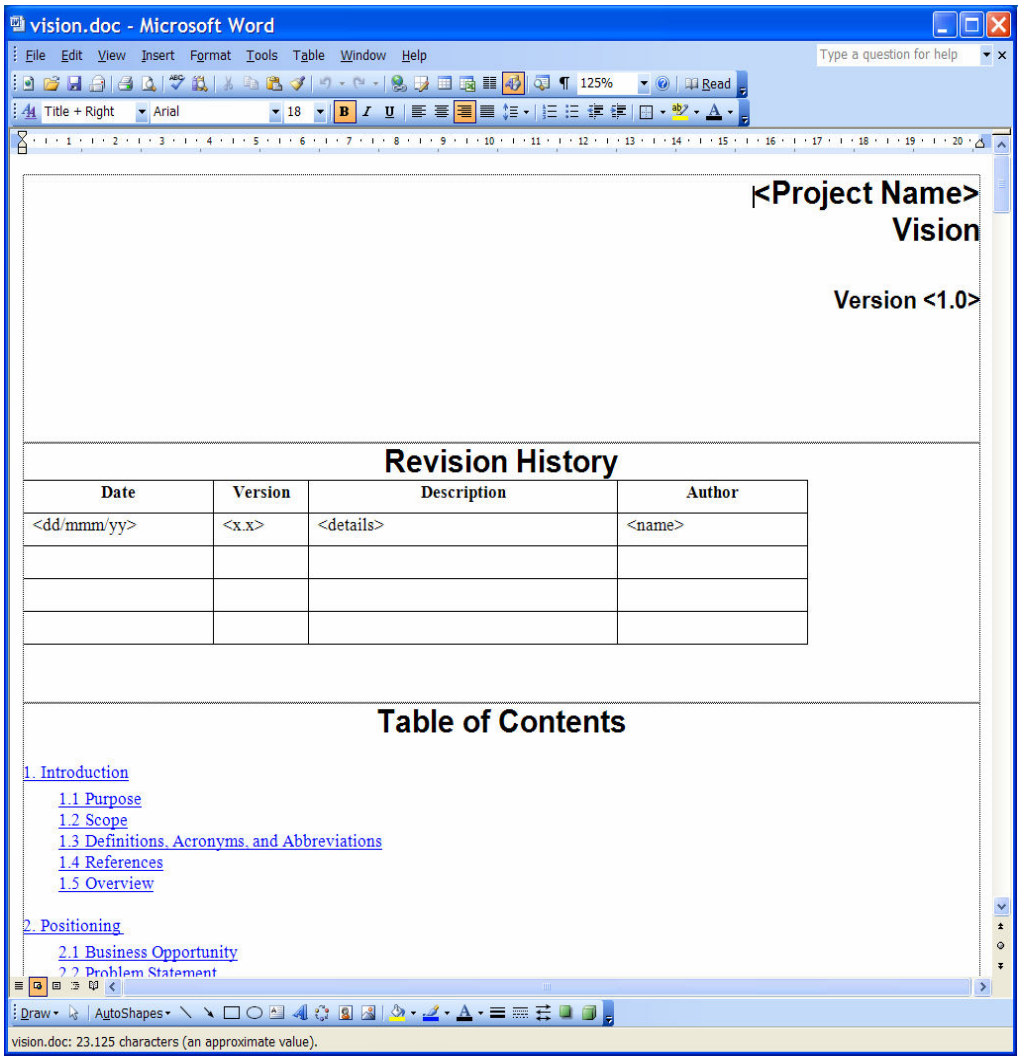

Figure 4.89: Traceability – Template

To create the first version "v1", the browse utility should be used. To confirm the selection the user clicks on the document twice or clicks on Open button. Then click on "Upload" button, to send the file to the server. Figure 4.90. Older versions could be viewed, when selected from "Older Versions" combobox.

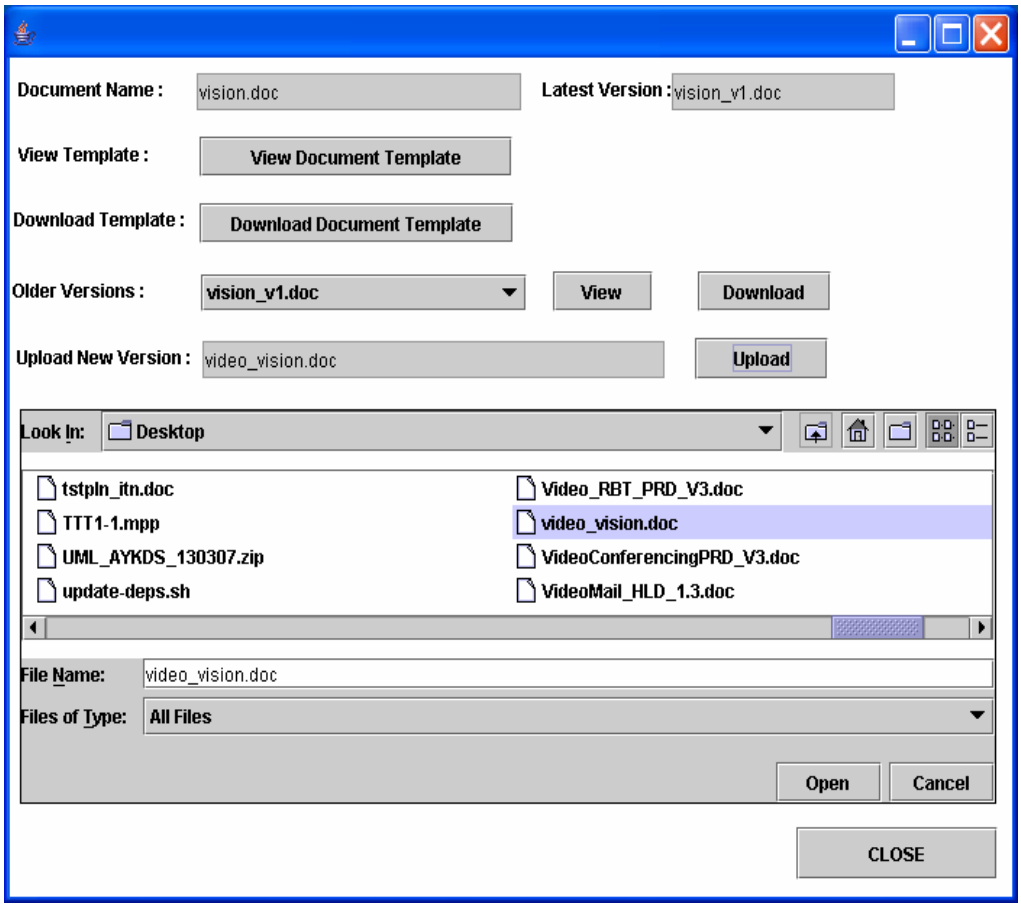

Figure 4.90: Traceability – Select Artifact

All the artifacts used in this project cycle will be versioned. This window is availale for all the artifacts in the process framework.

The next tracebility item on this process improvement framework is the db icon. All the requirements will be kept in a repository. So, every single requirement could be traced till the project plan.

To start requirement definition, click on the repository icon on Figure 4.91.

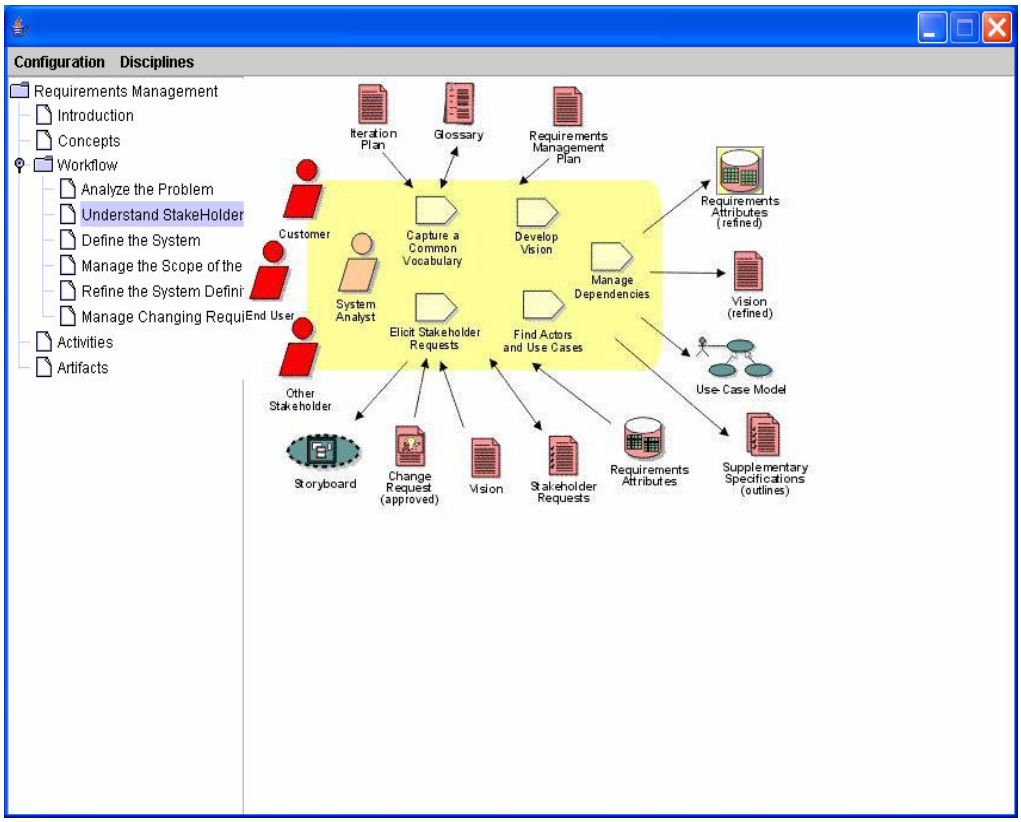

Figure 4.91: Traceability – Requirements

A new repository query window will be opened shown in Figure 4.91. There are three main types of requirements:

- Features,
- Requirements,
- High Level Design(HLD) components

The first step is define features for the product. All the fields are mandatory. Headline should be unique. Clicking on “OK ” button will save the reocrd to the project repository.

Features are not real requirements for the product. They represents ideas, market analysis items, customer whishes, etc..

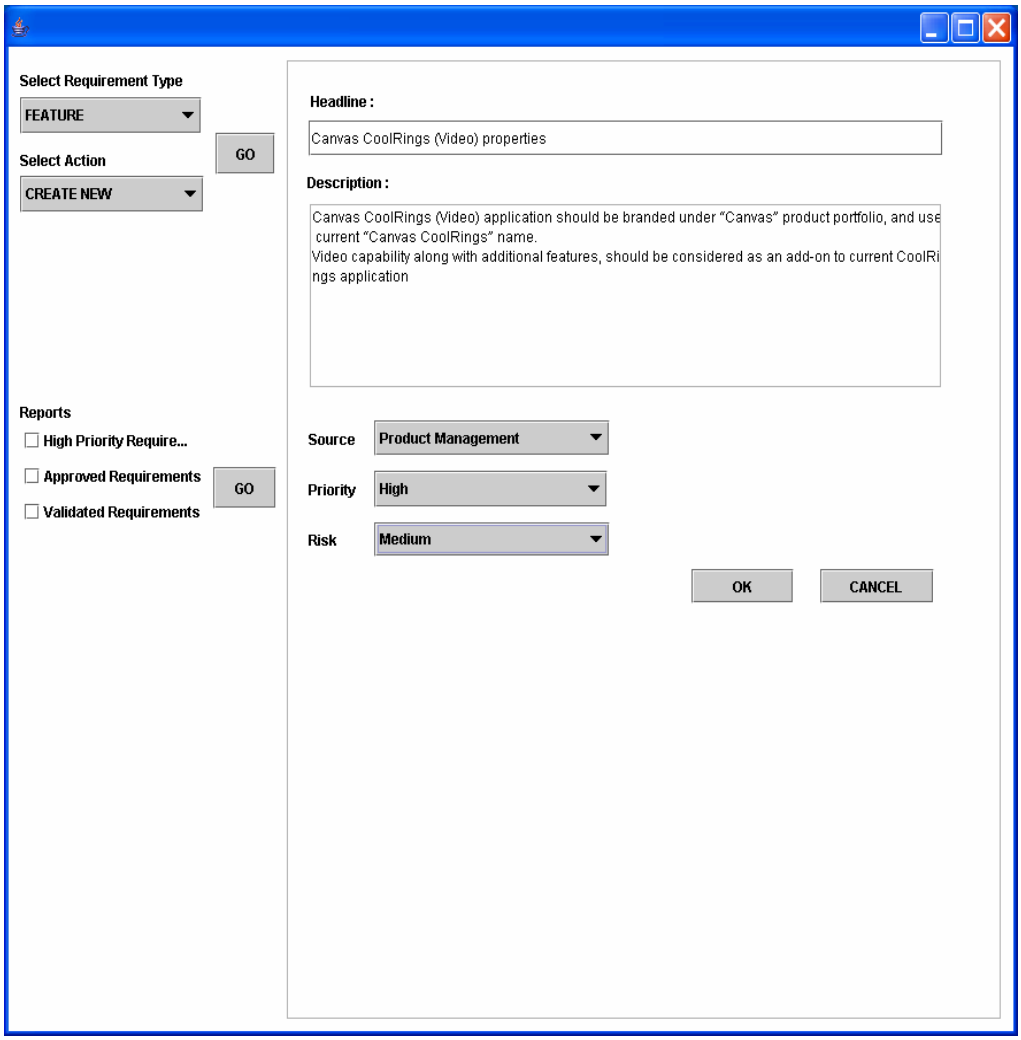

Figure 4.92: Traceability – Ceate Feature

A requirement is defined as "a condition or capability to which a system must conform". Requirements should be in a relation with features. A requirement spesific design is a customer centric approach to the issue. The Requirement creation screen could be seen on Figure 4.93.

The attributes assigned to each requirement will be used to manage the software development and to prioritize the features for each release.

The objective of requirements traceability is to reduce the number of defects found late in the development cycle. Ensuring all product requirements are captured in the software requirements, design, and test cases improves the quality of the product.

Status:

Set after the analysis has drafted the use cases. Tracks progress of the development of the use case from initial drafting of the use case through to final validation of the use case.

Proposed: Use Cases which have been identified though not yet reviewed and approved.

Approved:Use Cases approved for further design and implementation.

Validated:Use Cases which have been validated in a system test.

Priority

Set by the Project Manager. Determines the priority of the use case in terms of the importance of assigning development resources to the use case and monitoring the progress of the use case development. Priority is typically based upon the perceived benefit to the user, the planned release, the planned iteration, complexity of the use case (risk), and effort to implement the use case.

High: Use Case is a high priority relative to ensuring the implementation of the use case is monitored closely and that resources are assigned appropriately to the task.

Medium:Use Case is medium priority relative to other use cases.

Low:Use Case is low priority. Implementation of this use case is less critical and may be relayed or rescheduled to subsequent iterations or releases.

Technical Risk:

Set by development team based on the probability the use case will experience undesirable events, such as effort overruns, design flaws, high number of defects, poor quality, poor performance, etc. Undesirable events such as these are often the result of poorly understood or defined requirements, insufficient knowledge, lack of resources, technical complexity, new technology, new tools, or new equipment.

High:The impact of the risk combined with the probability of the risk occurring is high.

Medium:The impact of the risk is less severe and/or the probability of the risk occurring is less.

Low:The impact of the risk is minimal and the probability of the risk occurring is low.

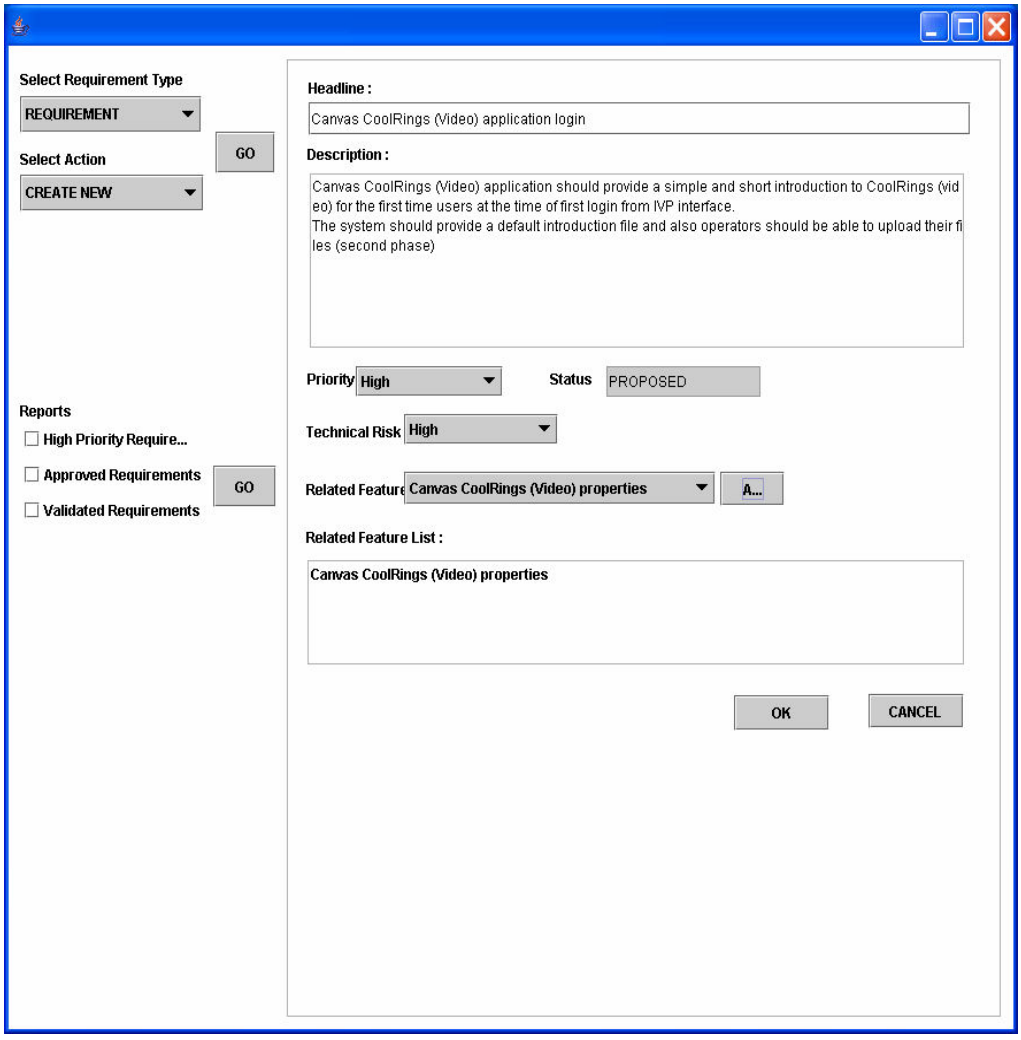

Figure 4.93: Traceability – Requirement Create

Requirements explains what the product have to do. In order to built the right product, they are very important. In every phase of the development, there are practices, that have directly relationship to requirements.

From requirements, first the activity items should be prepared. Activity items are High Level Design components. Choosing HLD Component from "Select Requirement Box" Combobox, new HLD Components could be createted.

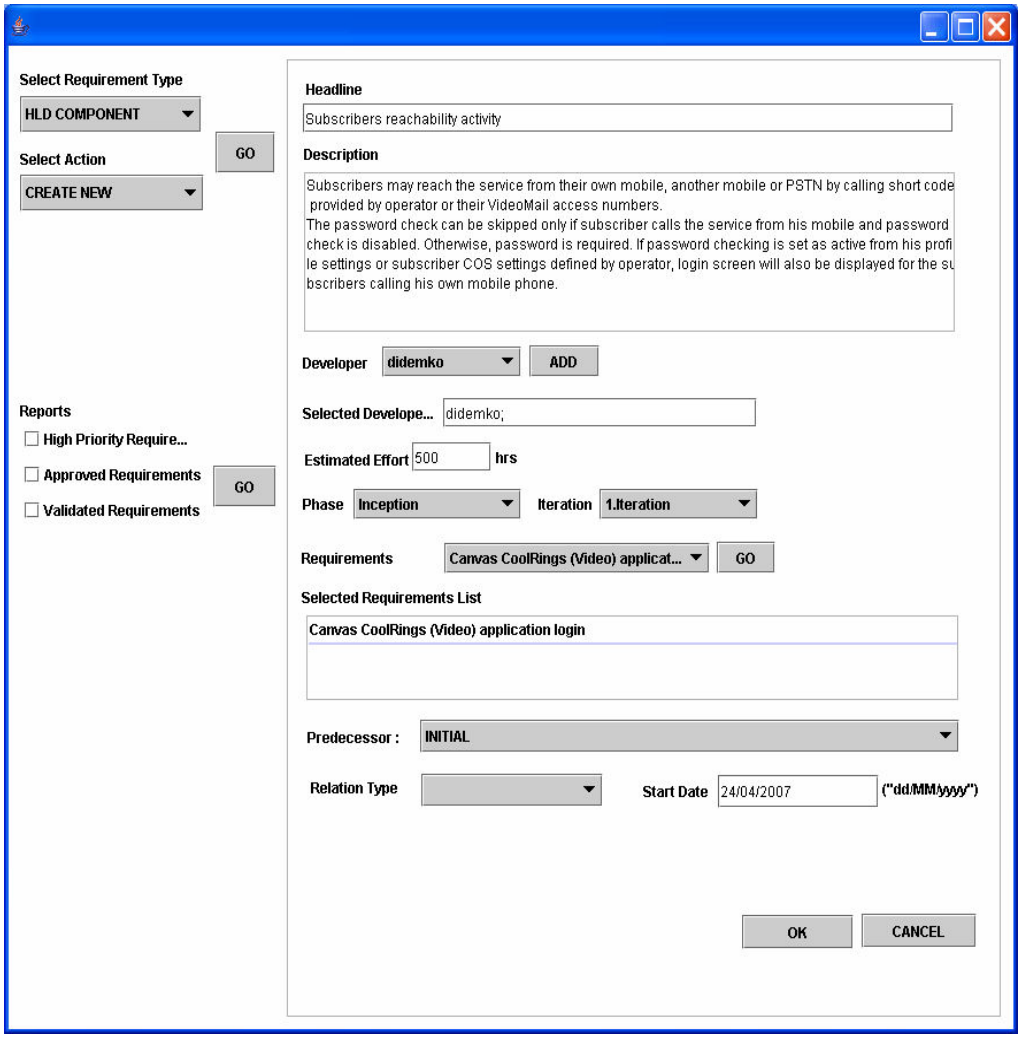

Figure 4.94: Traceability – Create HLD Component

Every HLD Component should cover one or more Requirements. One or more developers should be assigned to a HLD Component shown in Figure 4.94. The Phase and iteration number should be specified. HLD components will be used as acitivities when preparing automatically the project plan. So the predecessor combobox(if initial selected, then StartDate have to be filled), relation type property and estimated effort should not leave empty.

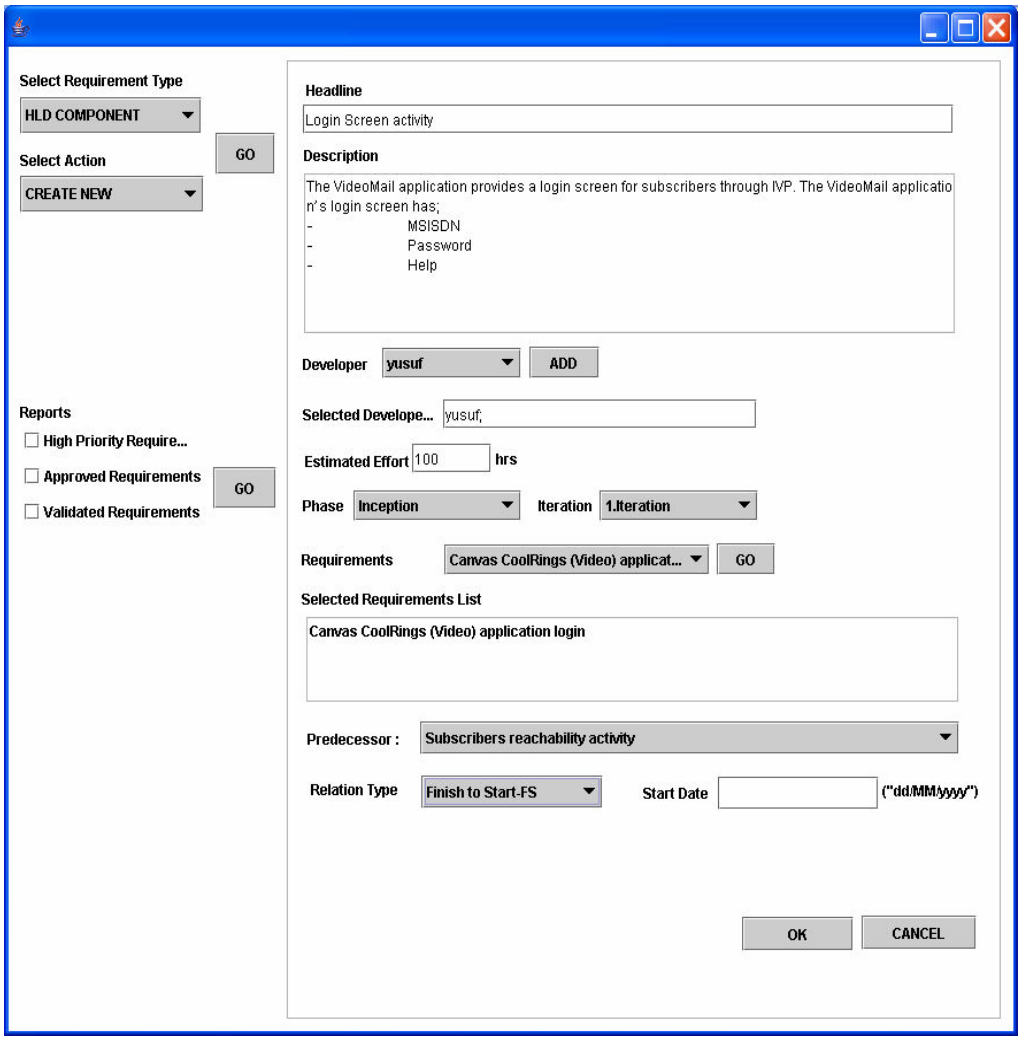

Figure 4.95: Traceability – Create HLD

In case that a predecessor is selected, then the user could leave startdate empty like shown in Figure 4.93.

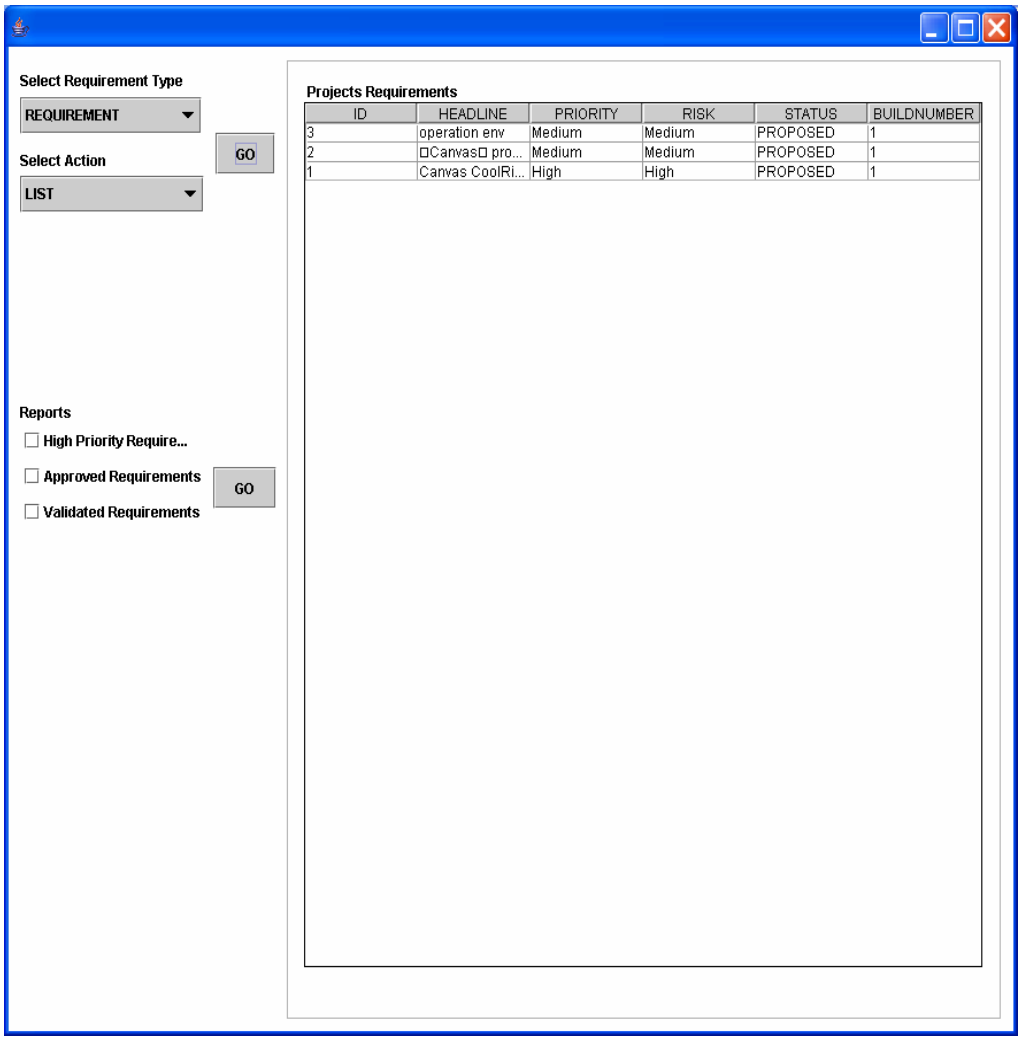

Figure 4.96: Traceability – List Requirements

For all three types of requirements, there are list and update screens. In Figure 4.96 a HLD component update screen is shown 4.97. After selecting a component from the first combobox, all the responsible fields are filled with the values from repository. The values could be changed using the update property.

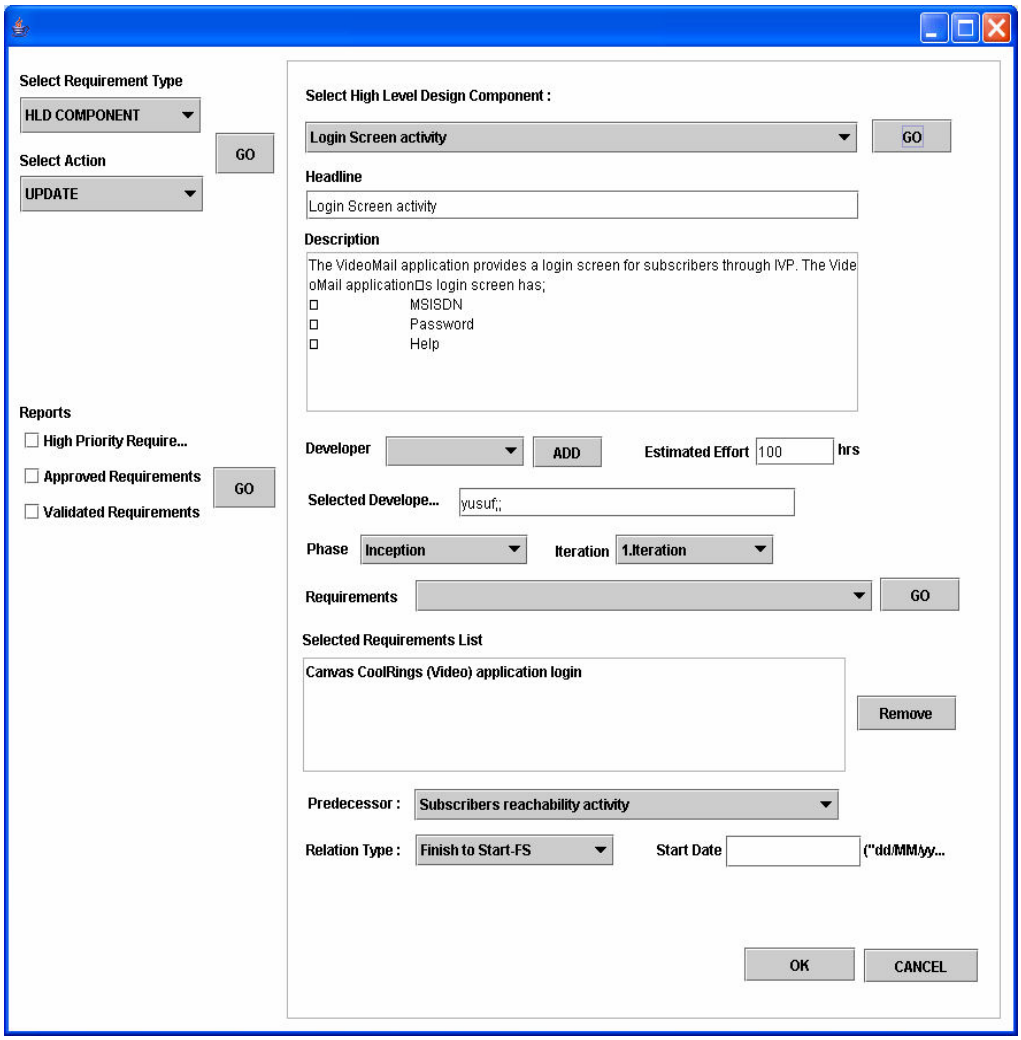

Figure 4.97: Traceability – HLD Update

The next step for tracesbility is to create automatically the project plan. From the HLD components created in "Requirements Management" discipline, a project plan in Microsoft Project will be created. All the HLD Components will be activities for project plan. Their relationships, developers, effort estimates are inputs for project plan. The responsible document for this relation is "Software Development Plan" as shown in Figure 4.98.

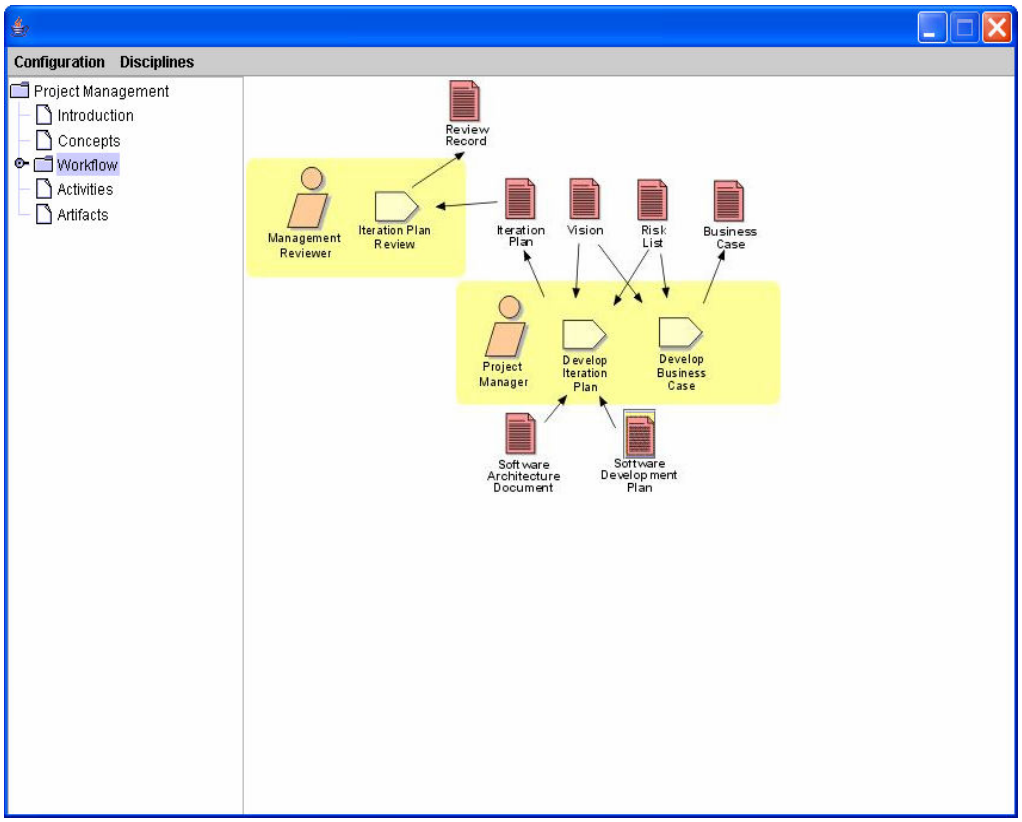

Figure 4.98: Traceability – Plan

Similar to version control window, the project plan window also helps to create project plan. Clicking "GO" button from "Create Initial Software Development Plan TimePlan/Schedule" will create a new project plan, depending on values from repository. If it is the first time, then the project plan created and the Latest version will display the "mssdpln_v1". To view the plan, select it from "Older Versions" and click on "View " button. I is possible to download and make modifications on hat plan. After finishing the modifications, it could be uploaded using "Upload and Synchronize with HLD Components Database" as shown in Figure 4.99.

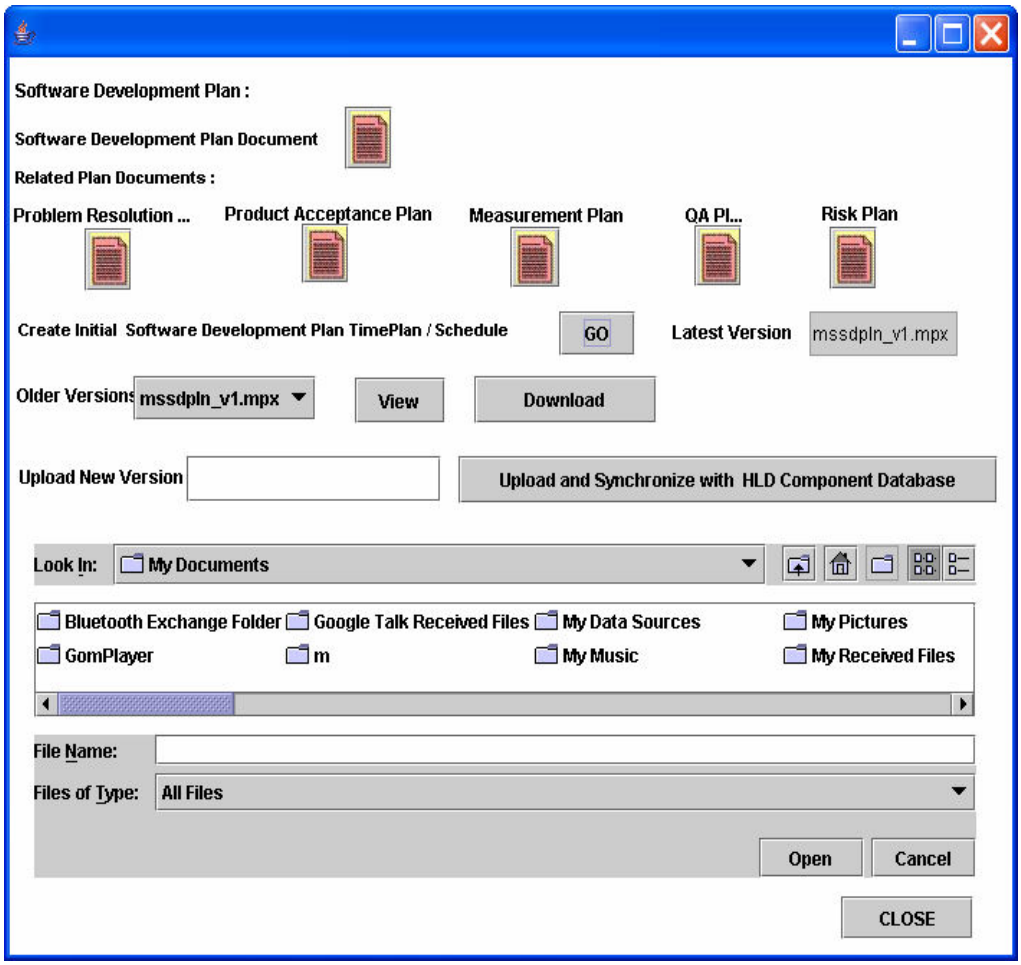

Figure 4.99: Traceability – Create Project

In Figure 4.100, the first version of project plan is created depending on repository values.

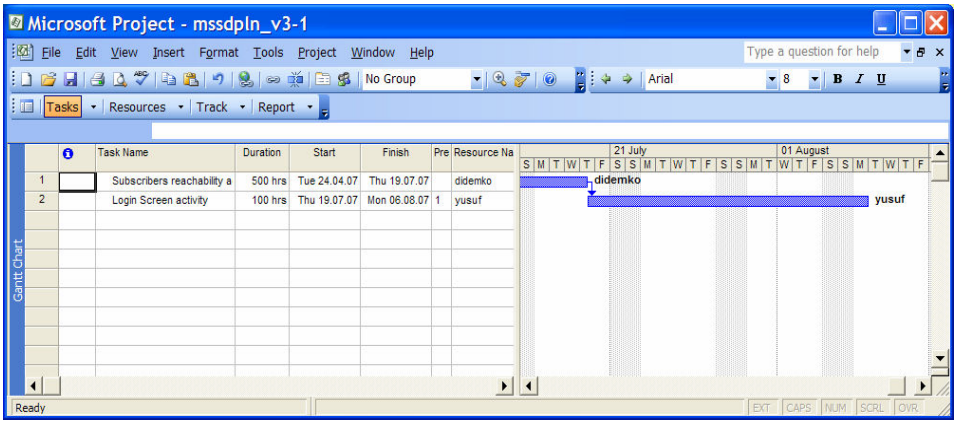

Figure 4.100: Traceability – Open MS Project

Traceability is in both ways. If there is modification in project plan, then it could be saved in MS Project format.

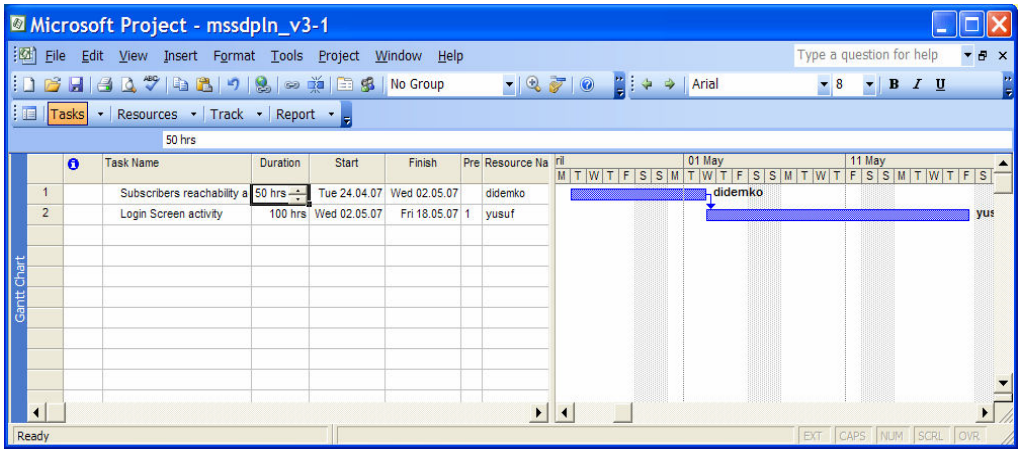

Figure 4.101: Traceability – Update MS Project

Changes in MS Project side like in Figure 4.101, could be uploaded using the "Upload and Synchronize with HLD Component Database" button. This option creates new version of the plan on server and updates related repository values as shown in Figure 4.102.

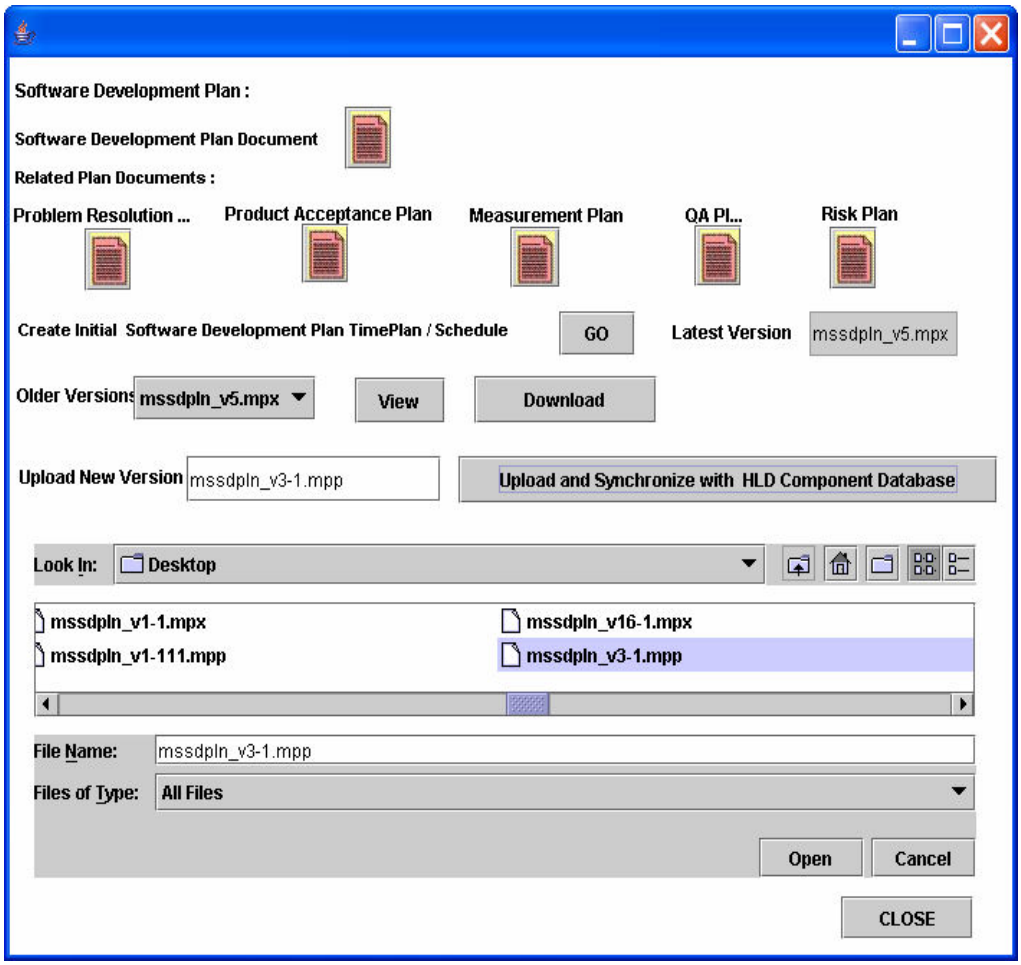

Figure 4.102: Traceability – Update Database

The update could also be traced from Requirement side. So both ways traceability could be established as seen on Figure 4.103.

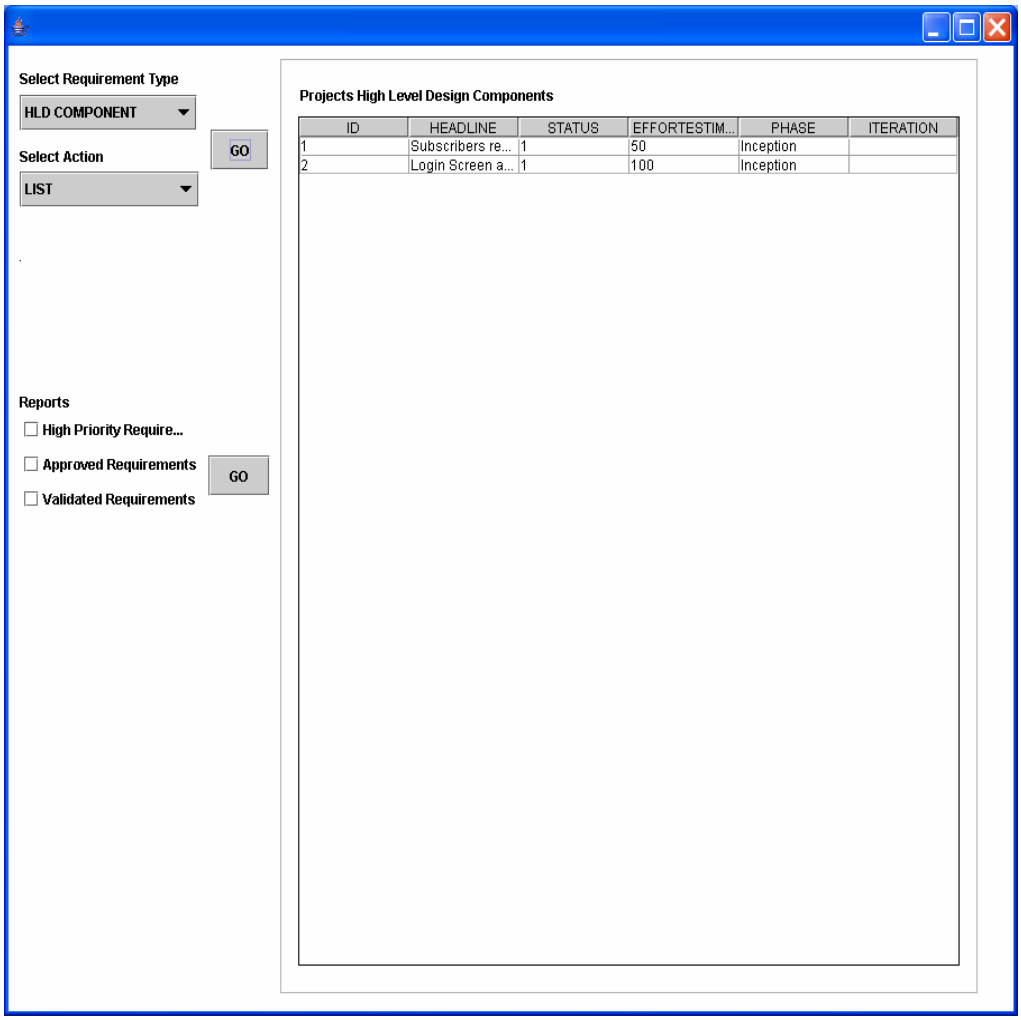

Figure 4.103: Traceability – List HLD

RESULTS AND DISCUSSIONS

The ultimate goal of software engineering is to develop a high quality product in time and at reasonable costs. But since the time software is developed a phenomenon called “software crisis” exists subsuming wrong schedules and cost estimates, low productivity of people as well as low product quality. A promising approach out of this crisis is now growing up in the software engineering community. The underlying assumption of this approach is that the quality of a software product to a high degree relies on the quality of the software process. Therefore quite a few software process improvement (SPI) approaches were developed during the last years.

Usually, todays software processes are supported and partly automated by tools. The umbrella stands for a large number of applications ranging from simple editing tools to environments supporting the whole software life cycle [3]. In point of view there are important interdependencies between an organization’s software development environment consisting of tools and people and a software process improvement approach. The configuration of the software development environment may influence progress and success of the implementation of a SPI approach to a high degree. Two viewpoints have to be distinguished.

The first one is concerned with human factors in SPI. People affected by changes have to be informed about activities planned as well as their goals and intents. Beyond that, they have to be motivated to actively participate in the improvement process and they have to be trained to be able to positively influence the SPI efforts. The implementation tool and underlying process framework is not enaugh for the SPI succes of a company or project. There should be users, which understand the needs and benefitsof the improvement process.

Second, the tool environment of a software development organization has to be adapted to the new way of software engineering driven by a SPI approach. Tools thus not only have to support production process activities like developing an analysis

document or coding a module, but also meta process support like process management or process monitoring. The important question for an organization is how to choose the right tools environment in order to promote the implementation of a software process improvement approach. The implemetation solution provided in this thesis is a right solution for the middle and small scale companies,which aren't yet so far institutionalized. With the concept institutionalization, the companies have had some processes, which could be hardly changed. What the thesis offers is an end to end solution. So the compaines should adapt their old processes to the tool.

CONCLUSION

Different advances have been made in the development of software process improvement (SPI) standards and models, e.g.capability maturity model (CMM), more recently CMMI, and ISO's SPICE. However, these advances have not been matched by equal advances in the adoption of these standards and models in software development which has resulted in limited success for many SPI efforts. The current problem with SPI is not a lack of standard or model, but rather a lack of an effective strategy to successfully implement these standards or models. The importance of SPI implementation demands that it be recognised as a

complex process in its own right and that organizations should determine their SPI implementation maturity through an organized set of activities. In the literature, much attention has been paid to ''what activities to implement'' instead of ''how to implement'' these activities. We believe that identification of only ''what'' activities to implement is not sufficient and that knowledge of ''how'' to implement is also required for successful implementation of SPI programmes.

Automated tool support is a productive way to enhance the visibility of processes, to identify processes weakness and to better understand the processes. A tool can also be used to observe the behaviour of different activities and their interactions. The participants suggested that this tool will speed up the process of SPI implementation assessment.

Despite all the differences, company type, application domain and CMM maturity levels, companies should get real benefits using process improvement framework and implementation tool. The framework acts as a guidence, which all the practitioners need during interpretation and implementation of the SPI models. Practical implementation of the things, which were explained in the models, could be easily done with the help of the implementation tool. This meta-support structure appears to

be practice introduction, refinement and extension, standardization, enforcement, measurement of results, analysis of measurements and trainibg of its users. When implemented as a whole package, the need for iterative improvementmay be eliminated alltgether, thus shortening the time to process improvement.

REFERENCES

[1] "*IEEE Standard Glossary of Software Engineering Terminology*," 1990 IEEE, Piscataway, NJ std 610.12-1990,.

[2] **P.C. Paulk, B. Curtis, M.B. Christie, C.V. Weber,** 1993, Capability Maturity Model for Software, version 1.1, Software Engineering Institute, Camegie Mellon University, CMU/SEI- **93**-TR-24

[3] **L,. Kerschberg, H. Gomaa, R.G. Mohan, G.A. Farmkh**, Feb 1996, "PROGEN: A Knowledge-based System for Process Model Generation, Tailoring and Reuse", ISSE-TR96-05, Information and software Systems Engineering, George Mason University

[4] **Victor R. Basili, H.Dieter Rombach.**, Sep. 1991, "Support for Comprehensive Reuse" *Software Engineering Journal* **6.5**, pp 303-3116.

[5] **Sergio Bandinelli, Elisabetta Di Nitto, Alfonso Fuggetta,** 1994 , "Policies and Mechanisms to Support Process Evolution in PSEEs", Proceedings of the 3rd intemational Conference on the software process.

[6] **V.R. Basili, H.D. Rombach,** June 1988, "The TAME project : Toward improvement-oriented software environment," IEEE Trans. On Software Engineering, Vol SE-**14**, pp 758-773,

[7] **Aaen, I. Aalborg Univ., Denmark;** 2003, Software Process Improvement: Blueprints versus recipes, Software, IEEE. ISSN: 0740-7459 INSPEC Accession Number: 7728301.Digital Object Identifier: 10.1109/MS.2003.1231159. 86-93.

[8] **Limerick, Ireland.,**2000 On page(s): 626-633 Meeting Date: 06/04/2000 - 06/11/2000 Location: ISBN: 1-58113-206-9 References Cited: 11 INSPEC Accession Number: 6727624 Digital Object Identifier: 10.1109/ICSE.2000.870456

[9] **Ngwenyama, O. Nielsen, P.A. ,** 2002, Dept. of Inf. Syst., Virginia Commonwealth Univ., Richmond, VA, USA; Competing values in

software process improvement: an assumption analysis of CMM from an organizational culture perspective; Engineering Management, IEEE Transactions on Publication Date: Feb. 2003 Volume: **50**, Issue: 1 On page(s): 100- 112 ISSN: 0018 9391 INSPEC Accession Number: 7709789 Digital Object Identifier: 10.1109/.808267

[10] **M. Fritsch, Monika Meschede,** 2000, Product innovation, process innovation, and size, Technical University Bergakademie Freiberg.

[11] **Brodman, J.G. Johnson, D.L.**, 1994. Proceedings. ICSE-16., 16th International Conference on Publication Date: 16-21 May 1994 On page(s): 331-340 Meeting Date: 05/16/1994 - 05/21/1994 Location: Sorrento, Italy ISBN: 0-8186-5855-X References INSPEC Accession Number: 4711538

[12] **www.software.org/quagmire**

[13] **By Philippe Kruchten** Published 2003 Addison-Wesley professional 320 pages ISBN 0321197704

[14] **CMMI® Distilled**, 2003, A Practical Introduction to Integrated Process Improvement, Second Edition By Dennis M. Ahern, Aaron Clouse, Richard Turner Publisher: Addison Wesley Pub Date: September 23, 2003 ISBN: 0-321-18613-3

[15] **Software Engineering Enstitute,** CMMI for Development, Version 1.2 CMMI-DEV, V1.2 CMU/SEI-2006-TR-008 ESC-TR-2006-008

[16] **Ojelanki Ngwenyama and Peter Axel Nielsen.** , Feb 2003, Competing Values in Software Process Improvement: An Assumption Analysis of CMM From an Organizational Culture Perspective. IEEE Transactions On Engineering Management, VOL. **50**, NO. 1

[17] **Proceedings of 1st International Conference on Information and Communication Technology, ICICT 2005, v 2005**, Proceedings of 1st International Conference on Information and Communication Technology, ICICT 2005, 2005, p 296-301 Implementation and analysis of CMMI's configuration management process area; Applicable to "defined" Level - 3

[18] **Rassa, Robert C. (Systems Supportability, Raytheon Electronic Systems Company, MS R1/B510); Garber, Vitalij; Etter, Delores,** 2002, 1 Capability maturity model integration (CMMI): A view from the sponsors. Source: Systems Engineering, v 5, n 1, February, 2002, p 3-6

[19] **Niazi, Mahmood (Faculty of Information Technology, University of Technology Sydney); Wilson, David; Zowghi, Didar**., 2005, A maturity model for the implementation of software process improvement: An empirical study, Journal of Systems and Software, v **74**, n 2 SPEC. ISS., p 155-172

[20] **Niazi, Mahmood (National ICT Australia, Empirical Software Engineering, Bay 15 Locomotive Workshop); Wilson, David; Zowghi, Didar**, A framework for assisting the design of effective software process improvement implementation strategies, Journal of Systems and Software, v **78**, n 2, p 204-222

[21] **Ellmer, E. (Dept. of Inf. Eng., Wien Univ., Austria); Merkl, D.,** 1996, Defining a set of criteria for the assessment of tool support for CMM-based software process improvement, Proceedings of the Fourth International Symposium on Assessment of Software Tools (Cat. No.96TB100054), p 77-86

[22] **Bilotta, J.G. (Charles Schwab & Co. Inc., San Francisco, CA, USA); McGrew, J.E., 1998,** A Guttman scaling of CMM Level 2 practices: investigating the implementation sequences underlying software engineering maturity, Empirical Software Engineering, v **3**, n 2, p 159-77.

[23] **Miller, M.J.; Pulgar-Vidal, F.; Ferrin, D.M.;** 2002, Achieving higher levels of CMMI maturity using simulation Simulation Conference. Proceedings of the Winter Volume **2,** Page(s):1473 - 1478 vol.2

[24] **Fredrik Ekdahl; Stig Larsson;** 2006, Experience Report: Using Internal CMMI Appraisals to Institutionalize Software Development Performance Improvement; Software Engineering and Advanced Applications, 2006. SEAA '06. 32nd EUROMICRO Conference, Page(s):216 – 223

[25] **David E. Drehmer and Sasa M. Dekleva**., 2001, A note on the evolution of software engineering practices; *Journal of Systems and Software*, Volume **57**, Issue 1, Pages 1-7

[26] **Yu-Whoan Ahn (Software Eng. Dept., Syst. Eng. Res. Inst., Taejeon, South Korea); Gil-Jo Kim; Ja-Kyong Koo; Hyun-Min Park; In-Geol Chun**; 1998, Design of knowledge-based integrated software process improvement tools, SMC'98 Conference Proceedings. 1998 IEEE International Conference on Systems, Man, and Cybernetics (Cat. No.98CH36218), pt. 3, p 2132-7 vol.**3**

[27] **Sharp, H.; Woodman, M.; Hovenden, F.; Robinson, H.;**., The role of `culture' in successful software process improvement , EUROMICRO Conference, 1999. Proceedings. 25th Volume 2, 8-10 Sept. 1999 Page(s):170 - 176 vol.2

[28] **James YLThong** An integrated model of information systems adoption in small businesses ; Journal of Management Information Systems; Spring 1999; 15, 4; ABI/INFORM Global Pages. 187

[29] **Margaret K. Kulpa, Kent A. Johnson**., 2003, Interpreting the CMMI, Auerbach Publications ISBN:0-8493-1654-5

[30] **Michael West,** 2004, Real Process Improvement using the CMMI, Auerbach Publications ISBN: 0-8493-2109-3

[31] **Dean Leffingwell, Don Widrig**., May 05 2003 , Managing Software Requirements: A Use Case Approach, Second Edition Publisher: Addison Wesley Pub ISBN: 0-321-12247-X Pages: 544

[32] **Karl E. Wiegers.**, 2003, Software Requirements, Second Edition, ISBN:0735618798Microsoft Press Pages:400-410

[33] **Howard Podeswa**., UML for the IT Business Analyst: A Practical Guide to Object-Oriented Requirements Gathering, Thomson Course Technology

[34] **David E. Bellagio, Tom J. Milligan**., May 23, 2005, Software Configuration Management Strategies and IBM® Rational® ClearCase® Second Edition A Practical Introduction. Publisher: Addison Wesley Professional Print ISBN: 0-321-20019-5 Pages: 384

[35] **Paul Goodman,** 2004, Software Metrics: Best Practices for Successful IT Management, ISBN:1931332266 Rothstein Associates Pages: 120-125

BIOGRAPHY

Didem Kökten earned her BS degree in Computer Engineering in 2000 from Istanbul Technical University (ITU). Her professional career started as a technical assistant in ITU computer labs, and continued in TUBITAK (Turkish National Science & Research Institute) throughout school years. After college, Didem started in KocBryce as a certified Sun Microsystems trainer on subjects such as Java, Solaris operating system and Network Administration. She resumed her professional career as a software support specialist, focusing on training as well as implementation of software engineering methodologies and process management practices. She is currently with Telenity, an international telecommunications software vendor, managing the company-wide CMMI initiative. Didem continues her masters degree in Istanbul Technical University on CMMI. She is also a part-time consultant on process management and CMMI with Mentor Project Management, Training and Consultancy.