



Middle East University for Graduate Studies

***Department of Computer Science
Faculty of Information Technology***

**A Customer-Oriented
Software Development Life Cycle**

***Yazan Al-Masaf'ah
(20060096)
May, 2008***

**Supervisor
*Prof. Ali Meligy***

A Customer-Oriented Software Development Life Cycle

By
Yazan Omar Al-Masa'fah
(20060096)

*Thesis Presented to the Faculty of Information Technology
Of the Middle East University for Graduate Studies
In Partial Fulfillment of the Requirements
for the Degree of*

*Masters of Science
In
Computer Science*

Supervisor
Prof. Ali Meligy

MEU, Amman-Jordan
The 25th of May, 2008

Copyright 2008, Yazan Al-Masa'fah

Authorization

I am Yazan Omar Al-Masa'fah authorizes the **Middle East University for graduate studies** to supply a copy of this thesis to the library or establishments or individual(s) upon request.

Signature:



26/5/2008

تفويض

أنا المدعو يزن عمر المساعدة أفوض جامعة الشرق الأوسط للدراسات العليا بتزويد نسخة من هذه الرسالة إلى المكتبة أو المؤسسات أو الأشخاص كما تراه مناسباً.



التوقيع:

2008\5\26

Committee Decision

This thesis (A Customer-Oriented Software Development Life Cycle) was successfully defended and approved on May 26th, 2008.

Examination Committee Signatures

Prof. Ali Meligy

Signature:



Prof. Mohammed AL-Haag Hasan

Signature:



Dr. Nidal Shilbayeh

Signature:



Prof. Asem Al-Shekh

Signature:



Dedication

To my father, Omar Al-Masa'fah, for being my idol...

To my Family, Eman, Mahmoud Kiswani, Nuha, Abdel-Rahman, Meis, Rayah, Bashar, Eman, Muhannad, Ruqaya and leen, for their unconditional Love...

To my wife, Amani, for being my sunshine...

Acknowledgements

I would like to express my deep appreciation to my supervisor, Prof. Ali Meligy for his time, patience, and understanding. Thanks go also to the academic staff of the department of computer science in Middle East University for their support; there are not enough words to describe their excellent work.

My gratitude to Dr. Omar Al-Masa'fah, for his advices. I express my deep appreciation to Omar Hunaty, Ahmad Tarabish, Ziad Masa'fah, Mu'taz, Suhib, bilal and Obada Hararah for their encouragement. In addition, I want to thank my colleagues at the Middle East University for their continuous help.

Table of Contents

Authorization	iii
Committee Decision	iv
Dedication.....	v
Acknowledgements.....	vi
Table of Contents	vii
Table of Figures.....	ix
Abbreviations	x
Abstract	xi
Chapter 1: Introduction.....	1
1.1 Software Engineering.....	1
1.2 Software Engineers Role.....	2
1.3 Main Software Process Models.....	3
1.3.1 <i>The Waterfall Model</i>	4
1.3.2 <i>Rapid Application Development Model</i>	5
1.3.3 <i>Evolutionary Development (The Prototyping Model)</i>	7
1.3.4 <i>The Incremental Model</i>	8
1.3.5 <i>The Spiral Model</i>	9
1.3.6 <i>Formal Systems Development Model</i>	11
1.3.7 <i>Agile Development Methods</i>	12
1.3.8 <i>Component-Based Development</i>	13
1.4 The Problem Definition.....	14
1.4.1 <i>Statement of The Problem</i>	14
1.4.2 <i>Goals</i>	14
1.4.3 <i>Why This Topic?</i>	15
1.4.4 <i>Methodology</i>	15
1.4.5 <i>The Model Initial Assumptions and Expected Results</i>	15
Chapter 2: Literature Review and Related Work	17
2.1 Agile Methods.....	17
2.1.1 <i>Extreme Programming</i>	17
2.1.2 <i>Scrum</i>	18
2.1.3 <i>Feature Driven Development</i>	19
2.1.4 <i>The Rational Unified Process</i>	20
2.1.5 <i>Dynamic System Development Method</i>	20
2.1.6 <i>Adaptive Software Development</i>	21
2.2 Joint Application Design.....	22
2.3 The Modular-Model.....	23
2.4 The Behavior Tree.....	24
2.5 Project Management.....	27
2.6 Projects Documentation.....	29
2.6.1 <i>The Project Charter</i>	29
2.6.2 <i>The Feasibility Study</i>	30
2.6.3 <i>Software Requirements Specification</i>	31
Chapter 3: Customer Participation.....	33
3.1 Introduction.....	33
3.2 Customer Involvement in Software Production Phases.....	34
3.2.1 <i>Software Specifications (Requirement engineering)</i>	34

3.2.2	<i>Software Design and Implementation</i>	36
3.2.3	<i>Software Verification and Validation</i>	37
3.2.4	<i>Software Evolution and Maintenance</i>	39
3.2.5	<i>Software Project Management</i>	40
Chapter 4: A Customer Oriented Software Development Life Cycle		41
4.1	The Model Main Phases	41
4.1.1	<i>Customer Preparation Phase</i>	42
4.1.2	<i>Requirement Engineering Phase</i>	43
4.1.3	<i>Design and Development Phase</i>	45
4.1.4	<i>Testing Phase</i>	47
4.1.5	<i>Closure Phase</i>	49
4.2	The Dataflow Model (The Model Main Tool)	49
4.3	The Workflow Model (Transaction Between Phases)	51
4.4	The Role Model (Major Roles and Responsibilities)	53
4.5	The Model Practices	55
4.6	The Model Documents	56
Chapter 5: Discussion		58
5.1	Introduction	58
5.2	The Model Classification	59
5.3	Comparison With Other Models	61
Chapter 6: Contributions, Conclusions and Future Work		64
6.1	Model Analysis	64
6.1.1	<i>A Five-Phases Model</i>	64
6.1.2	<i>A Tree-Based Data flow</i>	65
6.1.3	<i>A Call-Based Work Flow</i>	65
6.1.4	<i>Interactive Customer and Supplier Teams</i>	66
6.2	The Model Effectiveness	66
6.2.1	<i>CHAOS Report Confirmation</i>	67
6.2.2	<i>CMMI Model Measurement</i>	70
6.3	The Model Drawbacks	72
6.4	Conclusions and Future Work	72
References		74
Appendices		A1
Appendix A1: Behavior Tree Notation		A1
	<i>A1.1 Naming Conventions</i>	A1
	<i>A1.2 Behavior Tree Notation</i>	A3
Appendix A2: Curriculum Vitae		A6

Table of Figures

Figure 1: Build-And-Fix model	3
Figure 2: The waterfall model (Somerville 2004)	4
Figure 3: Rapid application development model (Somerville 2004).....	6
Figure 4: Evolutionary development model (Somerville 2004).....	7
Figure 5: Incremental model (Somerville 2004)	8
Figure 6: The spiral model	10
Figure 7: Formal system development model (Somerville 2004)	11
Figure 8: Formal transformation (Somerville 2004).....	12
Figure 9: Agile development basic principles (Somerville 2004).....	12
Figure 10: Component-Based development process.....	13
Figure 11: XP process (Abrahamson, et al. 2002).....	18
Figure 12: Scrum process (Abrahamson, et al. 2002).....	19
Figure 13: FDD process (Abrahamson, et al. 2002)	19
Figure 14: RUP phases (Abrahamson, et al. 2002).....	20
Figure 15: DSDM process (Abrahamson, et al. 2002).....	21
Figure 16: The ASD life cycle (Abrahamson, et al. 2002).....	21
Figure 17: JAD phases (Jennerich 1990)	23
Figure 18: The Modular Model (Maheswar 2002),	24
Figure 19: Behavior tree notation key elements (Powell 2007)	25
Figure 20: Translation of natural language to a behavior tree (Powell 2007).....	25
Figure 21: Behavior trees naming convention (Behavior Tree Group 2007),.....	26
Figure 22: The project management triangle.....	28
Figure 23: Difference between the real needs and the stakeholders needs	34
Figure 24: Software processes common activities.....	34
Figure 25: The requirement engineering process (Somerville 2004)	35
Figure 26: Software design and development stages	37
Figure 27: The Lifecycle of verification approach (Boehm 1996).....	38
Figure 28: The IEEE maintenance process (Canfora and Cimitile 2000).....	40
Figure 29: The proposed software development life cycle	41
Figure 30: The customer preparation phase	42
Figure 31: Requirement engineering process	44
Figure 32: Design and development phase activities.....	47
Figure 33: Testing phase and its interaction with other phases.....	48
Figure 34: The model behavior tree parts	50
Figure 35: Data flow of a sub tree and a single node.....	51
Figure 36: The calling system.....	51
Figure 37: The role and communication model.....	55
Figure 38: CHAOS 2004 projects resolution	67
Figure 39: Change in projects resolution (1994-2004)	68
Figure 40: Average percentage of cost overrun (1994-2004)	68
Figure 41: Average percentage of time overrun (1994-2004).....	69
Figure 42: An overview of the software CMMI levels (Paulk 2001)	71
Figure 43: Target Profiles and Equivalent Staging (Paulk 2001).....	71
Figure 44: Analysis of the developed model based on the CMMI	72

Abbreviations

ASD	<i>Adaptive Software Development</i>
BT	<i>Behavior Tree</i>
CBD	<i>Component-Based Development</i>
DBT	<i>Design Behavior Tree</i>
DSDM	<i>Dynamic System Development Method</i>
FDD	<i>Feature Driven Development</i>
IDL	<i>Interface Description Language</i>
IM	<i>Incremental Model</i>
IS	<i>Information Services</i>
JAD	<i>Joint Application Design</i>
MSC	<i>Message Sequence Chart</i>
PM	<i>Project Manager</i>
RAD	<i>Rapid Application Development</i>
RBT	<i>Requirements Behavior tree</i>
ROI	<i>Return on Investment</i>
RUP	<i>Rational Unified Process</i>
SDLC	<i>Software Development Life Cycle</i>
SE	<i>Software Engineering</i>
SPR	<i>Software Problem Reports</i>
SRS	<i>Software Requirements Specification</i>
SVVP	<i>Software Verification and Validation Plan</i>
V&V	<i>Validation and Verification</i>
XP	<i>Extreme Programming</i>

Abstract

Software production is considered to be one of the largest industries in the 21th century; any study that leads to increasing the efficiency of this industry could have tremendous effect on the world technology revolution.

The main purpose of software development is supporting the business functions of clients each in his field. Hence, this study introduced a software development model that is oriented to increasing customer involvement in each phase of the software development life cycle, from project initiation to completion, which –as hoped- will enhance customer satisfaction and the quality of the delivered software.

The goal of this thesis is developing a software life cycle that involves the customer frequently and effectively in projects. To achieve this, we discussed some of the existing software methodologies, aiming to find some activates that proved to be effective in enhancing the customer role in projects. The thesis discussed the customer role in each of the main phases of software development and the importance of the customer effective participation in software development.

We introduced through out this study a five phases model focusing on achieving an end-to-end life cycle that is oriented in increasing customer participation. Along side with the model, some supporting flows where proposed to enhance the model ability, including, a dataflow model to control the flow of data in each phase, a workflow model to describe the transaction between the model phase, and a role model to govern the personnel participation and roles.

We believe that our model is capable of achieving its main goal, but in order to give a realistic assessment of the effectiveness of the proposed software development life cycle, the model must be adopted by software engineers and project managers in the field, to verify its ability on the ground.

Yazan Al-Masa'fah
Supervisor: Prof. Ali Meligy

المُلخَص

تُعد عملية إنتاج البرمجيات واحدةً من أكبر الصناعات في القرن الحادي والعشرين، وأي دراسةٍ تؤدي إلى زيادة فعالية هذه الصناعة يمكن أن تترك أثراً هائلاً على ثورة العالم التكنولوجية.

إن الهدف الرئيس لعملية إنتاج البرمجيات هو دعم الأهداف التجارية للمستخدمين كل في حقله. وعليه، تقدم هذه الدراسة أنموذج نظام برمجي، موجه لزيادة تفاعل المستخدم مع كل مرحلةٍ من دورة حياة النظام البرمجي، منذ انطلاقة المشروع وحتى نهايته، و الذي سيؤدي -كما يؤمل- إلى إرضاء المستخدم وتحسين نوعية البرمجية المسلمة.

إن الهدف من هذه الرسالة هو تطوير دورة حياة برمجية تقوم بإقحام المستخدم باستمرار وبفعالية في المشاريع. لذلك قمنا بمناقشة بعض المنهجيات المستخدمة حالياً، بهدف إيجاد بعض الفعاليات التي أثبتت الكفاءة فيما يتعلق بتفعيل دور المستخدم في المشاريع. واستعرضت الرسالة دور المستخدم في كل المراحل الرئيسة في عملية تطوير البرمجيات وبينت أهمية المشاركة الفعالة للمستخدم في تطوير البرمجية.

قدمنا من خلال هذه الدراسة أنموذجاً من خمسة مراحل بهدف الحصول على دورة حياة مكتملة وموجهة لزيادة دور المستخدم. بالإضافة لذلك، تم عرض بعض التدفقات المساندة لتحسين قدرة الأنموذج، من ضمنها، أنموذج لتدفق المعلومات لتنظيم آلية انتقال المعلومات في كل مرحلة، وأنموذج عمل لتوضيح كيفية الانتقال من مرحلة لأخرى، بالإضافة لأنموذج للأدوار لإدارة عمل الأطراف المستخدمة وتوضيح أدوارها.

نحن نعتقد أن الأنموذج المطور قادر على تحقيق الأهداف التي أوجد من أجلها، ولكن من أجل توفير تقييم موضوعي لفعالية دورة الحياة البرمجية هذه، يجب ان يتم تبني الأنموذج من مهندسي البرمجيات ومدراء المشاريع العاملين في هذا الحقل، وذلك لقياس قدرة الأنموذج على أرض الواقع.

يزن المساعفة

إشراف: أ.د. علي مليجي

Blank

Chapter 1

Introduction

Computer Software has become a driving force. It is a tool that drives business decision making. It serves as the basis for modern scientific investigation and engineering problem solving. It is a key factor that differentiates modern products and services. Software is embedded in systems of all kinds: transportation, medical, telecommunications, military, industrial process, entertainment, office products and the list almost go endless. Software is virtually inescapable in a modern world. It is the driver for new advances in everything from elementary education to genetic engineering.

In essence, software affects nearly every aspect of our lives either directly or indirectly. As long as software continues to be intricately linked to commerce and culture the need for software engineering will exist.

1.1 Software Engineering

The field of software engineering (SE) can intuitively be described as the combination of techniques from both the engineering discipline and all aspects of software production. This includes all of the development stages from system specification to maintenance and possibly retirement. Alternatively, it may be defined as the “establishment and use of sound engineering principles in order to obtain economically software that is reliable and works efficiently on real machines.” (Pollice 2005)

Computer software can then be defined as the product that software engineers design and build. It includes the executable programs, documentation (both electronic and hard copy). In addition, it may also include data in the form of numbers and text, or even pictorial and multimedia formats. While Engineering stands for the analysis, design, construction, verification and management of technical (or social) entities.

In the early years of software development programs were relatively small as they were designed to perform a specific function that was often limited in scope and tied to a given platform. However over the last fifty years there has been a dramatic advancement in the technology sector leading to improvements in hardware performance and profound changes in computing architectures. These advances along with the vast increase in memory and storage capacities have all combined to produce complex computer-based systems that are capable of providing information in a wide variety of formats.

The introduction of third-generation computer hardware initially led to what is termed the “software crisis”. Basically, the dramatic increase in computer power made seemingly unrealistic computer applications a feasible proposition, marking the genesis of the era where software products that were magnitudes of order more complex than their predecessors. This increased sophistication carried with it the inherent possibility of hefty problems for a single programmer.

Inexperience with creating software on this scale often led to informal approaches being adopted which resulted in software that was over budget, delivered late, unreliable as well as difficult to operate and maintain.

An increasing importance was placed on the programmer's ability to answer questions like, why does it take so long to get software finished? Why are the development costs so high? Why can not we find all the error before the software is released? And why is there difficulty in measuring progress as the software is being developed?

As with any entity carrying possible financial benefits, whether the profit generating or loss limiting, software production needed to be optimized. Typically, a team of software specialists is employed to tackle the complexity issue. However, as the scope is often large and intricate a structured approach is required and a standard must also be maintained so that, for example, in the event of any staffing changes, continuity would not be severely affected. The control, organization and stability offered by a structured approach are crucial for the successful development a good software product.

Software engineering -in principle- is concerned with four main parts:

- The customer: The individual or organization for which the product is developed.
- The supplier: The individual(s) or organization(s) responsible for the production of the required software.
- The user: The person(s) who use the software.
- Software development: Covers all aspects of software production before the product enters the maintenance phase.

1.2 Software Engineers Role

Software engineering is often described as a layered technology where the emphasis placed on quality. The foundation of SE includes a process, management, technical methods and tools. In essence, the process establishes (<http://En.Wikipedia.Org/>):

- The framework for management control of the software project.
- The mechanism by which scheduling is maintained and quality is ensured
- The proper management of change
- The context in which technical methods are applied
- The appropriate tools for a project

Methods provide the technical information on stages required to successfully build the software product. This ranges from the embryonic stages of development to the maintenance stages. Depending on the given stage, models and various other forms of documentation are required.

Tools -on the other hand- provide automated or semi-automated support for the process and methods. The software engineer should have a global view of the production procedure. That is he should be aware of:

- The problem to be solved
- The complete objective of the final product

- The means by which the final product will be built and the tools required - strategy
- The design, testing and maintenance considerations

1.3 Main Software Process Models

The software process model maybe defined as a simplified description of a software process, presented from a particular perspective (Somerville 2004). In essence, each stage of the software process is identified and a model is then employed to represent the inherent activities associated within that stage. Consequently, a collection of 'local' models may be utilized in generating the global picture representative of the software process. Examples of models include the workflow model, the data-flow model, and the role model (<http://En.Wikipedia.Org/>).

- The workflow model: shows the sequence of activities in the process along with their inputs, outputs and dependencies. The activities in the model represent human actions.
- The dataflow model: represents the process as a set of activities each of which carries out some data transformation. It shows how the input to the process such as specification is transformed to an output such as design. The activities here maybe lower than in a workflow model. They may represent transformations carries out by people or computers.
- The role model: represents the roles of people involved in the software process and the activities for which they are responsible.

In the early days of software revolution, software were produced following a “very simple” model called Build-and-Fix model, shown in figure 1. In this model, the product is built without proper specifications and design steps. Essentially, the product is built and modified as many times as possible until it satisfies the customer. The cost of using this approach is greater than if specifications are drawn up and a design is carefully developed. Software engineers are strongly discouraged from using this development approach since it is the worst model for developing a project.

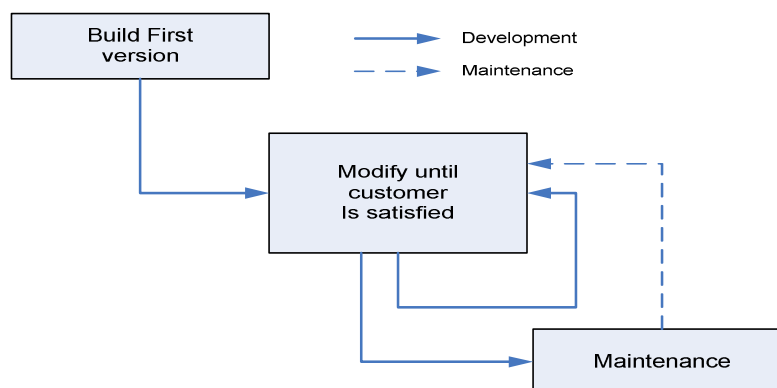


Figure 1: Build-And-Fix model

The traditional generic software process models that effectively demonstrate the different approaches to software development are:

- Waterfall Model
- Rapid Application Development (RAD) Model

- Prototyping (Evolutionary) Model
- Incremental Model
- Spiral Model
- Formal Systems Development Model
- Agile development
- Component-Based Development.

1.3.1 *The Waterfall Model*

The waterfall model derives its name due to the cascading effect from one phase to the other as is illustrated in figure 2. In this model each phase has well defined starting and ending point, with identifiable deliveries to the next phase.

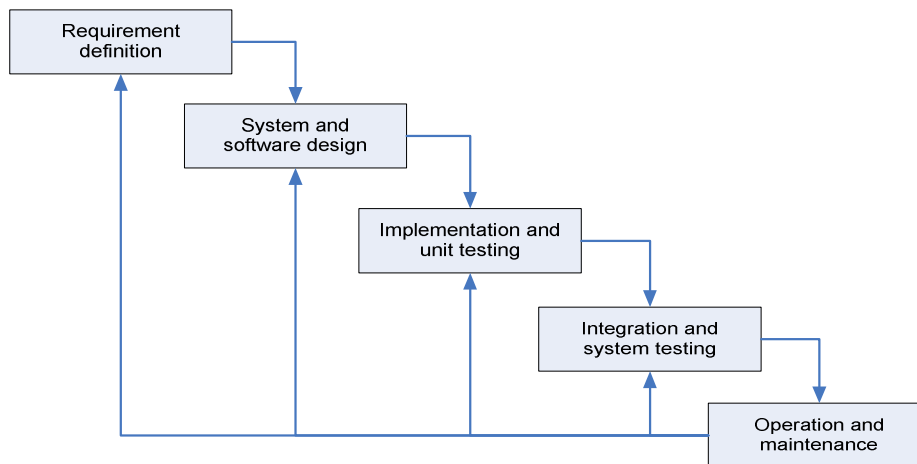


Figure 2: The waterfall model (Somerville 2004)

The model consists of five distinct stages, namely (Somerville 2004):

1. The requirements definition phase:
 - a. The problem is specified along with the desired service objectives (goals).
 - b. The constraints are identified.
 - c. System specification document is produced from the detailed definitions of (a) and (b). This document should clearly define the product function.
2. In the system and software design phase, the system specifications are translated into a software representation. The software engineer at this stage is concerned with:
 - a. Data structure
 - b. Software architecture
 - c. Algorithmic detail and
 - d. Interface representations
 - e. The hardware requirements
 - f. The overall system architecture.

By the end of this stage the software engineer should be able to identify the relationship between the hardware, software and the associated interfaces. Any faults in the specification should ideally not be passed down stream.

3. In the implementation and testing phase stage the designs are translated into the software domain
 - a. Detailed documentation from the design phase can significantly reduce the coding effort.
 - b. Testing at this stage focuses on making sure that any errors are identified and that the software meets its required specification.
4. In the integration and system testing phase all the program units are integrated and tested to ensure that the complete system meets the software requirements. After this stage the software is delivered to the customer – i.e. the software product is delivered to the customer for acceptance testing.
5. The maintenance phase the usually the longest stage of the software. In this phase the software is updated to:
 - a. Meet the changing customer needs
 - b. Adapted to accommodate changes in the external environment
 - c. Correct errors and oversights previously undetected in the testing phases
 - d. Enhancing the efficiency of the software

The feed back loops in the model allow for corrections to be incorporated into the model. For example a problem/update in the design phase requires a revisit to the specifications phase. When changes are made at any phase, the relevant documentation should be updated to reflect that change.

The main Advantages of the waterfall model are:

- Testing is inherent to every phase of the waterfall model
- It is an enforced disciplined approach
- It is documentation driven, that is, documentation is produced at every stage

The waterfall model is the oldest and the most widely used paradigm. However, many projects rarely follow its sequential flow. This is due to the inherent problems associated with its rigid format. Namely:

- It only incorporates iteration indirectly, thus changes may cause considerable confusion as the project progresses.
- As The customer usually only has a vague idea of exactly what is required from the software product, this model has difficulty accommodating the natural uncertainty that exists at the beginning of the project.
- The customer only sees a working version of the product after it has been coded. This may result in disaster if any undetected problems are precipitated to this stage.

1.3.2 *Rapid Application Development Model*

Rapid Application Development (RAD) is an incremental software development process model that emphasizes a very short development cycle. The RAD model,

shown in figure 3, is a high-speed adaptation of the waterfall model, where the result of each cycle provides a fully functional system (Somerville 2004).

- The processes of specification, design and implementation are concurrent. There is no detailed specification and design documentation is minimized.
- The system is developed in a series of increments. End users evaluate each increment and make proposals for later increments.
- System user interfaces are usually developed using an interactive development system.

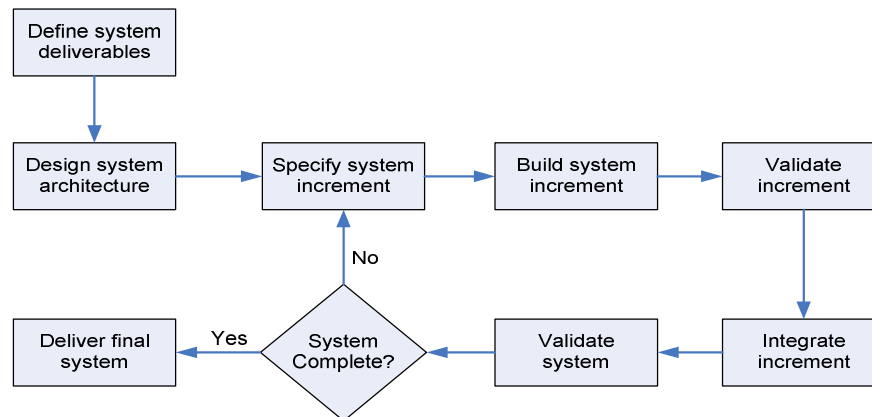


Figure 3: Rapid application development model (Somerville 2004)

The main Advantages of RAD include:

- Accelerated delivery of customer services. Each increment delivers the highest priority functionality to the customer.
- User engagement with the system. Users have to be involved in the development which means the system is more likely to meet their requirements and the users are more committed to the system.

On the other hand RAD faces many problems such as,

- Management problems: Progress can be hard to judge and problems hard to find because there is no documentation to demonstrate what has been done.
- Contractual problems: The normal contract may include a specification; without a specification, different forms of contract have to be used.
- Validation problems: Without a specification, what is the system being tested against?
- Maintenance problems: Continual change tends to corrupt software structure making it more expensive to change and evolve to meet new requirements.

If a business application can be modularized so that each major function can be completed within the development cycle then it is a candidate for the RAD model. In this case, each team can be assigned a model, which is then integrated to form the whole product.

Disadvantages of RAD contain:

- For large -but scalable- projects, RAD requires sufficient resources to create the right number of RAD teams.

- RAD projects will fail if there is no commitment by the developers or the customers to rapid activities necessary to get a system complete in a much abbreviated time frame.
- If a system cannot be properly modularized, building components for RAD will be problematic
- RAD is not appropriate when technical risks are high, e.g. this occurs when a new application makes heavy use of new technology.

1.3.3 Evolutionary Development (The Prototyping Model)

In many instances the customer only has a general view of what is expected from the software product. In such a scenario where there is an absence of detailed information regarding the input to the system, the processing needs and the output requirements, the prototyping model may be employed.

This model –shown in figure 4 (<http://www.cs.odu.edu>)- reflects an attempt to increase the flexibility of the development process by allowing the customer to interact and experiment with a working representation of the product. The developmental process only continues once the customer is satisfied with the functioning of the prototype. At that stage the developer determines the specifications of the customer's real needs.

There are two well-known approaches in this model. Throw-away prototyping uses the prototype as a means of quickly determining the needs of the customer; it is discarded once the specifications have been agreed on. The emphasis of the prototype is on representing those aspects of the software that will be visible to the customer/user. Thus it does not matter if the prototype hardly works.

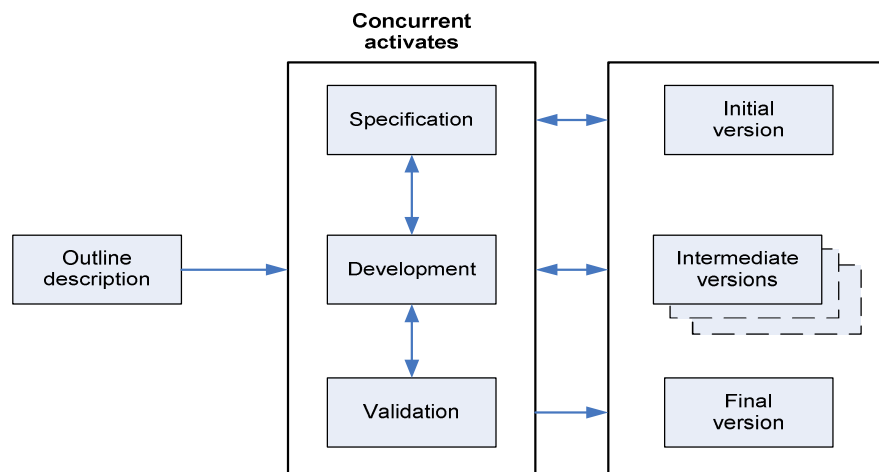


Figure 4: Evolutionary development model (Somerville 2004)

Alternatively, exploratory development uses the prototype as the specifications for the design phase. The advantage of this approach is speed and accuracy, as not time is spent on drawing up written specifications. The inherent difficulties associated with that phase (i.e. incompleteness, contradictions and ambiguities) are then avoided. The main objective is to work with customers and to evolve a final system from an initial

outline specification. It should start with well-understood requirements and add new features as proposed by the customer.

The main disadvantages of evolutionary development are:

- Customers often expect that a few minor changes to the prototype will be more sufficient to their needs. They fail to realize that no consideration was given to the overall quality of the software in the rush to develop the prototype.
- The developers may lose focus on the real purpose of the prototype and compromise the quality of the product. For example, they may employ some of the inefficient algorithms or inappropriate programming languages used in developing the prototype. This mainly due to laziness and an over reliance on familiarity with seemingly easier methods.
- A prototype will hardly be acceptable in court if the customer does not agree that the developer has discharged his obligations. For this reason using the prototype as the software specification is normally reserved for software development within an organization.

To avoid the above problems the developer and the customer should both establish a protocol, which indicates the deliverables to the customer as well as the contractual obligations.

1.3.4 *The Incremental Model*

The Incremental Model (IM), illustrated in figure 5, derives its name from the way in which the software is built. More specifically, the model is designed, implemented and tested as a series of incremental builds until the product is finished. A build consists of pieces of code from various modules that interact together to provide a specific function.

At each stage of the IM a new build is coded and then integrated into the structure, which is tested as a whole. Note that the product is only defined as finished when it satisfies all of its requirements.

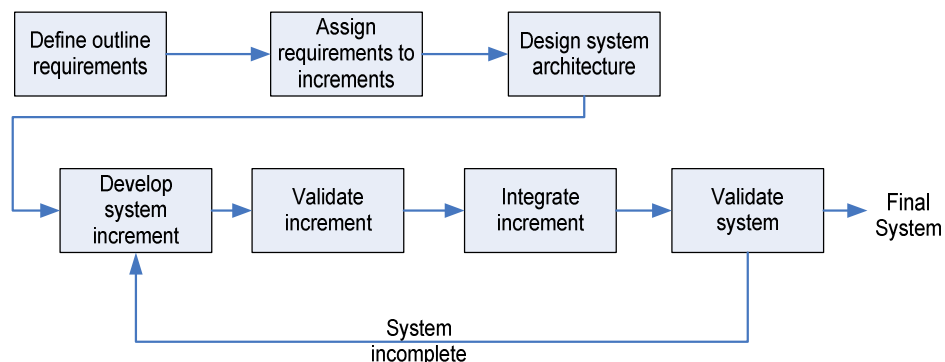


Figure 5: Incremental model (Somerville 2004)

This model combines the elements of the waterfall model with the iterative philosophy of prototyping. However, unlike prototyping the IM focuses on the delivery of an operational product at the end of each increment.

The first increment is usually the core product which addresses the basic requirements of the system. This may be either used by the customer or subjected to detailed review to develop a plan for the next increment. This plan addresses the modification of the core product to better meet the needs of the customer, and the delivery of additional functionality. More specifically, at each stage:

- The customer assigns a value to each build not yet implemented
- The developer estimates cost of developing each build
- The resulting value-to-cost ratio is the criterion used for selecting which build is delivered next

Essentially the build with the highest value-to-cost ratio is the one that provides the customer with the most functionality (value) for the least cost. Using this method the customer has a usable product at all of the development stages.

However, if the incremental model is inappropriate or misused, it has the following disadvantages:

- Fielding of initial increments may destabilize later increments through unplanned levels of user change requests.
- If requirements are not as stable or complete as thought earlier, increments might be withdrawn from service, reworked, and re-released.
- Managing the resulting cost, schedule, and configuration complexity may exceed the capabilities of the organization.
- Each phase of an iteration is rigid and do not overlap each other.
- Problems may arise pertaining to system architecture because not all requirements are gathered up front for the entire software life cycle.

1.3.5 *The Spiral Model*

Our understanding of the spiral model is illustrated in figure 6. The spiral model combines the iterative nature of prototyping with the controlled and systematic aspects of the waterfall model, therein providing the potential for rapid development of incremental versions of the software. In this model the software is developed in a series of incremental releases with the early stages being either paper models or prototypes. Later iterations become increasingly more complete versions of the product.

The model is divided into a number of task regions. These regions are:

1. Objective setting: Specific objectives for the phase are identified.
2. Risk assessment and reduction: Risks are assessed and activities put in place to reduce the key risks.
3. Development and validation: A development model for the system is chosen which can be any of the generic models.
4. Planning: The project is reviewed and the next phase of the spiral is planned.

The cycling process begins at the centre position and moves in a clockwise direction. Each traversal of the spiral typically results in a deliverable. For example, the first and second spiral traversals may result in the production of a product specification and a

prototype, respectively. Subsequent traversals may then produce more sophisticated versions of the software.

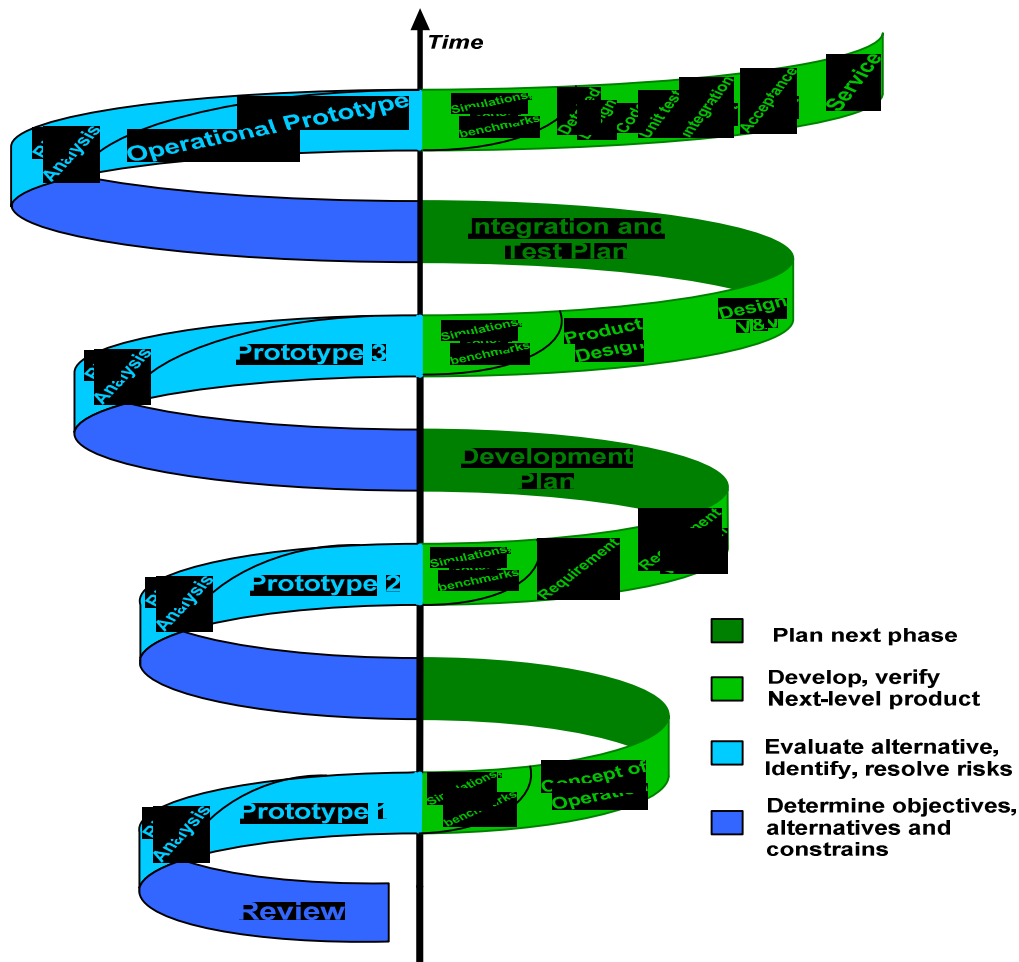


Figure 6: The spiral model

An important distinction between the spiral model and other software models is the explicit consideration of risk. There are no fixed phases such as specification or design phases in the model and it encompasses other process models. For example, prototyping may be used in one spiral to resolve requirement uncertainties and hence reduce risks. This may then be followed by a conventional waterfall development. It is important to note that:

- Each passage through the planning stage results in an adjustment to the project plan (e.g. cost and schedule are adjusted based on the feedback from the customer, project manager may adjust the number of iterations required to complete the software....)
- Each of the regions is populated by a set of work tasks called a task set that are adapted to characteristics of the project to be undertaken. For small projects the number of tasks and their formality is low. Conversely, for large projects the reverse is true.

Advantages of the spiral model include:

- The spiral model is a realistic approach to the development of large-scale software products because the software evolves as the process progresses. In

addition, the developer and the customer better understand and react to risks at each evolutionary level.

- The model uses prototyping as a risk reduction mechanism and allows for the development of prototypes at any stage of the evolutionary development.
- It maintains a systematic stepwise approach, like the classic life cycle model, but incorporates it into an iterative framework that more reflect the real world.
- If employed correctly, this model should reduce risks before they become problematic, as consideration of technical risks are considered at all stages.

While some of its disadvantages are:

- Demands considerable risk-assessment expertise
- It has not been employed as much proven models and hence may prove difficult to 'sell' to the customer, especially where a contract is involved, that this model is controllable and efficient.

1.3.6 Formal Systems Development Model

The formal systems development model, shown below in figure 7, utilizes a development process that is based on formal mathematical transformation of system models to executable programs. Similar to the waterfall model, the formal approach has clearly defined cascading phase boundaries. The critical distinctions between the two models are:

- The software requirements and specification phases are refined into a detailed formal specification, which is expressed mathematically.
- The design, implementation and unit testing are replaced by a single formal transformation phase as illustrated in figure 8.

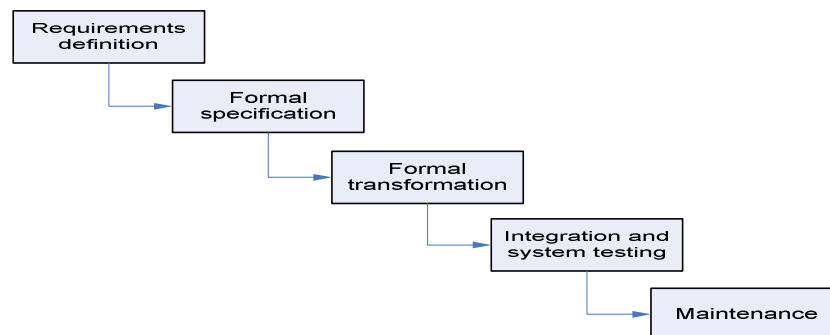


Figure 7: Formal system development model (Somerville 2004)

During the formal transformation process -shown in figure 8- the mathematical representation of the specifications is systematically refined. More specifically, at each transformation stage (T_x , $x = 1, 2, \dots, n$) more detail is added to produce a refined specification (R_x , $x = 1, 2, \dots, n$) until the formal specification is converted into the equivalent program.

Each transformation is made should be sufficiently close to avoid excessive verification efforts and reduce the possibility of transformation errors. In the absence of such errors the program would represent the true implementation of the specifications.

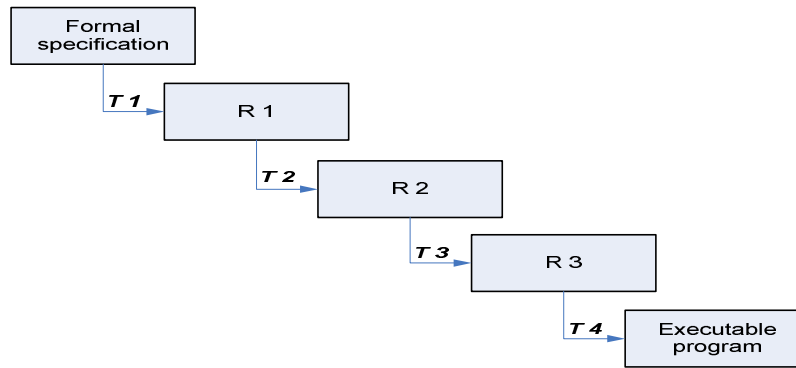


Figure 8: Formal transformation (Somerville 2004)

The formal systems development model is typically employed when developing systems that require strict safety, reliability and security requirements. However, the expertise required for the mathematical notations used for the formal specifications adds to the system development effort and cost making this model impractical for the development of other systems. Especially as there are no significant quality or cost advantages over other approaches.

1.3.7 Agile Development Methods

Agile methods are a set of development processes intended to create software in a lighter, faster, more people-centric way. Among these methods are extreme programming, scrum, dynamic systems development method, adaptive software development, and feature driven development. These methods:

- Focus on the code rather than the design.
- Are based on an iterative approach to software development.
- Are intended to deliver working software quickly and evolve this quickly to meet changing requirements.

Agile methods are probably best suited to small/medium-sized business systems or personal computer products. The basic principles of agile development are shown in figure 9.

Principle	Description
Customer involvement	The customer should be closely involved throughout the development process. Their role is provide and priorities new system requirements and to evaluate the iterations of the system.
Incremental delivery	The software is developed in increments with the customer specifying the requirements to be included in each increment.
People not process	The skills of the development team should be recognised and exploited. The team should be left to develop their own ways of working without prescriptive processes.
Embrace change	Expect the system requirements to change and design the system so that it can accommodate these changes.
Maintain simplicity	Focus on simplicity in both the software being developed and in the development process used. Wherever possible, actively work to eliminate complexity from the system.

Figure 9: Agile development basic principles (Somerville 2004)

The most obvious disadvantages of agile development methods are:

- It can be difficult to keep the interest of customers who are involved in the process.
- Team members may be unsuited to the intense involvement that characterizes agile methods.
- Prioritizing changes can be difficult where there are multiple stakeholders.
- Maintaining simplicity requires extra work.
- Contracts may be a problem as with other approaches to iterative development.

One of the open questions about agile methods is where the boundary conditions lie. One of the problems with any new technique is that the developer is not really aware of where the boundary conditions until they cross over them and fail. Agile methods are still too young to see enough action to get a sense of where the boundaries are. This is further compounded by the fact that it's so hard to decide what success and failure mean in software development, as well as too many varying factors to easily pin down the source of problems (Fowler 2005).

1.3.8 *Component-Based Development*

Component-Based Development (CBD) is a branch of the software engineering discipline, with emphasis on decomposition of the engineered systems into functional or logical components with well-defined interfaces used for communication across the components.

Recently, software component technology, which is based on building software systems from reusable components, has attracted attention because it is capable of reducing developmental costs. In a narrow sense, a software component is defined as a unit of composition, and can be independently exchanged in the form of an object code without source codes. The internal structure of the component is not available to the public.

The characteristics of the component-based development are the following:

- Black-box reuse
- Reactive-control and component's granularity
- Using RAD tools
- Contractually specified interfaces
- Introspection mechanism provided by the component systems
- Software component market (CALS)

Software components often take the form of objects or collections of objects (from object-oriented programming), in some binary or textual form, adhering to some interface description language (IDL) so that the component may exist autonomously from other components in a computer.

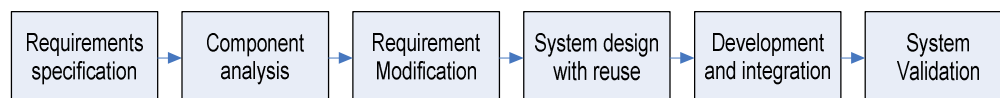


Figure 10: Component-Based development process

1.4 The Problem Definition

1.4.1 *Statement of The Problem*

The goal of any software development supplier is to satisfy customer needs in the shortest time and with a minimal implementation cost, which will lead to higher short and long term profit.

The main problem is that customers change their requirement too often and the supplier understanding of a customer requirement is not always right, which makes the product loops into iterations until it reaches a certain level that satisfy most of the customer needs. These iterations will lengthen the implementation time, and hence increase its cost, as well as, reducing customer and supplier profit.

This thesis is aiming to solve these problems by developing a new software life cycle that enhances the customer involvement in the process, and thus reducing the main effects that slow the software development process, and add to its cost.

If the customer is well involved in the software development process, he will have a clear understanding of the effect of any change in the requirements, and hence, he will make every effort to stop this change, or understand the reasons of delay in delivery due to this change. This will eventually lead to reduce unnecessary change of requirements from the customer side.

An onsite customer involvement in software development can enhance the speed of explaining requirements to the developers, as well as, detecting any misunderstood requirements and correct them. Thus, avoiding supplier misunderstanding of customer requirements.

To reduce the effect of iterations, the thesis aims to develop a model that handles the requirements in a modular fashion, where each individual or group of requirement is handled as a single module. These modules will then be developed, tested and combined together to form the final product. This will be governed by the customer and the supplier, as they will validate each requirement, selects the requirements to be developed, and test each individual requirement along side with testing the final product.

1.4.2 *Goals*

- Introducing a software development model that aims to increase customer involvement in each phase of the software development lifecycles, from project initiation to completion.
- Producing a model that focuses on involving the customer in the project management activities.
- Developing a model that leads to deliver software solutions that enhances customer satisfaction in addition to improving software quality and productivity.
- Combining the existing software methodologies that proved to be successful - such as extreme programming (Gray 2006); and organize them in a manner that will lead to enhance the whole software development life cycle.

- Building a model that handles the customer requirements in a modular fashion, where each individual or group of requirement is handled as a single module.

1.4.3 *Why This Topic?*

- Any study that leads to increasing the efficiency of software production industry could have tremendous effect on the world technology revolution (IMF World Economy Forecast Report 2002).
- Develop a software development processes that are adequate to fulfill the demand on software applications of greater size and complexity.
- Many researches have shown that customer involvement in the software process is crucial to achieving the required functionality. Engaging customers early and often on the software process will certainly improve project success probability (Standish Group 1995).
- Some of development organizations do not consult the customer, but instead isolate the choice of features to their business analyst or product marketing teams, which is usually the reason for the failure of projects (IBM Corporation 2005).

1.4.4 *Methodology*

In order to develop a software development life cycle that satisfies the main objectives of this thesis, we will start by looking for the existing software methodologies that proved to be successful in terms of customer participation.

Next, we will start building our model from designing the main architecture and defining the phases and identifying the key activities in each of these phases; then we will go down to develop processes that make the model running including a dataflow model, a workflow model and a role model. We will then define the main documents that should be generated through and as a result of the project.

Finally, an evaluation as well as a measure of the model will be carried out to conclude our study.

1.4.5 *The Model Initial Assumptions and Expected Results*

The proposed model aims to enhance the customer contribution to the development process, as well as trying to increase the customer side participation in project management activities. The following are initially assumed:

- The supplier participation in the development process is more than the customer.
- The leading role in the process should be in the hand of supplier.
- Overall management of the project is led by the supplier; customer participation will be in some –but not all– of the project management activities.
- The customer and supplier will take part in each phase of the software lifecycle process.

Our aim is to introduce a model that shows the communication methodology between the customer and the supplier in each phase of the software development life cycle. The model expected results include:

- The feasibility study will increase the customer and supplier interaction, in order to identify the customer request, and to make sure if the idea is feasible and can be satisfied using the existing software and hardware technologies.
- Requirements elicitation and analysis is among the most communication-rich processes of software development. The model will engage different stakeholders, from both the customer and the developer sides, who need to intensively communicate and collaborate.
- The requirement validation phase will attempt to increase confidence that a given requirement corresponds to the end-customer's desires, it is concerned with showing that the requirements define the requested system with no conflicts, contradiction, errors and omissions.
- Increase the direct contact between the developer and the customer in the development phase.
- Define the software verification and validation activities in a software verification and validation plan (SVVP). Customers, managers and developers all need to assure that the software does what it is supposed to do.
- Define the communication channel during the Acceptance test plan between the customer and the supplier.
- Define the project management activities that the customer can participate in. Among these activities are:
 - Schedule/Time management
 - Cost Management
 - Quality Management
 - Human Resource Management
 - Contract/Procurement Management
 - Communications Management
 - Scope Management
 - Risk Management

Chapter 2

Literature Review and Related Work

The proposed software engineering life cycle will use some of the utilities and methods that are used in other software engineering models and proved to be successful. We organized them in a manner that will enhance the efficiency of the proposed model. The model relies on the ideas provided by the below studies as a guide for its development.

2.1 Agile Methods

In the past few years there's been a blossoming of a new style of software methodology - referred to as agile methods. Agile software development has strong relation with the desired model since it focuses on customer participation as well as individuals communication. Hence, Major agile development methodologies along side with the recent studies on them, are described in detail in this section.

2.1.1 *Extreme Programming*

Extreme Programming (XP) promotes radical changes in how software development organizations traditionally work (Talby et al. 2006). It has been evolved from the problem caused by long development cycles of traditional development (Beck, K 1999a), based on practices that had been found effective in software development in the past years. XP is a lightweight process that provides principles for guiding projects and relies on the participants for its success (Gray 2006).

XP is characterized by pair programming (all production code is written by two people at one computer), rapid development iterations and releases, on-site customer involvement, a “test-first” approach to development, collective ownership of all code and an open team room workspace. Unlike traditional software development methodologies in which solo programmers can be found, team involvement is crucial in Extreme Programming. Team members support each other, learn from each other and feed creatively off each other.

Another difference between traditional software methodologies and XP is the style of communication. While Traditional software development approaches are characterized by paper communication: each development phase typically concludes with the production of a document; XP –on the other hand- is based on human communication, where it is important for software developers who employ the XP techniques, to be able to keep each other informed, resolve issues as they arise, interact productively with customers and generally communicate effectively (Johnson 2001). XP embraces both communication and feedback as interdependent process values which are essential for projects to achieve successful results (Korkala, et al. 2006).

The life cycle consists of five phases shown in figure 11. The exploration phase in which the customers define the requirement (stories) they want to be included in the first release, at the same time, the project team familiarize themselves with the technology to be used and the architecture possibilities by building a prototype of the system.

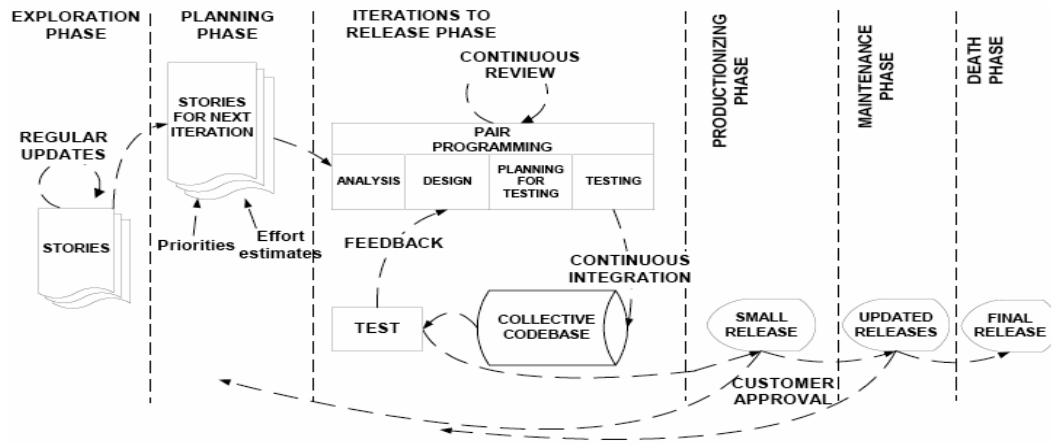


Figure 11: XP process (Abrahamson, et al. 2002)

The Iteration phase includes several iterations before the first release. The schedule set in the planning phase is broken down into a number of iterations. The customer decides the stories to be implemented in each iteration. At the end of the last iteration the system is ready for production.

In the maintenance phase XP team must keep the product running along side with building new iterations, this phase requires efforts also for the customer support tasks.

The XP team consists of programmer, customer, tester, tracker, coach, consultant and manager. XP requires that customer has to be present and available full-time for the team

2.1.2 Scrum

The idea of Scrum was presented in (Takeuchi, et al. 1986). Scrum concentrates on how the team members should function in order to produce the flexibly in a constantly changing environment (Takeuchi, et al. 1986). Scrum process includes three phases, as shown in figure 12.

The pre-game phase is divided into two sub-phases, the panning and architecture phase in which a product Backlog list is created that contains all the requirements that are currently known. The requirements can originate from the customer or the software developer. This Backlog is continuously updated based on the ongoing iterations. The requirements are prioritized and their needed effort is estimated in this phase.

The post-game phase contains the closure of the release, it is entered when the customer and the supplier agrees that the requirements are completed, after this point, no changes are allowed.

Roles in the Scrum are distributed between Scrum master, product owner, scrum team, customer and management (Schwaber, et al. 2002). Customer participation is on the tasks related to the Backlog items for the system being involved or enhanced. It has been noted that Scrum teams with Scrum Masters seems to scale naturally

especially where strong technical leadership is applied (British Broadcasting Corporation 2007).

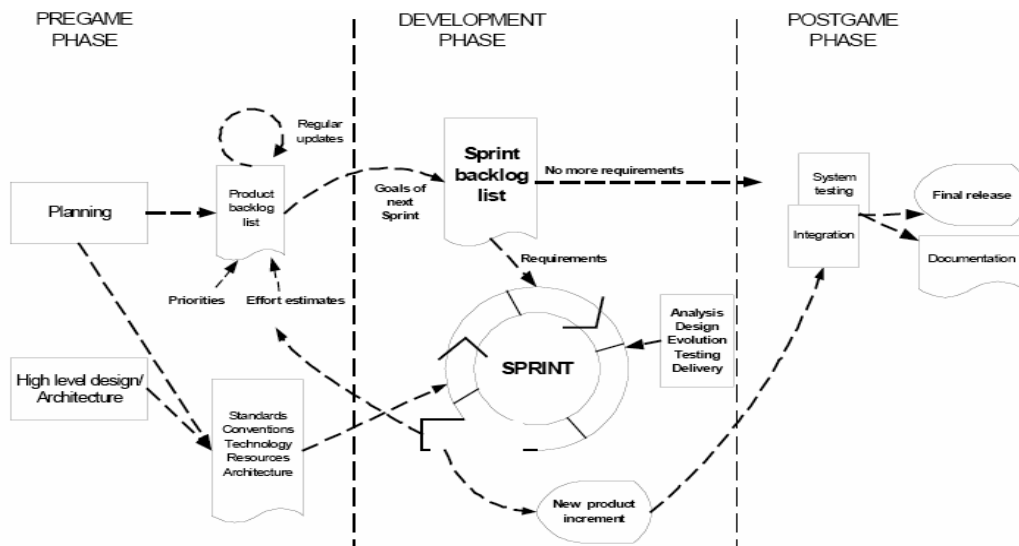


Figure 12: Scrum process (Abrahamson, et al. 2002)

2.1.3 Feature Driven Development

Feature Driven Development (FDD) was first reported by (Coad, et al. 2000), it does not cover the entire software development process, but rather focuses on the design and building phases (Palmer, et al. 2002). FDD consists of five sequential processes and provides the methods, techniques and guidelines needed by the project stakeholders to deliver the system.

The process begins with developing an over all model –as shown in figure 13- in which a "walkthrough" is carried out by the domain experts to inform the chief architect and the team members of the high-level description of the system. The overall domain is further divided into sub-domains and development team works in each domain to produce an object model, to construct an overall model for the system.

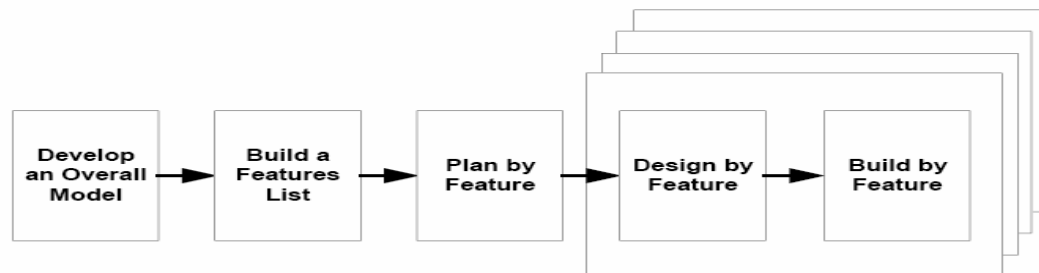


Figure 13: FDD process (Abrahamson, et al. 2002)

In the next phase, a feature list is constructed, in which, the development team presents each of the customer valued function included in the system. The functions are presented for each of the domain areas and for groups called feature list, which in their turn, further divided into feature sets. The customer and the suppliers review the feature list for their validity and completeness.

The tasks and responsibilities are distributed between the project manager, chief architect, development manager, chief programmer, class owner, domain expert, domain manager, release manager, language lawyer, build engineer, toolsmith, system administrator, tester, deployer and technical writer.

2.1.4 *The Rational Unified Process*

Rational unified process (RUP) is a well-known software engineering process that provides a disciplined approach to assigning tasks and responsibilities within a development organization (de Barros Paes, et. al. 2007).

The life span of RUP projects is divided into four phases, each split into iterations -as shown in figure 14- that have the purpose of producing a demonstrable piece of software.

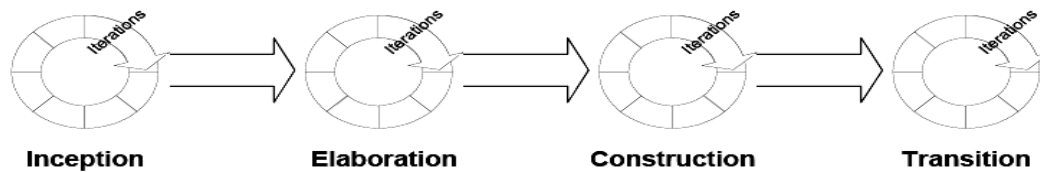


Figure 14: RUP phases (Abrahamson, et al. 2002)

All the needs of every stakeholder is considered in the Inception phase, along side with critical use cases to be used, the candidate architectures of the system, and the schedule and cost of the entire project. (Abrahamson, et al. 2002).

The transition phase is entered when the product is mature enough to be released. Based on the customer response some releases could be made to correct any problem of finishing any postponed feature.

Among various workflows carried by RUP, Business modeling is performed to ensure that the customer's needs are satisfied. By analyzing the customer's organization and business process, a better understanding of the structure and of the business is gained.

2.1.5 *Dynamic System Development Method*

(Norton 2007) presents dynamic system development method (DSDM). DSDM consists of five phases, as shown in figure 15. The feasibility study phase is mainly concerned with the technical ability to build the required software, judging the domain of the project as well as, deciding whether to use DSDM or not.

The business study involves workshops where a sufficient number of customer's experts are gathered to be able to consider all relevant aspects of the system including the requirements and the effected business processes. Another two outputs are the architecture definition and the prototyping plan.

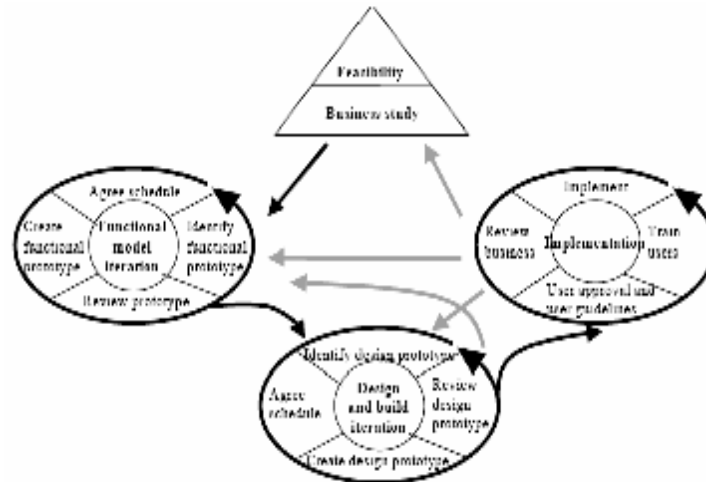


Figure 15: DSDM process (Abrahamson, et al. 2002)

The design and build iteration is where the system is mainly built; the output is tested to make sure that the system satisfies the requirements. The final phase is the implementation where the system is transferred to the customer environment.

Key responsibilities are assigned to the developers, technical coordinator, executive sponsor and a visionary. A visionary is the customer participant who has the most accurate perception of the business objective of the system and the project.

2.1.6 Adaptive Software Development

Adaptive software development (ASD) focuses on the problems in developing complex, large systems. The method aims to provide a framework with enough guidance to prevent projects from falling into disorder (Highsmith, J.A. 2000).

Figure 16 illustrates the adaptive software development life cycle, which contains three main phases. The project initiation phase defines the cornerstones of the project, and is begun by defining the project mission.

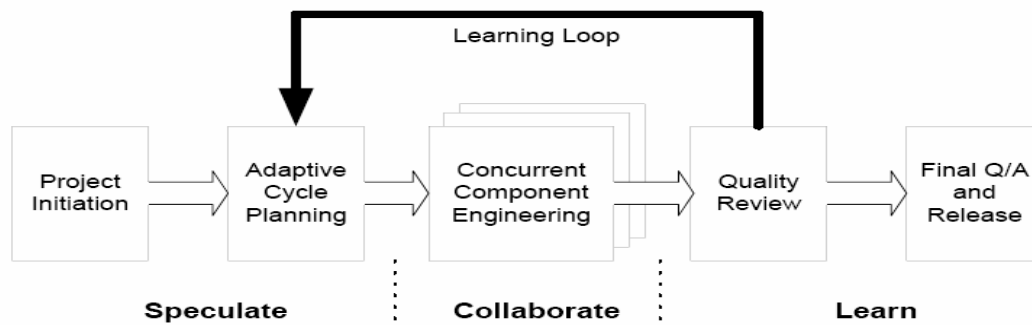


Figure 16: The ASD life cycle (Abrahamson, et al. 2002)

ASD is explicitly component oriented rather than task oriented. It focuses more on results and their quality rather than the tasks and the process used for producing the result. This is achieved in the collaborate phase, where several components may be under concurrent development.

Basis for the learning loop is gained from repeated quality reviews performed by the customer group of experts called the customers focus- group, and is carried out using joint application design sessions (workshops).

2.2 Joint Application Design

Joint application design (JAD) is a methodology that involves the customer or end customer in the design and development of an application, through a succession of collaborative workshops called JAD sessions. The JAD approach, in comparison with the more traditional practice, is thought to lead to faster development times and greater customer satisfaction, because the customer is involved throughout the development process. In comparison, in the traditional approach to systems development, the developer investigates the system requirements and develops an application, with customer input consisting of a series of interviews ([Http://Www.Bitpipe.Com/Tlist/Joint-Application-Development.html](http://www.bitpipe.com/Tlist/Joint-Application-Development.html)).

JAD is used in the Systems Development Life Cycle to collect business requirements while developing new information systems for a company. It consists of a workshop where “knowledge workers and IT specialists meet, sometimes for several days, to define and review the business requirements for the system (Haag, et al. 2006).” This acts as “a management process which allows Corporate Information Services (IS) departments to work more effectively with users in a shorter time frame (Jennerich 1990).

As shown in figure 17, JAD defines a set of steps for a successful requirement collecting process using the workshops method, starting with Identifying project objectives and limitations which is vital to have clear objectives for the workshop and for the project as a whole. Then identify critical success factors for both the development project and the business function being studied. As well as defining the schedule of workshop activities along side with selecting the participants which are the business users, the IS professionals, and the outside experts that will be needed for a successful workshop.

The facilitator is responsible for preparing the workshop material before the workshop; he also organizes workshop activities and exercises, and design workshop exercises and activities to provide interim deliverables that build towards the final output of the workshop.

Key participants in the workshops are the executive sponsor, project manager, facilitator, Documentation Expert: Customers and Observers.

When properly used, JAD can result in a more accurate statement of system requirements, a better understanding of common goals, and a stronger commitment to the success of the new system. A drawback of JAD is that it opens up a lot of scope for inter-personal conflict ([Http://En.Wikipedia.Org/](http://en.wikipedia.org/)).

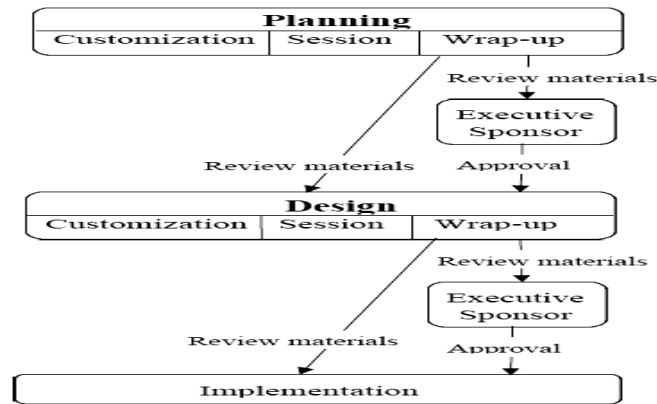


Figure 17: JAD phases (Jennerich 1990)

2.3 The Modular-Model

The Modular-Model proposed In (Maheswar 2002), aims at reducing the uncertainty and complexity of project by dividing the project into modules and each logical module is viewed independently so that parallelism is achieved and thus improves efficiency.

The model clearly defines the task of complete application including the customer and it also incorporates iteration increment, which ensures that the application meets the customer's requirement by proper verification and validation, incorporates the demand for faster delivery there-by focuses on value and return on investment (ROI).

Features like prototyping, modular-division, risk analysis and clearly defined tasks enhance the capability of the model to achieve the planned target within the prearranged limits of time, budget and scope. The Modular-Model for the software development lifecycle basically divides the complete application into various modules based on the customer's requirements and specifications that are known using prototyping phenomenon.

The model typically divides the whole processes into two segments, as shown in figure 18.

- Customer/Programmer
- Developer/Testing.

To make the understanding better, the modular model has been further divided into four quadrants:

- Developer Phase Involving Customer.
- Developer Phase Involving Programmer.
- Testing Phase Involving Programmer.
- Testing Phase Involving Customer.

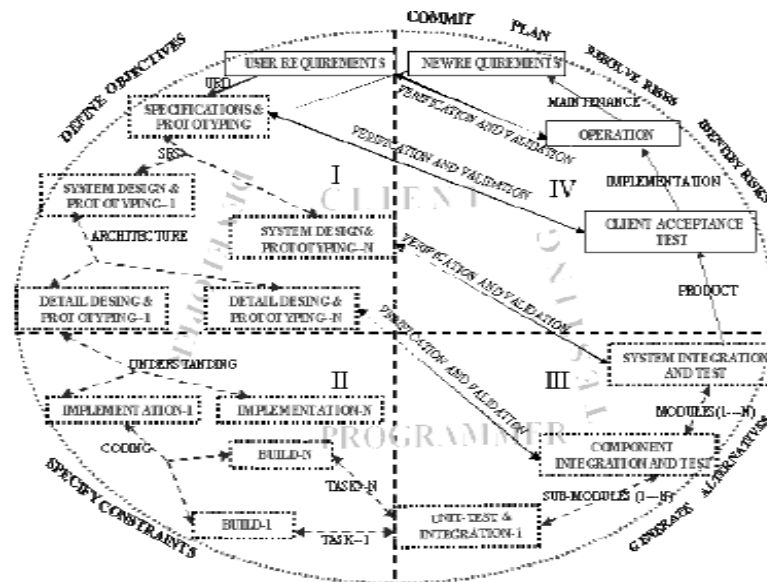


Figure 18: The Modular Model (Maheswar 2002),

2.4 The Behavior Tree

(Grunske et al. 2007) and (Itani and Logrippo 2005) uses an approach to create, formalize and analyze behavior tree models used for checking the consistency, completeness, and soundness of system requirements. (Dromey 2006) claims that using behavior trees to represent software system behavior, can help making significant gains in software development.

Behavior trees are defined by (Zheng and Dromey 2003) as a formal, tree-like graphical form that represents behavior of individual or networks of entities which realize or change states, make decisions, respond-to/cause events, and interact by exchanging information and/or passing control.

The Behavior Tree Notation captures in a simple tree-like form of composed component-states what usually needs to be expressed in a mix of other notations. Behavior is expressed in terms of components realizing states, augmented by the logic and graphic forms to capture behavior expressed in the natural language representation of functional requirements as to provide an abstract graphical representation of behavior expressed in a program. They provide a direct and clearly traceable relationship between what is expressed in the natural language representation and its formal Behavior Tree equivalent. The Behavior Tree method allow the engineer to manage complexity and scale in the construction of a Behavior Tree model which is "built out of" its requirements (Powell 2007). Behavior trees key elements notation is shown in figure 19. More detailed description is provided in Appendix A.

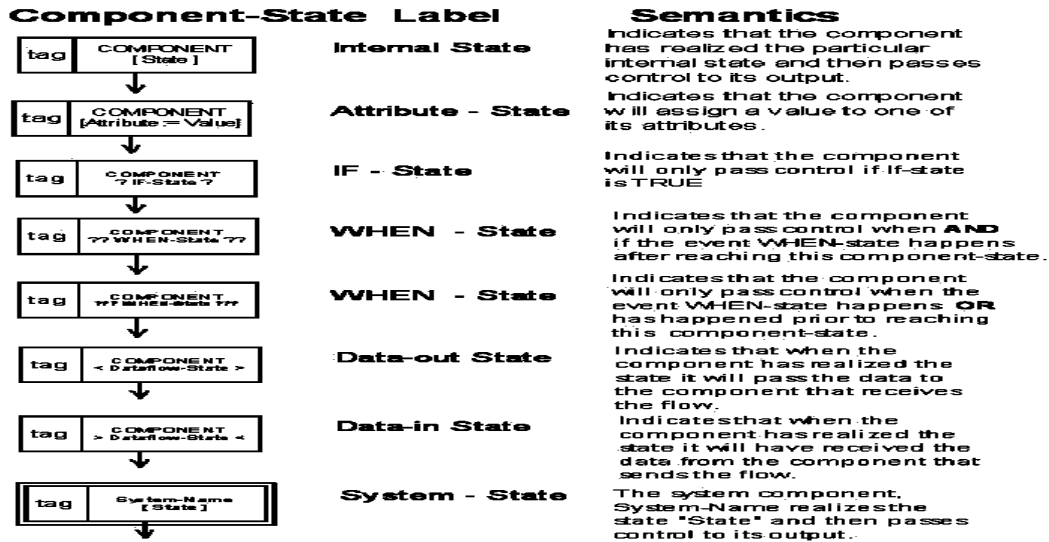


Figure 19: Behavior tree notation key elements (Powell 2007)

Figure 20 shows an example of how natural language is translated into a behavior tree notation.

Behavior

When a car is at the entrance if the gate is open the car may proceed, otherwise if the gate is closed, when and if the driver presses the button the gate will open and then the car may proceed.

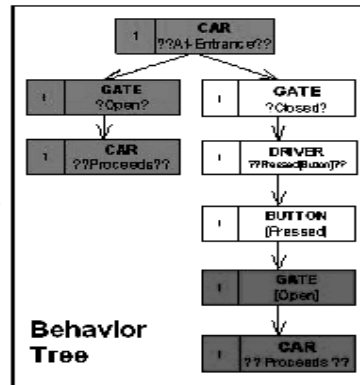


Figure 20: Translation of natural language to a behavior tree (Powell 2007)

(Behavior Tree Group 2007) describes the naming convention of a behavior tree as shown in figure 21.

Notations like sequence diagrams, class and activity diagrams from UML, data-flow diagrams, statecharts and Message Sequence Charts (MSCs), accommodate behavior we find expressed in functional requirements and designs. Individually however, none of these notations provide the level of constructive support and defect visibility we need. behavior trees on the other end, provides a clear, simple, constructive and systematic path for going from a set of functional requirements to a design that will satisfy those requirements. And, in the process, it provides a representation in which defects are much easier to define in precise, concrete and structural terms (Zheng and Dromey 2003).

Label	Name	Description
A	Root Node	The first node in a tree (does not have a parent)
B	Edge	
C	Leaf Node	A node with no children
D	Subtree	A tree contained within the tree rooted at the node of interest
	Branch	A synonym for subtree

1.4 Tree Branch Naming Convention

The following conventions are used to refer to branches of a tree relative to a node of interest.

Label	Name	Description
A	Root Node	The first node in a tree (does not have a parent)
B	Edge	
C	Leaf Node	A node with no children
D	Subtree	A tree contained within the tree rooted at the node of interest
	Branch	A synonym for subtree

Table 1.4: Branches of a Behavior Tree

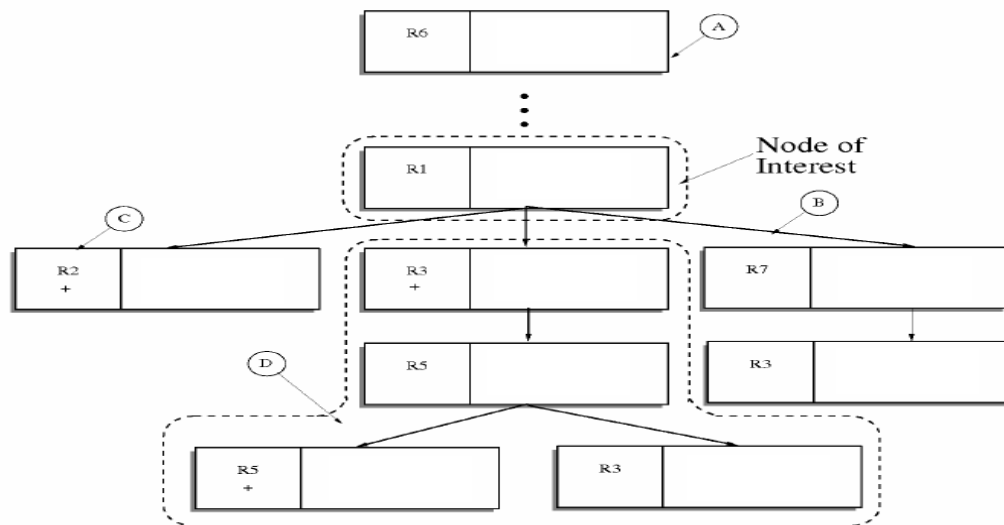


Figure 1.3: Tree Branch Naming Convention

Figure 21: Behavior trees naming convention (Behavior Tree Group 2007),

The complete behavior tree is constructed through two main steps during which, defects in the requirements are captured. It begins with "Requirements Translation", its purpose is to translate each natural language functional requirement, one at a time, into one or more behavior trees called a requirements behavior tree (RBT). Translation identifies the components (including actors and users), the states they realize (including attribute assignments), the events and decisions/constraints that they are associated with, the data components exchange, and the causal, logical and temporal dependencies associated with component interactions.

As the translation is carried out on a requirement-by-requirement basis independent of other requirements, this effort does not tax human short-term memory, regardless of the scale and complexity of a specification.

Translation phase captures the five principal types of defects (Powell 2007):

- Aliases exist where different words are used to describe a particular entity, state, action and event. Hence, it is necessary to maintain a vocabulary of component names and a vocabulary of states associated with each component to maximize the chances of detecting aliases.

- Ambiguities are detected where not enough context has been provided to allow distinguishing among more than one possible interpretation of the behavior described. Ambiguity is often a result of loose language in a requirement.
- Incompleteness can be identified during translation as missing, implied and/or alternate behavior. The behavior tree method does not add any information to a specification unless behavior is missing. These types of incompleteness problems are often found during a walkthrough of the resulting model.
- Inaccuracy is identified as incorrect causal, logical and temporal attribution. Inaccurate atomic statements, usually specified values or ranges, may also be inaccurate.
- Inconsistency is detected during translation if a single requirement statement is inconsistent within itself.

The next step is "Requirements Integration", where individual RBTs are integrated by the precondition and interaction axioms. In practice, it most often involves locating where the root node of one behavior tree occurs in the other tree, and grafting the two trees together at that point. As for translation, integration can be performed without regard for order, again facilitating the engineer's ability to handle scale and complexity by allowing concentration on just the requirements trees being integrated. The outcome of this step is a design behavior tree (DBT) that represents the entire system as defined by the specification being analyzed.

Many defects with requirements can be discovered only by creating an integrated view, because examining requirements individually gives no clue that there is a problem. The defects that are captured during this phase are (Powell 2007):

- Aliases are further detected during integration as requirements are integrated in context. Often when one component is given two different names it becomes apparent during integration.
- Ambiguities are often detected as incompleteness in contextual information during integration. That is ambiguous statements make it difficult to properly model preconditions, which in turn lead to an inability to integrate behavior.
- Incompleteness is usually associated with either incomplete pre and post conditions making integration difficult or impossible, with incomplete sets of events for triggering a behavior, or with incomplete sets of conditions.
- Inaccuracy defect detection is facilitated as behavior is placed in context.
- Inconsistency is detected during integration as a formal integration problem. That is, attempting to integrate two or more inconsistent RBTs into the one DBT would result in contradictory behavior. Inconsistency defects can be model checked.
- Redundancy is also detectable as an integration problem. Redundancy is to be considered a serious defect as it has an impact not only on understanding but on change management.

2.5 Project Management

Project management has emerged as a strong discipline practiced by highly trained, certified professionals as organizations have come to realize that they cannot stay in business if they cannot manage their projects (Wessels 2007).

The first challenge of project management is to make sure that a project is delivered within defined constraints. The second, more ambitious challenge is the optimized allocation and integration of inputs needed to meet pre-defined objectives. A project is a carefully defined set of activities that use resources (money, people, materials, energy, space, provisions, communication, etc.) to meet the pre-defined objectives.

Project management is the province and responsibility of an individual. This individual seldom participates directly in the activities that produce the end result, but rather strives to maintain the progress and productive mutual interaction of various parties in such a way that overall risk of failure is reduced.

A project manager (PM) is often a customer representative –i.e. a supplier employee that explain the customer perspectives- and has to determine and implement the exact needs of the customer, based on knowledge of the firm he is representing. The ability to adapt to the various internal procedures of the contracting party, and to form close links with the nominated representatives, is essential in ensuring that the key issues of cost, time, quality, and above all, customer satisfaction, can be realized.

Projects need to be performed and delivered under certain constraints. Traditionally, these constraints have been listed as scope, time, and cost. These are also referred to as the Project Management Triangle –shown in figure 22, where each side represents a constraint. One side of the triangle cannot be changed without impacting the others. A further refinement of the constraints separates product 'quality' or 'performance' from scope, and turns quality into a fourth constraint. (Jenkins 2006)

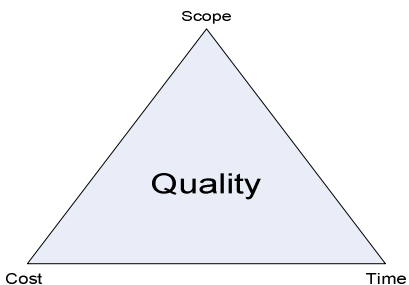


Figure 22: The project management triangle

The time constraint refers to the amount of time available to complete a project. The cost constraint refers to the budgeted amount available for the project. The scope constraint refers to what must be done to produce the project's end result. These three constraints are often competing constraints: increased scope typically means increased time and increased cost, a tight time constraint could mean increased costs and reduced scope, and a tight budget could mean increased time and reduced scope. The discipline of project management is about providing the tools and techniques that enable the project team to organize their work to meet these constraints.

Another approach to project management is to consider the three constraints as finance, time and human resources. If a job need to finish in a shorter time, more people can work on it, which in turn will raise the cost of the project, unless by doing this task quicker we will reduce costs elsewhere in the project by an equal amount (Kerzner 2006).

Customers, on the other hand, can dictate the extent of three variables: time, cost, and scope. The remaining variable "risk" is managed by the project team, ideally based on solid estimation and response planning techniques. Through a negotiation process among project stakeholders, an agreement defines the final objectives, in terms of time, cost, scope, and risk, usually in the form of a project charter.

There are several approaches that can be taken to managing project activities including agile, interactive, incremental, and phased approaches. Regardless of the approach employed, careful consideration needs to be given to clarify surrounding project objectives, goals, and importantly, the roles and responsibilities of all participants and stakeholders (Augustine and Woodcock 2003).

2.6 Projects Documentation

One basic goal of software engineering is to produce the best possible working software along with the best possible supporting documentation. Documentation in this terminology refers to the system documentation generated during software development life cycle -not end user manuals. Empirical data shows that software documentation products and processes are key components of software quality (Cook and Visconti 1994). This section will describe three of the main software documents.

2.6.1 *The Project Charter*

In project management, a project charter or project definition is a statement of the scope, objectives and participants in a project. It provides a preliminary delineation of roles and responsibilities, outlines the project objectives, identifies the main stakeholders, and defines the authority of the project manager. It serves as a reference of authority for the future of the project (<http://en.wikipedia.org/>).

The project charter is usually a short document that refers to more detailed documents such as a new offering request or a request for proposal. The project charter establishes the authority assigned to the project manager. It is considered industry best practice.

The purpose of the project charter is to document:

- Reasons for undertaking the project
- Objectives and constraints of the project
- Directions concerning the solution
- Identities of the main stakeholders

The project charter is a one-time announcement. It clearly establishes the project manager's right to make decisions and lead the project. The intent of a project charter is to give notice of the new project and new project manager and to demonstrate the upper management support for the project and the project manager. It is also used by the supplier to provide a broad direction for the project to the project manager. The charter should precede the other project documents as it establishes the project manager's authority which, in turn, is necessary to get the Stakeholder agreements written. (State of North Dakota 2002).

Project Charter outline should contain the following sections:

1. Project Title.
2. Background to the Project.
3. Aims and Objectives.
4. Criteria of Success.
5. Consequences of Failure.
6. Assumptions.
7. Constraints.
8. Risk Analysis.
9. Contingency plans.
10. Project Documentation.
11. Key Dates in the Project.
12. Project Control.
13. Key Project Personnel.

The charter can make or break a successful project. It can make it by specifying necessary resources and boundaries that will in turn ensure success; it can break it by reducing team focus, effectiveness and motivation.

2.6.2 *The Feasibility Study*

A feasibility study is an analysis of the viability of an idea. The feasibility study focuses on helping answer the essential question of “should the customer proceed with the proposed project idea?” All activities of the study are directed toward helping answer this question.

Feasibility study is undertaken to determine and document a project's viability. The term feasibility study is also used to refer to the resulting document. These results of this study are used to make a decision whether to proceed with the project, or table it. If it leads to a project being approved, it will - before the real work of the proposed project starts - be used to ascertain the likelihood of the project's success (Wickham 2006).

A feasibility study should examine the Technical and organizational requirements of the project, this includes plant and equipment issues, like the type of equipment and technology that the business need to produce the product, the costs involved, the potential suppliers of the equipment, and the time needed to acquire the equipment and begin operations. It also includes managerial and organizational issues, as the right structure for the business, the importance of finding fixed source of supply to the success of the business, the needed qualifications to manage the system operations, and the key staff positions that need to be filled with in the organization to support the product operations.

Within a feasibility study, seven areas must be reviewed, including those of a Needs Analysis, Economics, Technical, Schedule, Organizational, Cultural, and Legal (Thompson 2005).

Conducting a feasibility study is a good business practice; it thoroughly examines all of the issues and assessing the probability of business success (Hofstrand and Holz-Clause 2006).

2.6.3 *Software Requirements Specification*

Software requirements specification (SRS) document is basically an organization's understanding of a customer or potential customer's system requirements and dependencies at a particular point in time usually prior to any actual design or development work. It's a two-way insurance policy that assures that both the customer and the organization understand the other's requirements from that perspective at a given point in time (Jackson 1995).

It's important to note that an SRS contains functional and nonfunctional requirements only; it does not offer design suggestions, possible solutions to technology or business issues, or any other information other than what the development team understands the customer's system requirements to be.

A well-designed, well-written SRS accomplishes four major goals:

- It provides feedback to the customer.
- It decomposes the problem into component parts.
- It serves as an input to the design specification.
- It serves as a product validation check.

The SRS is typically developed during the first stages of "Requirements Development," which is the initial product development phase in which information is gathered about what requirements are needed--and not. This information-gathering stage can include onsite visits, questionnaires, surveys, interviews, and perhaps a ROI analysis or needs analysis of the customer or customer's current business environment. The actual specification, then, is written after the requirements have been gathered and analyzed.

IEEE 830-1998 standard has identified nine topics that must be addressed when designing and writing an SRS:

1. Interfaces
2. Functional Capabilities
3. Performance Levels
4. Data Structures/Elements
5. Safety
6. Reliability
7. Security/Privacy
8. Quality
9. Constraints and Limitations

There's not a "standard specification template" for all projects in all industries because the individual requirements that populate an SRS are unique not only from company to company, but also from project to project within any one company. The key is to select an existing template or specification to begin with, and then adapt it to meet needs.

A "strong" SRS is one in which the requirements are tightly, unambiguously, and precisely defined in such a way that leaves no other interpretation or meaning to any individual requirement. A well-belt SRS should contain the following chapters:

1. Introduction.
2. Overall Description.
3. External Interface Requirements.
4. System Features.
5. Other Nonfunctional Requirements.
6. Other Requirements.

(Kamata, M. I. and Tamai, T. 2007) applied various statistical analysis techniques over the SRS quality data and project outcomes. Some interesting relations between requirements quality and project success or failure were found, including:

1. A relatively small set of SRS items have strong impact on project success or failure.
2. Descriptions of SRS in normal projects tend to be balanced.
3. SRS descriptions, where purpose, overview and general context of SRS are written, are rich in normal projects and poor in overrun projects.
4. When the descriptions of SRS are poor while those of functions and product perspective are rich, the project tends to result in a cost overrun.

Chapter 3

Customer Participation

Today's users of software demand software applications of greater size and complexity than before. Modern software development processes and methodologies focus on fulfilling that demand beside the increased order on software. Our aim is to develop a software development processes with attendant methodologies and technologies that focus on meeting the user requirements as well as improving software quality and productivity.

Improving quality is based on how well that software meets the requirements and the expectations of its users as long as it is kept adequate, reliable, and efficient. Increasing the ratio between the resources required for development and the size and complexity of the developed software is the determinate of improved productivity (Kelley Cyr 2002).

In their survey of IT executives and technical managers, the Standish Group lists ten factors supporting project success; at the top of the list is user involvement (Standish Group 1995).

3.1 Introduction

The main purpose of software development is supporting the business functions of some customer on a certain field, hence, customer trustworthiness is the most important factor in the success of a software project, trustworthiness or dependability "essentially means the degree of user confidence that the system will operate as they expect and the system will not 'fail' in normal use (Somerville 2004).

Software availability, reliability, safety and security are the measures of the software dependability and either one of them have direct relation with customer's service, environment or data. On the other hand, the level of customer confidence in the software is of equal importance and depends on the purpose of the system, the expectations of its users and its current marketing environment.

Obviously if the system fails at any time and does not match its specification, there will be direct impact on the customer especially if they are using the software to serve their own customers –like a banking system or a customer care interface, on the mid and long term the supplier of that software will face bigger loss.

To reduce this risk, suppliers must work closely with customers involving them early and often on the software development lifecycle which will improve project success probability (Laura Rose 2006). Looking from other perspective and as indicated by the Software Engineering Code of Ethics and Professional Practice "Software engineers shall act in a manner that is in the best interests of their customer and employer, consistent with the public interest" (ACM/IEEE-CS 1999), this includes customer satisfaction which can not be achieved if the software does not live up to the customer expectations.

At first glance the stakeholders stated needs could be considered as the highest level reference point for characterizing defects in software production. Unfortunately

stakeholders often get wrong what they require of a system. The highest reference point needs for any system must therefore be the “real needs” of the system subject to domain and quality requirements that meet professional and community standards. This vague reference point is complicated further because needs of a system change over time). The difference between the real and the stakeholders needs is shown in figure 23 (Zheng and Dromey 2001).

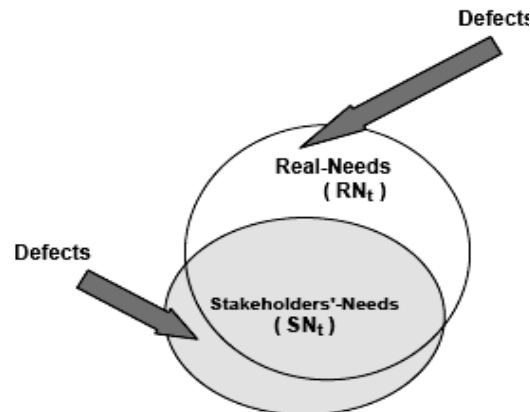


Figure 23: Difference between the real needs and the stakeholders needs

Hence, stakeholders needs are likely to be incomplete, some needs may be wrong and still others may be in conflict, inconsistent, redundant or even unnecessary. It follows that the reference point of “real-needs” only provides a context because it is not likely to be fully known either by the Stakeholders or the software engineers. The aim of this thesis is to develop a software engineering life cycle that enhances customer participation in projects, which will assist in decreasing the gap between the stakeholder needs and the system real needs. .

3.2 Customer Involvement in Software Production Phases

Although software processes are different and variant, they all carry some common activities. As illustrated in figure 24, these activities include Software Specification, Design and Implementation, Validation and Verification (V&V), Operation and Maintenance (O&M), and Evolution.

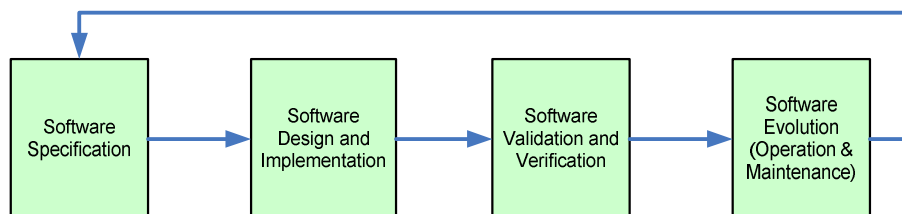


Figure 24: Software processes common activities

3.2.1 Software Specifications (Requirement engineering)

The Requirement engineering process or software specification is a particularly critical stage of the development lifecycle "as errors at this stage inevitably lead to later problems in the system design and implementation" (Somerville 2004).

The most important function of this activity is to specify a system that will meet the needs of the customer, which –in their part- plays a key role in the success of the process as they must be involved in the elicitation and validation of the requirements to ensure that the problem suppliers has built full understanding of the problem, and that the requirements are accurate (Beckworth and Ganer 1994). The main phases of the requirement engineering process are shown in figure 25.

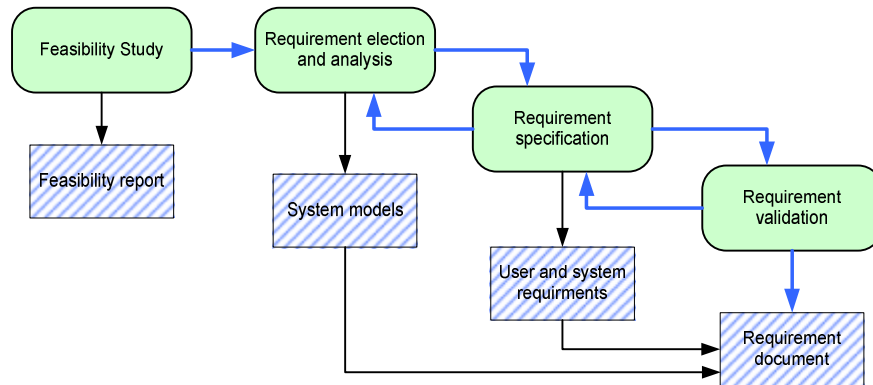


Figure 25: The requirement engineering process (Somerville 2004)

The feasibility study involves customer and supplier interaction to understand and identify the customer request and to make sure if the idea is feasible and can be satisfied using the existing software and hardware technologies.

Requirements elicitation and analysis is among the most communication-rich processes of software development. It engages different stakeholders, from both the customer and the developer sides, who need to intensively communicate and collaborate.

As a key part of the requirements engineering Process, requirements elicitation has a great impact on the later development activities; any omission and incompleteness may lead to important mismatches between customer's needs and released product. Elicitation techniques include questionnaires and surveys, interviews and workshops, documentation analysis and participant observation (Lanubile 1996).

During this phase requirements should be negotiated and analyzed carefully since many software projects have failed because their requirements were poorly negotiated among stakeholders (Boehm 1996). In the other hand, software architecture alternatives cannot be evaluated in a thorough way without consideration of different stakeholders' negotiated requirements (Thomas and Millett 2007).

The SRS is the product of the requirement specification phase; the basic issues that the SRS should tackle are the following:

- **Functionality:** What is the software supposed to do?
- **External interfaces:** How does the software interact with people, the systems hardware, other hardware, and other software?
- **Performance:** What is the speed, availability, response time, recovery time of various software functions, etc.?
- **Attributes:** What is the portability, correctness, maintainability, security, etc. considerations?

- Design constraints imposed on an implementation: Are there any required standards in effect, implementation Language, policies for database integrity, resource limits, operating environment(s) etc.?

Joint Preparation is of high importance for producing a well-written and completely understood SRS, because usually neither the customer nor the supplier is qualified to write a good SRS alone.

Customers usually do not understand the software design and development process well enough to write a usable SRS, but they have big understanding to their demands, and have wider knowledge in the needed external interfaces and the non functional requirements of the system. Suppliers usually do not understand the customer's problem and field of endeavor well enough to specify requirements for a satisfactory system (IEEE Std. 610.12).

The validation phase attempts to increase confidence that a given requirement corresponds to the end-user's desires (Beckworth and Ganer 1994). It is concerned with showing that the requirements define the requested system with no conflicts, contradiction, errors and omissions; checks is carried out on the requirement to guarantee this, these checks include validity checks, consistency checks, completeness checks, realism checks and verifiability checks (Somerville 2004).

The requirement document is the outcome of the requirement engineering process, it include beside the SRS, the product perspective and functions, the user characteristics, assumptions and dependences as well as general constrains; other documents could be produced to target matters pertaining to production of software. These could include items such as (IEEE Std. 610.12):

- Cost
- Delivery schedules
- Reporting procedures
- Software development methods
- Quality assurance
- Validation and verification criteria
- Acceptance procedures

The intended users of these documents are the system customers, managers, engineers, test engineers, and maintenance engineers (Somerville 2004).

3.2.2 *Software Design and Implementation*

The implementation stage of the software development is the process of converting system specifications into an executable product; as shown in figure 26 at this stage supplier's architectures, designers, and developers do most of the work. Customer participation in this activity is tangential unless iterative or agile development schemes are used, in these approaches, adaptive and customer-driven planning is forefront.

Customer-driven development implies that the choice of features for the next iteration or release comes from the customer. The focus is on whatever the customer perceives

as the highest business value. A developer's job is to create quality code, applications, and products that are valuable to the customer.

However, Customer interaction is a never-ending activity and having detailed discussions with the customer about early versions of the product design verifies that developer is on the right track.

Customers also need to see something concrete before they can give useful feedback. By providing quick prototypes, earlier design review, and short iterations to delivery, along side with the normal beta release, the risk of the customer finding critical usability issues at the later stages is reduced.

Still, many development organizations do not consult the customer, but instead isolate the choice of features to their business analyst or product marketing teams. Prioritization of features is done by the product managers via their interpretations of the market trends and competition.

Unclear and vague requirements are passed to the development teams, who typically have even less contact with the customer. They code the features to their perceptions - which are far removed from the customer's viewpoint. Customers often do not see the application until after code freeze or during a formal beta cycle, which is too late.

In contrast, if the supplier carried the right level of commitment, combined with frequent customer interaction and a willingness to remain flexible in implementing the solution, software development teams are much more likely to provide the features that make a software project successful (Laura Rose 2006).

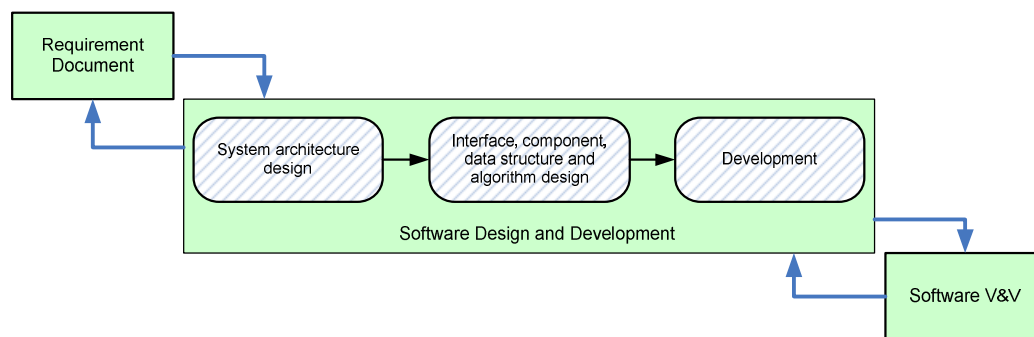


Figure 26: Software design and development stages

3.2.3 *Software Verification and Validation*

Software V&V processes determines whether the development products of a given activity conform to the requirements of that activity, and whether the software satisfies its intended use and user needs.

Verification is defined as the process of evaluating a system or component to determine whether the products of a given development phase satisfy the conditions imposed at the start of that phase. Whereas Validation is the process of evaluating a system or component during or at the end of the development process to determine whether it satisfies specified (IEEE P1012/D12). In other words, Validation is 'end-to-end' verification (Boehm 1996).

Whatever the size of project, software verification and validation greatly affects software quality. People are not infallible, and software that has not been verified has little chance of working. Typically, 20 to 50 errors per 1000 lines of code are found during development, and 1.5 to 4 errors per 1000 lines of code remains even after system testing (Gibson 1992). Hence, every project must verify and validate the software it produces, this is done by (Boehm 1996):

- Checking that each software item meets specified requirements.
- Checking each software item before it is used as an input to another activity.
- Ensuring that checks on each software item are done, as far as possible, by someone other than the author.
- Ensuring that the amount of verification and validation effort is adequate to show each software item is suitable for operational use.

Each project must define its Software Verification and Validation activities in a SVVP. Users, managers and developers all need to be assured that the software does what it is supposed to do.

An important objective of testing is to show that software meets its specification. The 'V diagram' in figure 27, shows that unit tests compare code with its detailed design, integration tests compare major components with the architectural design, system tests compare the software with the software requirements, and acceptance tests compare the software with the user requirements.

All these tests aim to 'verify' the software -i.e. show that it truly conforms to specifications (Boehm 1996). In the other hand, Customer involvement in the testing process is crucial to achieving the required functionality (Arthur and Nance 2000).

One of the most important tests in the V&V cycle is the acceptance testing, which is a Formal testing procedure, conducted to determine whether or not a system satisfies its acceptance criteria and to enable the customer to determine whether or not to accept the system. This test is often carried out by the customer, on his site and using real data.

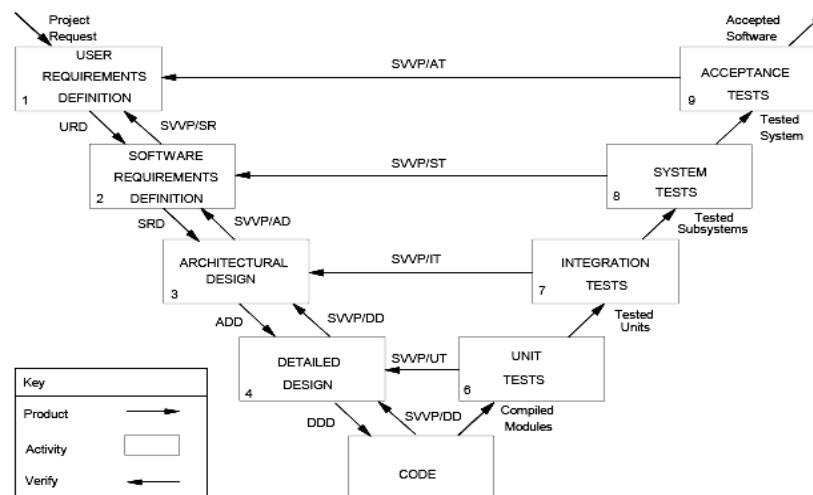


Figure 27: The Lifecycle of verification approach (Boehm 1996)

Testing methods and tools, in themselves, do not guarantee effective testing and ensure high quality of software. The key to improving the effectiveness of testing is to improve the attitude of software developers towards testing and the nature and culture of the organization. Also, testing has to be seen in a broader perspective of maximizing customer satisfaction and providing feedback for process refinement, rather than just detecting and correcting errors in the software (Murugesan 1994). This could be achieved by involving customers in the testing process and getting their online feedback through direct participation.

3.2.4 *Software Evolution and Maintenance*

Software evolution and maintenance is a very broad activity often defined as including all work made on a software system after it becomes operational (Canfora and Cimitile 2000). It starts when the initiator provisionally accepts the software (Boehm 1996). The IEEE defined this phase as “the process of modifying a software system or component after delivery to correct faults, improve performances or other attributes, or adapt to a changed environment” (IEEE P1012/D12).

Maintenance plays an important role in the lifecycle of a software product. It is estimated that there are more than 100 billion lines of code in production in the world. As much as 80% of it is unstructured, patched and not well documented. Maintenance can alleviate these problems (Kegan et al. 2003). According to (Niessink and van Vliet 2000), customers judge the quality of software maintenance differently from how they judge the quality of software development. This implies a need to carry out software maintenance through different processes from those used by the average software development organization.

The software maintenance process is classified into four categories (IEEE P1012/D12):

- Corrective maintenance: reactive modification of a software product performed after delivery to correct discovered faults.
- Adaptive maintenance: modification of a software product performed after delivery to keep a computer program usable in a changed or changing environment.
- Perfective maintenance: modification of a software product performed after delivery to improve performance or maintainability.
- Emergency maintenance: unscheduled corrective maintenance performed to keep a system operational.

There are two major activities related to software operations have:

- User support: it includes 'end user' that utilizes the products or services of a system. And an 'operator' who controls and monitors the hardware and software of a system. A user may be an end user, an operator, or both.
- Problem reporting: Users should document problems in Software Problem Reports (SPR). These should be genuine problems that the user believes lie in the software, not problems arising from unfamiliarity with it.

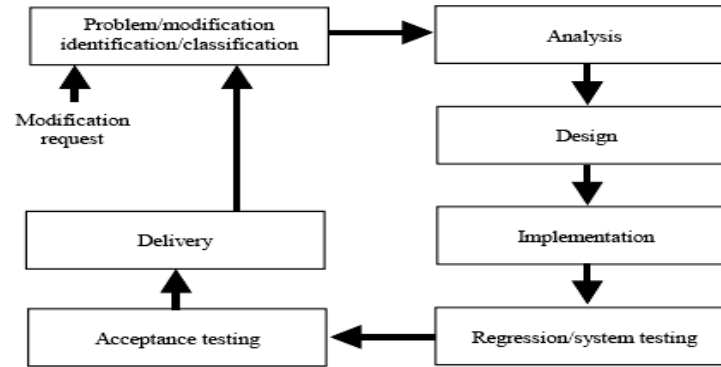


Figure 28: The IEEE maintenance process (Canfora and Cimitile 2000)

3.2.5 Software Project Management

The customer participation in the project management activities is a key factor in project success. On the customer side, Project Management duties should not be considered as a part-time function for an employee or more appropriately a manager that has other full-time responsibilities. Prior project management training or experience is a must for the successful completion of complex projects.

Adaptation of standardized project management methodologies should also be implemented by the Customer. The customer could take part on some or all of the below project management activities (Thomas and Millett 2007):

- Schedule/Time management
- Cost Management
- Quality Management
- Human Resource Management
- Contract/Procurement Management
- Communications Management
- Scope Management
- Risk Management
- Project Integration Management

Chapter 4

A Customer Oriented Software Development Life Cycle

Software development life cycle (SDLC) is defined as a concept of providing a complete support to a software product throughout all stages of its evolution. Where as, Software life cycle is a Period of software product life from its conception, development and roll-out until end of use and removal from market. Hence, the software development life cycle provides control over the software lifecycle.

In this chapter we propose a new SDLC that focuses on enhancing the customer participation in the software production processes, relying on his knowledge in the environment, practices, communication methodology, and structure of the hosting environment.

4.1 The Model Main Phases

Our proposed model contains five phases (as shown in figure 29):

- Customer Preparation Phase.
- Requirement Engineering Phase.
- Design and Development Phase.
- Testing Phase.
- Closure Phase.

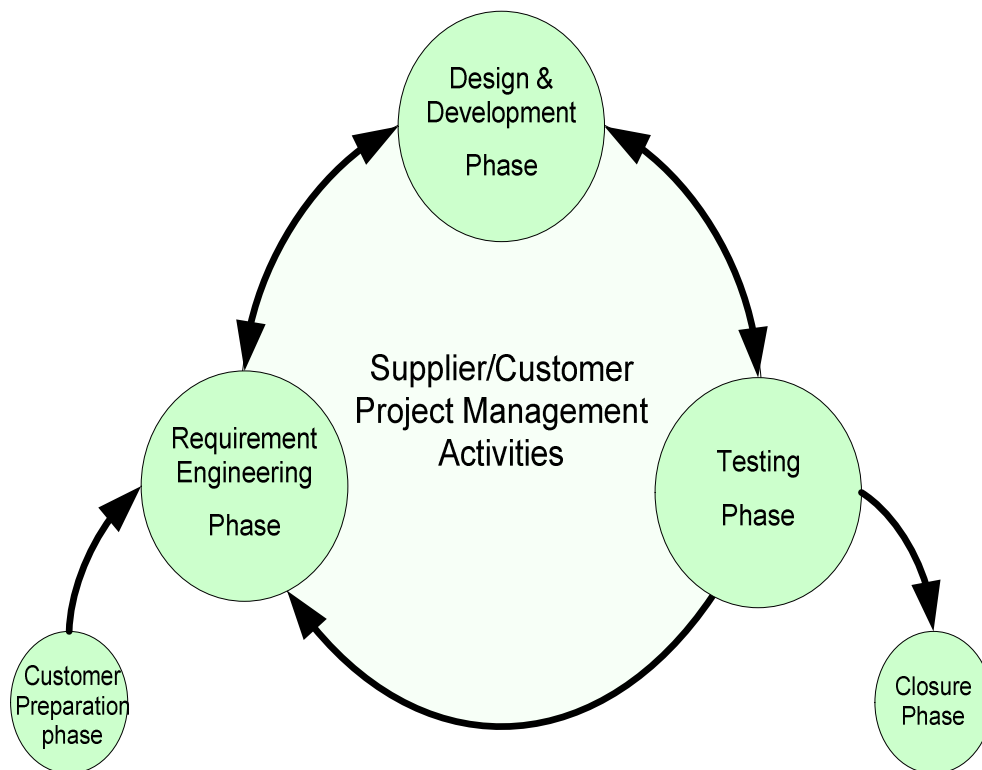


Figure 29: The proposed software development life cycle

4.1.1 Customer Preparation Phase

The customer preparation stage determines the nature and the scope of the development, as well as provide basic understanding to the customer team about the nature of software projects and software development. If this stage is not performed well, it is unlikely that the project will be successful in meeting its needs. The key project controls needed here are an understanding of the business environment and making sure that all necessary controls are incorporated into the project.

This stage -shown in figure 30- should include a series of activities that covers the following areas:

1. Feasibility study of the project.
2. Defining stakeholders, project managers, and project team
3. Customer team training.
4. Defining the project business goals and objectives.
5. Producing and signing of the project charter.

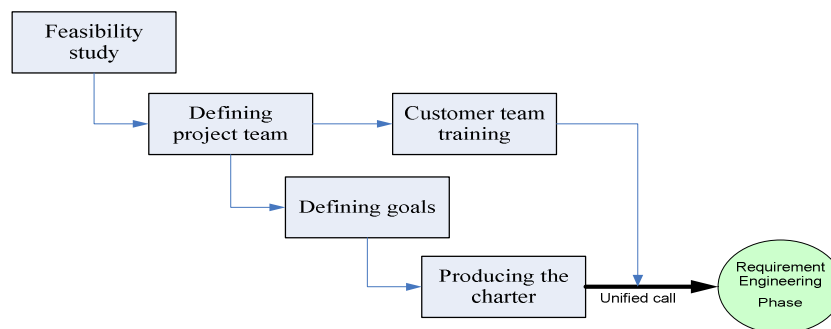


Figure 30: The customer preparation phase

The first step is conducting a rough feasibility study; that is used to measure and assess the technical viability of the projected outcome. This rough feasibility is carried by the customer and the supplier together, and focuses on answering questions about whether the technology needed for the system exists? How difficult it will be to build? And whether the firm of the supplier has enough experience using that technology?

On the other hand, this study only evaluates the ability of the supplier to provide the system in general terms; A more detailed and thorough feasibility will be conducted through the requirement engineering phase that measures the system based on an outline design of system requirements in terms of input, output, fields, programs, and procedures.

A conclusion will be reached in the end of this activity, to indicate whether to proceed with the proposed project or not; if the results of the feasibility study are positive, the project can proceed to next steps.

The key participants are then identified, and their contact information is shared between the customer and the supplier, those are:

- Customer and Supplier stakeholders.
- Supplier project manager.
- Customer project manager.

- Technical writer.
- Software manager.
- Process controller.
- Tree builder.
- Consultant group.

Upon the completion of this activity, the customer will select their representative team in the project, and send them to the supplier firm in order to learn more about the supplier firm, get more experience in the communication methodology, and obtain basic understanding of the supplier environment.

On the same time, the supplier and the customer will identify the goals of the project, this will help focus design decisions and prevent the project from going off course. The goals of the project should be SMART:

- Specific
 - Well defined
 - Clear to anyone that has a basic knowledge of the project
- Measurable
 - The goal should be obtainable.
 - Know when it will be achieved
- Agreed Upon
 - The goals should be agreed between all the customer stakeholders.
- Realistic
 - The goals can be satisfied.
- Time Based
 - Enough time is available to achieve the goal.

Having measurable goals helps in having statistical analysis for the project, examples of these goals are:

- Time
- Accuracy
- Overall success
- Satisfaction

In the last step of this phase, the supplier and customer project managers will develop a project charter that will be signed by project stakeholders from both sides. By the end of the phase the customer and the supplier will agree to take the project to the next step –i.e. requirement engineering phase.

4.1.2 *Requirement Engineering Phase*

This phase includes requirements elicitation, analysis, definition, and specification. The major outcomes of this phase are a behavior tree along side with the system requirement specification document.

The major role in this phase is for the customer, where they make the election of the requirements, the analysis and validation of these requirements will be done by the supplier, and the definition will be done by both parties as well as the specifications.

For the requirement engineering phase in particular, the model will use a RUP-like process –i.e. a process that follows the RUP flow, shown in figure 31, to build the requirement of the system.

Requirements Elicitation focuses on obtaining overall requirements of product from customer including information and control needs, product function and behavior, overall product performance, design and interfacing constraints and other special needs.

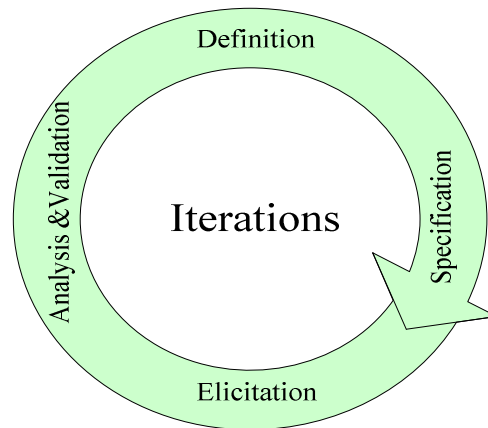


Figure 31: Requirement engineering process

The elicitation will be made through a series of JAD meetings, which will be organized by the customer side project manager. These meetings will be attended by all the project main personals and any needed expert from outside the project personals group, experts could be from the customer end, the supplier end, or a third party.

Customers will be responsible in electing the requirements and explaining them to the supplier, as well as, making sure that the requirements are complete and fully understood by the supplier team.

The supplier specialist staff will perform requirements analysis and validations, which include those tasks that go into determining the needs or conditions to meet customer requirements, taking account of the possibly conflicting requirements, and the feasibility of these requirements. Requirements analysis is critical to the success of a development project.

Customers typically know what they want, but not what software should do, hence, incomplete, ambiguous or contradictory requirements are recognized by skilled and experienced software engineers. The validation process tries to answer the following:

- Validity: Does the system provide the functions which best support the customer's needs?
- Consistency: Are there any requirements conflicts?
- Completeness: Are all functions required by the customer included?
- Realism: Can the requirements be implemented given available budget and technology
- Verifiability: Can the requirements be checked?

Once the general requirements are collected from the customer, an analysis of the domain of the software as well as the scope of the development should be determined and clearly stated. Certain functionality may be out of scope of the development project as function of cost, others as a result of unclear requirements at the time the development has begun.

After that, these process requirements will be classified into four types:

- Accepted
- Conflicting
- Require more clarification.
- Rejected (could be not feasible or out of scope).

Accepted requirements are then defined, categorized, prioritized and distributed over a behavior tree, where they are given a unique identity number for every single requirement.

Specification is the task of precisely describing the software to be written, possibly in an exact way. Specifications are most important for external interfaces that must remain stable. The detailed specification of each requirement is then studied between the customer and the supplier and written on the SRS.

Non-Accepted requirements will repeat the whole process again until a complete and accepted behavior tree and SRS are created. The identity number of each requirement should be the same on both the behavior tree and the SRS.

During this phase, the customer team that was selected on the preparation phase will continue training in the supplier firm.

4.1.3 *Design and Development Phase*

In this phase the designers of the supplier will design the software architecture based on the developed behavior tree and the SRS. The developers will program the requested software. This phase can be based on evolution where an upgrade is done on existing software, which is owned by the supplier or customer.

The main part in this phase is for the supplier, but the customer will participate by creating a virtual tunnel between the development team and product stakeholders from the customer end. So, the customer team that was trained in previous phases will start working at this phase.

This phase includes six processes:

1. Behavior tree analysis:
The development team will start the development process by analyzing the BT that was produced from the requirement phase. This will include further improvement to the resulted BT, by making more grouping for the similar subtrees and redesign the BT to produce an optimized system BT.
2. Software architecture:
The architecture of a software system refers to an abstract representation of that system. Architecture is concerned with making sure the software system will meet the requirements of the product, as well as ensuring that future

requirements can be addressed. The architecture step also addresses interfaces between the software system and other software products, as well as the underlying hardware or the host operating system.

The system architecture will be based on the system BT, considering various subtrees and groups off the tree, which will make the architect work easier.

3. Integration protocol definition:

During this activity, a protocol will be created to identify the coding scheme and the protocol that should be followed by the developers for subtrees integration.

4. Teams distribution:

At the beginning of this activity the project managers will introduce the system for the customer and vendor teams that was not participating on the process from the beginning which will help in building an overall understanding of the required job.

Depending on the architecture results, the development team will be divided into several -but smaller- teams. Based on the size of work, the software manager will distribute the subtrees over these teams, where related subtrees will be assigned to a single team.

The integration protocol defined in the previous step will be explained to all the teams in order to be considered through coding.

5. Subtree coding:

Reducing a design to code may be the most obvious part of the software engineering job, but it is not necessarily the largest portion. During this activity the developers will start coding the subtrees where successfully coded requirements will be moved to the testing phase.

The customer team, on the other hand, will create a communication channel between various project members and teams trying to explain the requirements in a real time manner to the developers to complete their task with the minimal time.

6. Subtree Integration coding:

If a subtree is tested successfully, a development team specialized in integration will glue the subtree code to the other parts of the system that are already developed.

Upon completion of all the subtrees integration, the system will be sent for acceptance testing.

The customer team will work closely with all teams trying to explain the requirement to developers, if any ambiguous requirements appeared during this phase the customer team will work with the supplier team to send this requirement back to the requirements engineering phase, accordingly, an update will be done on the BT, to indicate the status of this requirement.

The same applies for requirements moved to the testing phase, where the state of these requirements should also be reflected on the BT. Figure 32 illustrates the various activities on this phase and their relation with other phases.

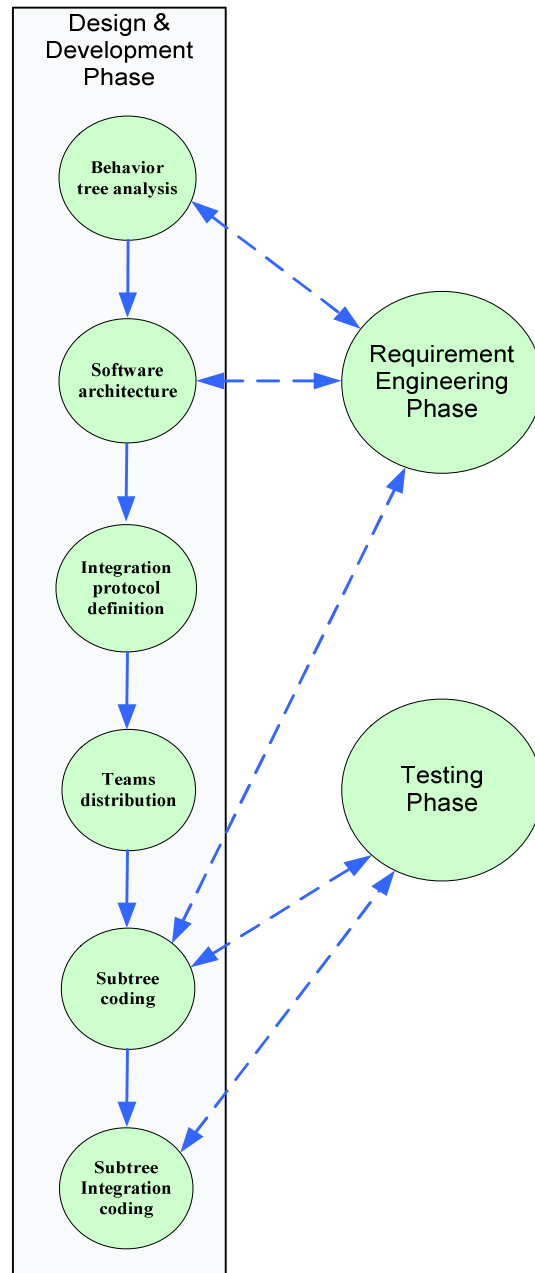


Figure 32: Design and development phase activities

4.1.4 *Testing Phase*

The testing phase –shown in figure 33- is a separate phase which is performed by a different team after the implementation is completed, Because it is hard to see one's own mistakes, and a fresh eye can discover obvious errors much faster than the person who has read the material many times.

There are four types of tests in this phase:

- Internal testing
- Subtree testing
- Application testing
- Acceptance testing

Internal testing deals with low-level implementation, where each function or component is tested. This testing is accomplished by the implementation teams. This test is conducted by a developer on a single node -on the behavior tree- or a group of connected nodes.

Subtree testing deals with testing a single sub tree. This could test the interaction of many functions but impound the test within one subtree. The goal of Subtree testing is to isolate each part of the program and show that the individual parts are correct.

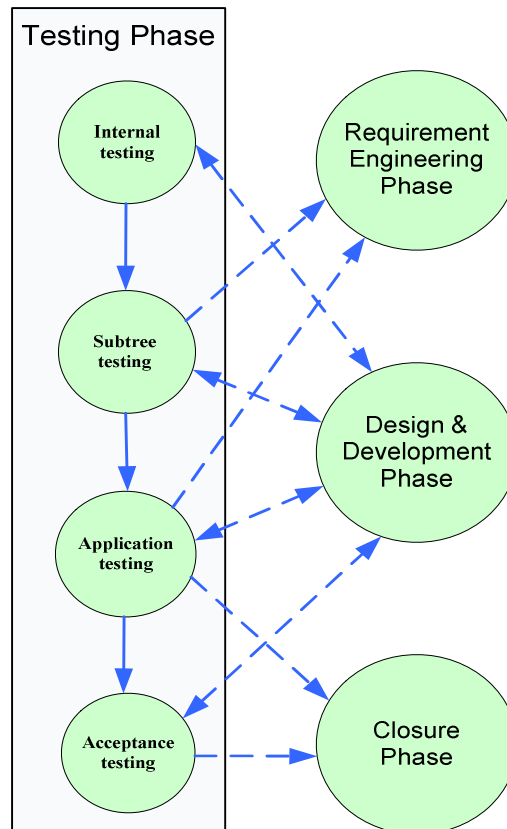


Figure 33: Testing phase and its interaction with other phases

Application testing deals with tests for the entire application. This is driven by the scenarios collected from the SRS and based on the behavior tree. This test includes testing of application limits and features. The application must successfully execute all scenarios before it is ready for general customer availability

Acceptance Testing is performed on the customer environment and using actual inputs. It allows customers to ensure that the system meets their business requirements. This additional test will probably be required for the customer to give final approval for the system.

The customer specifies scenarios to test based on the built behavior tree. Acceptance tests are black box system tests. Each acceptance test represents some expected result from the system. Customers are responsible for verifying the correctness of the acceptance tests and reviewing failed and successful tests. A node is not considered complete until it has passed its acceptance tests.

The acceptance test is the last opportunity customers have to make sure that the system is what they asked for. When this final test is complete, the team expects that the customer will formally approve the system or point out any problems that still need to be resolved. Therefore, unlike all the other tests performed so far, acceptance testing is the customers' responsibility.

Any node or subtree that fails any of these tests will be sent back to the design and development phase. If the whole system does not satisfy the customer requirements, the customer and supplier could agree on repeating the entire requirement engineering phase all over again. Otherwise, the system will be considered ready for delivery and the system will move to the closure phase.

4.1.5 *Closure Phase*

Closure phase is the final phase of the proposed model, it contains the following activities:

- Delivering the final product to the customer.
- Delivering the project documentation to the customer.
- Performing training for the customer operational team.
- Project Evaluation.

The final product will be installed on the customer environment, and a formal acceptance of the system will be signed by the project stakeholders, to indicate that the project had completed.

Most successful projects have one thing that is very evident - they were adequately documented, with clear objectives and deliverables. These documents are a mechanism to align sponsors, customers, and project team's expectations. These documents will be updated in each phase of the project and then delivered to the customer on project closure.

A trainer from the supplier end will then perform the system training for a selected customer team, which will use and operate the system.

The over all project will be evaluated through a questionnaire distributed to the project members and stakeholders. Another evaluation will be separately made, in a form of project assessment document, by the customer and the supplier, through which, each part identify the following:

- Over all assessment of the project based on initial goals.
- Reasons of success, partial success or failure.
- Assessment of project members.
- A track of changes in requirement and in delivery.

4.2 **The Dataflow Model (The Model Main Tool)**

The data flow of the proposed model is based on the behavior tree,; the behavior tree contains three parts (as shown in figure 34):

1. root node: in the below figure, R0 represents a root node for the whole tree, R1, R2 and R3 represents root nodes for its derived subtrees. Also R2.1 could be considered as root node for the nodes R2.1.1 and R2.1.2.

2. Subtree: any part of the tree that can be categorized as related requirements, the sub tree will be named by the root node of it, in the below example, the selected sub trees are named as ST1, ST2 and ST3 respectively. If R2.1 is considered as a root node, its sub tree will be ST2.1.
3. Node: a single requirement that is represented on the behavior tree in one block only. Each node has a unique identifier assigned to it in the following scheme:
 - The main root node of the tree will have the number R0.
 - Nodes of the same parent will be numbered from left to right starting from 1 to n.
 - Each node number will be preceded by its parent number.
 - Nodes with more than one parent will derive its number from the most left parent.

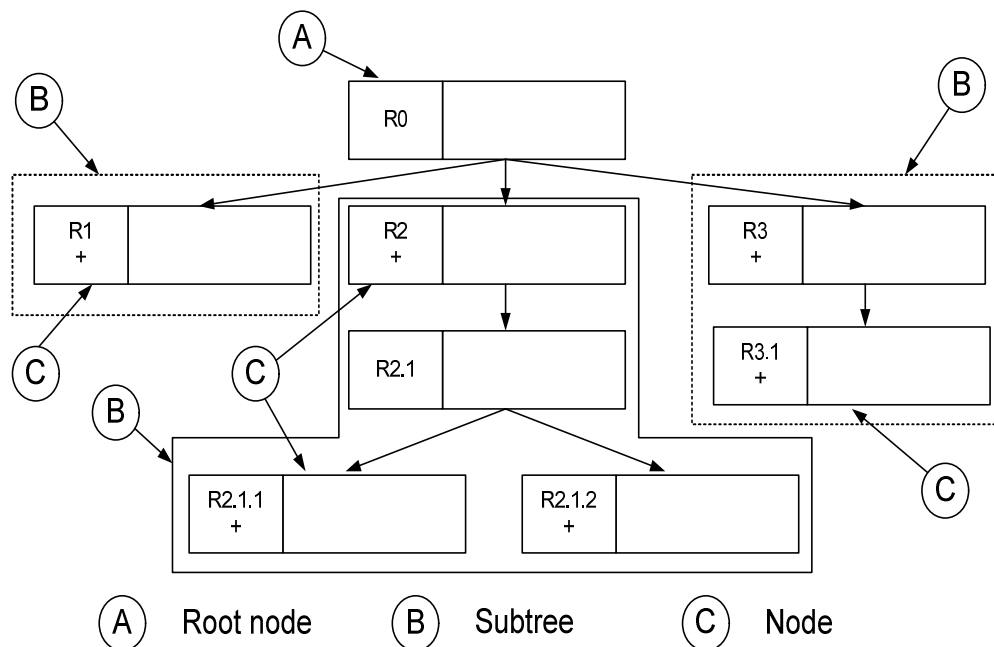


Figure 34: The model behavior tree parts

The behavior tree will represent only functional requirements that are well defined on the SRS, the numbering plan of requirements on the SRS should follow the requirements numbering on the behavior tree, not the other way around.

A coloring scheme should be adopted by the project stakeholders to indicate the status of a certain node, four colors should be defined, each will represent one of following:

1. Node is in requirement engineering phase.
2. Node is in the development phase.
3. Node is in the testing phase.
4. Node is ready.

Figure 35 shows the life cycle of a single node. A node could go back and forth between the project phases, once it's ready, it can not move any more. A subtree passes through the same process as a single node.

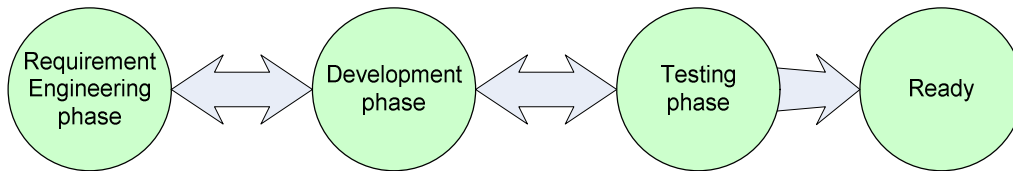


Figure 35: Data flow of a sub tree and a single node

4.3 The Workflow Model (Transaction Between Phases)

Based on behavior tree nodes status, the phases of the proposed model are interleaved; for example, while a certain subtree are in development phase, another subtree in the same project could be in the testing phase and another one may be in the requirement phase.

The transaction between phases –moving from one phase to another or moving the constructed behavior tree or part of it from one phase to another- is based on the calling system –shown on figure 36; the calling system defines the authority of a decision on a certain phase, as well as, govern the project transaction between phases.

The proposed model suggests three main types of calls:

- Customer call: A customer initiated request to move a single node, a subtree, or the whole behavior tree from a phase to another.
- Supplier call: A supplier initiated request to move a single node, a subtree, or the whole behavior tree from a phase to another.
- Unified call: A customer/supplier initiated request to move a subtree or the whole behavior tree from a phase to another.

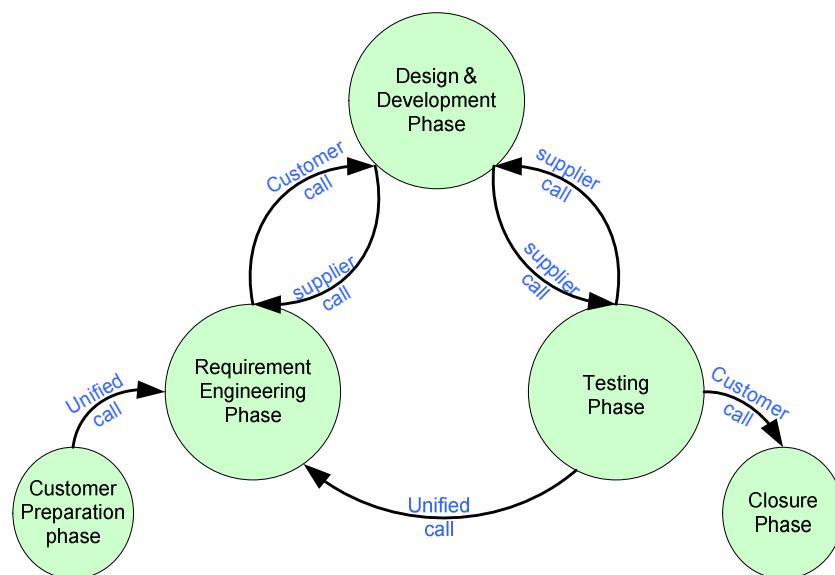


Figure 36: The calling system

Calls can be major or minor. In major calls the whole behavior tree is moved to another phase, while in minor calls, a subtree or a single node is moved.

Every phase on a project has certain types of calls, based on the authority in this phase. In the customer preparation phase a major unified call will be performed if

both parties -the customer and supplier- reached an agreement to proceed with the project. This call will officially indicate the beginning of the requirement engineering phase.

In the requirement engineering phase, the following calls could be issued:

- Customer major call: Upon completion of this phase for the first time, the instance tree will complete and all nodes have the same status -requirement phase status, hence, the customer -who is responsible on verifying the requirements- will issue a major call to move the whole instance tree to the design and development phase. Accordingly, all the nodes on the behavior tree will change status to development phase status.
- Customer minor call: A single node or a subtree could be sent back to this phase from the design and development phase, or the testing phase, after performing the requirement engineering process to these requirements, they will be sent to design and development phase again, and their corresponding status will be changed accordingly.

The authority in the design and development phase is in the hands of the supplier, who may issue the following calls:

- Supplier minor call: if any defects are found with a single requirement (node) or with a subtree, the supplier makes a call to send the defected node(s) back to the requirement engineering phase. The statuses of these nodes are changed to requirement phase status. If a node or a subtree is developed the supplier can also issue minor calls to require testing of these nodes, this includes internal and subtree testing; the status of the corresponding nodes will be changed to testing phase status.
- Supplier major call: if the whole application is finished -every node in the behavior tree is satisfied- the supplier will issue a major call to declare the end of the design and development phase and to require application and acceptance testing, the whole instance tree status will change accordingly.

In the testing phase the following calls can be initiated:

- Supplier minor call: the supplier will be responsible on internal and subtree testing, if a node or a subtree fails to pass on of these tests, it will be sent back to design and development phase with development status. If the test was successful, it will be sent back to development phase, but this time with a ready status, in order to be integrated with other nodes; no development will be carried out on ready nodes.
- Unified minor call: if a defect is found in the supplier understanding of subtree, the customer can agree to repeat the requirement engineering process for this part again, and the status of the contained nodes will be changed as well.
- Unified major call: if the system does not satisfy one of the basic principles of software development, like dependability or security, during the application testing process, the whole instance tree could be sent back to the requirement phase, repeating the to return the whole project starting from the Requirement phase, all the nodes statuses will be initialized to the requirement phase status.
- Customer major call: upon successfully completing the application testing with the supplier, the customer will mark the whole instance tree as ready, and the acceptance test will take place, if successfully completed the customer will issue a major call to declare acceptance of the new project.

No calls are conducted through the closure phase since the development process will be finished by then.

4.4 The Role Model (Major Roles and Responsibilities)

There are different roles in the proposed model for different tasks and purposes during the process and its practices. These roles are described as follow:

- Customer and Supplier stakeholders:
The managers from the customer and supplier end, that are responsible on making a final decision upon disagreement between the customer and the supplier representative teams. The stakeholders will monitor the project through weekly progress reports initiated from the supplier and the customer project managers. The stakeholders are responsible of declaring the preparation and the closure of the project.
- Supplier project manager:
The administrative and financial leader of the project, working as a representative for the customer from the supplier perspective, and the main representative for the supplier from the customer perspective; the model do not define any customized tasks for the Supplier project manager, other than controlling the customer project manager role and involving him in some of the project management activities, and initiating the supplier calls and unified calls from supplier end.
- Customer project manager:
Works along side with the supplier project manager and under his leadership; main tasks include:
 - Creating the project charter with the supplier project manager.
 - Initiating the customer calls and unified calls from customer end.
 - Holding and managing the JAD meeting during the requirement phase.
 - Inviting the customer and the third party consultants during the requirement phase.
 - Explaining the customer requirements to the design and development team of the supplier in the design and development phase.
 - Communicating with the customer to handle any urgent requests from the supplier.
 - Selecting the customer testing team for the acceptance testing, and performing unit, subtree and application testing with the supplier testing team.

The customer project manager will assess the supplier project manager in performing the following tasks:

1. Analysis and design of objectives and events.
2. Planning the work according to the objectives
3. Risk Management
4. Organizing the work
5. Directing activities
6. Controlling project execution
7. Tracking and reporting progress
8. Analyzing the results based on the facts achieved
9. Defining the products of the project
10. Issues management

11. Defect prevention
12. Communicating to stakeholders

- Technical writer:

Responsible of documenting the JAD meetings, writing the requested documents from the project manager, and building the SRS. Having technical writers involved throughout the entire SRS development process can offer several benefits:

- Technical writers are skilled information gatherers, ideal for eliciting and articulating customer requirements. The presence of a technical writer on the requirements-gathering team helps balance the type and amount of information extracted from customers, which can help improve the SRS.
- Technical writers can better assess and plan documentation projects and better meet customer document needs. Working on SRS provides technical writers with an opportunity for learning about customer needs early in the product development process.
- Technical writers know how to determine the questions that are of concern to the user or customer regarding ease of use and usability. Technical writers can then take that knowledge and apply it not only to the specification and documentation development, but also to user interface development, to help ensure the User Interface models the customer requirements.
- Technical writers involved early and often in the process, can become an information resource throughout the process, rather than an information gatherer at the end of the process.

- Software manager:

An experienced developer, who participates in the requirement analysis and the design of the project. The Software manager is responsible in leading small teams in the analysis, design and development phases of the required software. The development manager is responsible of selecting and organizes teams for each subtree or group of subtrees. He is also responsible on the integration plan between each group of requirements, by building a basic structure for the integration that all the teams should follow to guarantee smooth integration of subtrees.

- Process controller:

A supplier repetitive that is responsible of monitoring that the model process is followed, he is responsible of controlling and guiding the project members to follow the model process.

- Tree builder:

Responsible in building the behavior tree and tracking the status of each node as well as defining the nodes that moved at each calls; tree builder responsibility includes mentoring the calls and reflecting its effect on the behavior tree.

- Testing manager:

Responsible of performing and managing all the required tests, and providing feedback to the customer and supplier project managers.

- Consultant group:

This group contains two types of personals that participate in the project in the requirement phase:

- Customer consultants who are experts in a certain products that the needed software need to integrate with it, those could help on defining the interface requirements.
- Third party consultants who are owners of a product that the requested software need to integrate with, those will assist in defining the requirements of the integration interface.

Any contact between the customer and the supplier representatives will be done through the project manager. No communication between the two parties is allowed unless it was controlled by the project managers. The role and communication model between the customer and the supplier is shown in figure 37.

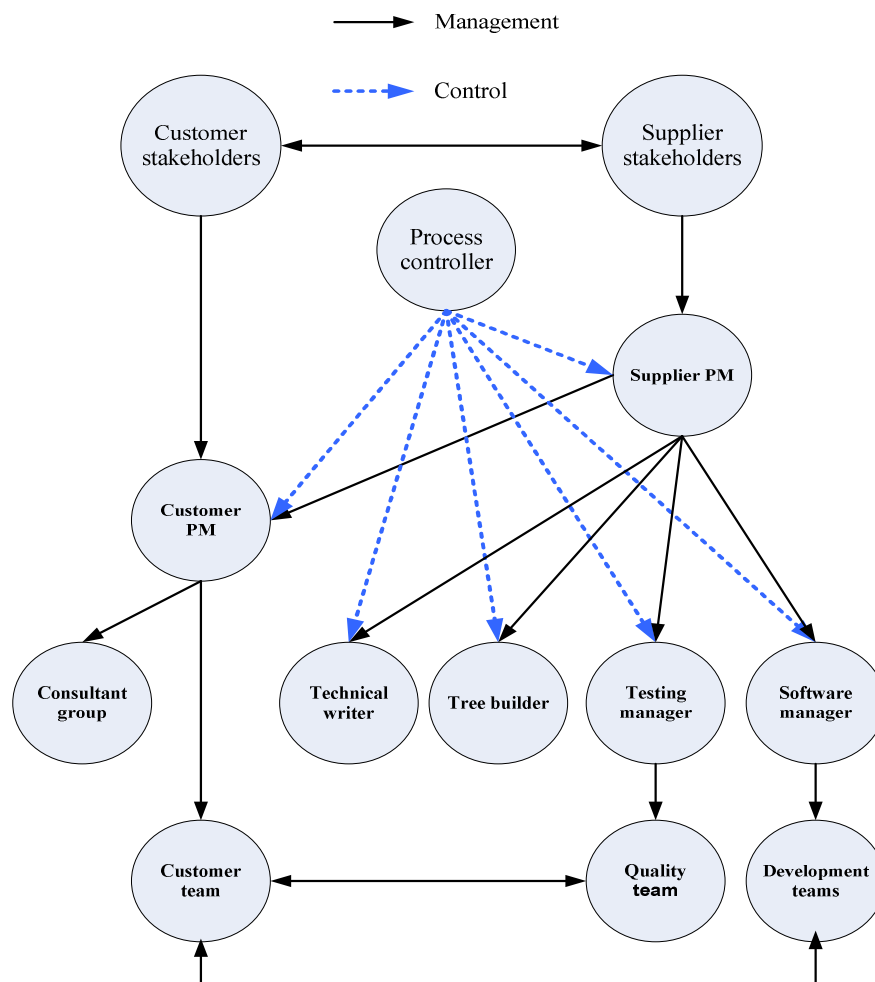


Figure 37: The role and communication model

4.5 The Model Practices

The model focuses on some effective practices taken from other development methods as follow:

From extreme programming the model recommends the existence of an on-site customer individual. This role is done by the customer project manager, who will create a virtual communication tunnel between the customer and the supplier.

Also as XP, the model focuses on human communication, where it is important for software developers, to be able to keep each other informed, resolve issues as they arise, interact productively with customers and generally communicate effectively.

Feature driven development defines the role of the chief manager where the proposed model gives the same responsibility to the software manager. Also as FSS, the model suggests using business modeling to ensure that the customer's needs are satisfied. By analyzing the customer's organization and business process, a better understanding of the structure and of the business is gained.

Joint application Design meetings is held on the proposed model as defined in the process practices, to develop the requirements of the customer.

Based on dynamic system development method, the model recommends developing a feasibility study that is mainly concerned with the technical ability to build the required software and judging the domain of the project.

DSDM also uses a business study that involves workshops where a sufficient number of customer's experts are gathered to be able to consider all relevant aspects of the system including the requirements and the effected business processes. Another two outputs are the architecture definition and the prototyping plan.

4.6 The Model Documents

Careful documentation can save an organization time and money. Unless you are able to produce a document that makes the user comfortable and agreeable, no matter how superior your product might be, people will refuse to accept it

Most successful projects have one thing that is very evident - they were adequately documented, with clear objectives and deliverables. These documents are a mechanism to align suppliers, customers, and project teams expectations, the model suggest establishing the following documents:

1. Project Charter
2. Feasibility Study
3. Scope Statement
4. Project management plan
5. Work Breakdown Structure
6. Change Control Plan
7. Risk Management Plan
8. Risk Breakdown Structure
9. Communications Plan
10. Governance Model
11. Risk Register
12. Issue Log
13. Action Item List
14. Resource Management Plan
15. Project Schedule
16. Status and weekly Reports
17. Responsibility assignment matrix
18. Database of lessons learned
19. Stakeholder Analysis
20. Project assessment.

These documents are normally hosted on a shared resource and are available for review by the project's stakeholders -except for the Stakeholder Analysis, since this document comprises personal information regarding certain stakeholders. Only the supplier project manager has access to this analysis. Changes or updates to these documents are explicitly outlined in the project's configuration management.

Over the course of any project, the work scope changes. Change is a normal and expected part of the development process. Beyond executing the change in the field, the change normally needs to be documented to show what was actually developed.

Chapter 5

Discussion

It is difficult to compare and contrast models of software development because their proponents often use different terminology, and the models often have little in common except their beginnings (marked by a recognition that a problem exists) and ends (marked by the existence of a software solution) (Davis, et al. 1988). This chapter aims to evaluate the proposed SDLC by comparing it with some of the existing software engineering life cycles, showing the strengths and weaknesses of the model.

5.1 Introduction

The task of objectively comparing just about any methodology with another is difficult and the result is often based upon the subjective experiences of the practitioner and the intuitions of the authors (Song and Osterweil 1991). Two alternative approaches exist: informal and quasiformal comparison (Song and Osterweil 1992). Quasiformal comparison attempts to overcome the subjective limitations of an informal comparison technique. According to (Sol H.G. 1983) quasiformal comparisons can be approached in five different ways :

1. Describe an idealized method and evaluate other methods against it.
2. Distill a set of important features inductively from several methods and compare each method against it.
3. Formulate a priori hypotheses about the method's requirements and derive a framework from the empirical evidence in several methods .
4. Define a metalanguage as a communication vehicle and a frame of reference against which you describe many methods.
5. Use a contingency approach and seek to relate the features of each method to specific problems.

Comparison often implies valuing one method over another. Hence, by using the second approach, some important features concerning the method and its adoption are chosen as perspectives through which the methods are analyzed. Our goal, then, is to identify the differences and similarities between our method and the different software development methods.

However, the variety of different approaches leads to a dilemma when it comes to selecting the most suitable one for a project. At the beginning of every project the manager is expected to commit to a development approach. This is often driven by past experience or other projects that are, or have been, undertaken by the organization. Project managers are expected to select the most suitable approach that will maximize the chances of successfully delivering a product that will address the client's needs and prove to be both useful and usable. The choice should clearly relate to the relative merits of each approach. In practice, little work has been conducted in this area and aside from theoretical papers that compare and contrast some of the models listed above. It is unusual to find studies comparing empirical result (Benediktsson, et al. 2006).

(Benediktsson, et al. 2006) was the last published experiment that tried to address the difference between various software development life cycles. The experiment took place at University of Iceland during the winter 2003-2004 as part of a full year, two semester project. The experiment involved 55 student-developers working in fifteen teams developing fifteen comparable products from the same domain. The objective was to investigate the differences in terms of development effectiveness and quality given the different approaches.

5.2 The Model Classification

Typical life cycle approaches to select from include sequential, incremental, evolutionary and agile approaches. Each is likely to be better suited to a particular scenario and environment and to result in certain impacts on the overall effort and the developed products. Below we aim to classify the developed model as sequential, incremental, evolutionary or agile.

- ***Sequential approaches:***

Sequential approaches (e.g. waterfall model,) refer to the completion of the work within one monolithic cycle. Projects are sequenced into a set of steps that are completed serially and typically span from determination of user needs to validation that the given solution satisfies the user. Progress is carried out in linear fashion enabling the passing of control and information to the next phase when pre-defined milestones are reached and accomplished. This approach is highly structured, provides an idealized format for the contract and allows maximum control over the process. On the other hand, it is also resistant to change and the need for corrections and re-work.

Sequential development is also referred to as serial engineering. The serial focus ensures interaction between phases as products are fed into the next step and frozen upon completion of a milestone. This essentially represents a comparison between the input to and the output of each phase. Sequential engineering also implies a long development sequence as all planning is oriented towards a single hand-over date. Explicit linearity offers a structured approach rich in order, control, and accountability. In order to overcome the impact of a late hand-over and delayed feedback, specific decision mechanisms, review point and control gates are introduced to ensure early discovery and correction of errors and a reduced aggregate cost to fix.

It's clear from the above discussion that the developed SDLC can not be classified under the sequential approaches category. First of all, there are many cycles within the main cycle, the progress of work is not linear since it can go back and forth, and the developed SDLC can not be described as high structured but rather it's rely on people to control the process flow. While the sequential approaches are most resistance to change, the developed SLDC attempt to reduce the effect of change by showing its effect directly to the customer, but dealing with the change it self can be handled. Also, all the phases are running concurrently on the developed model, so the phase products are not frozen at the end of the phase.

The developed SDLC matches the sequential approaches in terms of delivery mechanism only, where the running version is implemented on the customer environment in a single shot after the acceptance test.

- ***Incremental Approaches:***

Incremental approaches -the spiral model could also be classified as incremental- emphasize phased development by offering a series of linked mini-projects (referred to as increments, releases or versions) working from a pre-defined requirements specification up front. Work on different parts and phases, is allowed to overlap throughout the use of multiple mini-cycles running in parallel. Each mini-cycle adds additional functionality and capability.

The approach is underpinned by the assumption that it is possible to isolate meaningful subsets that can be developed, tested and implemented independently. Delivery of increments is staggered as calendar time progresses. The first increment often acts as the core product providing the functionality to address the basic requirements. The staggered release philosophy allows for learning and feedback which can modify some of the customer requirements in subsequent versions. Incremental approaches are particularly useful when the full complement of personnel required to complete the project is not available and when there is an inability to fully specify the required product or to fully formulate the set of expectations.

The development SDLC has much in common with the incremental approach, but they differ on the principle of delivery to the customer. While the incremental approaches focuses on prioritizing the customer demands and delivering them on phases, the developed SDLC attempt to build the whole system then delivers it to the customer.

- ***Evolutionary Approaches:***

Evolutionary approaches recognize the great degree of uncertainty embedded in certain projects and allow developers and managers to execute partial versions of the project while learning and acquiring additional information and gradually evolving the conceptual design.

Evolutionary projects are defined in a limited sense allowing a limited amount of work to take place before making subsequent major decisions. Projects can start with a macro estimate and general directions allowing for the fine details to be filled-in in evolutionary fashion. The initial implementation benefits from exposure to user comments leading to a series of iterations. Finite goals are thus allowed to evolve based on the discovery of user needs and changes in expectations along the development route. Projects in this category are likely to be characterized by a high degree of technological risk and lack of understanding of full implications by both stakeholders and developers. Evolutionary approaches are particularly effective in change-intensive environments or where resistance to change is likely to be strong. The developed SDLC can not be described as an evolutionary approach.

- ***Agile Approaches:***

Agile development is claimed to be a creative and responsive effort to address users' needs focused on the requirement to deliver relevant working business applications quicker and cheaper. The application is typically delivered in incremental (or evolutionary or iterative) fashion.

The agile development approaches are typically concerned with maintaining user involvement through the application of design teams and special workshops. The delivered increments tend to be small and limited to short delivery periods to ensure rapid completion.

The management strategy utilized relies on the imposition of time boxing, the strict delivery to target which dictates the scoping, the selection of functionality to be delivered and the adjustments to meet the deadlines. Agile development is particularly useful in environments that change steadily and impose demands of early (partial) solutions. Agile approaches support the notion of concurrent development and delivery within an overall planned context.

The developed SDLC confirms with the agile approaches on the principles related to user involvement, the main difference derived from the incremental delivery nature of agile approaches.

Based on the above analysis we can not classify the developed SDLC as sequential, incremental, evolutionary nor agile, on the other hand, it could be classified as a hybrid model of sequential, incremental and agile approaches.

5.3 Comparison With Other Models

Since the model is classified as a hybrid model of sequential, incremental and agile approaches. We aim to compare it with these models only in terms of advantages and disadvantages, since it can be compared with other methods that have different scheme and orientation.

As indicated in chapter 1, the waterfall model –as a representative of sequential approaches- cumbersome the following disadvantages:

- Changes may cause considerable confusion as the project progresses.
- As The customer usually only has a vague idea of exactly what is required from the software product, this model has difficulty accommodating the natural uncertainty that exists at the beginning of the project.
- The customer only sees a working version of the product after it has been coded. This may result in disaster if any undetected problems are precipitated to this stage.

It's clear that the developed SDLC covered these problems and solved them; the main common activity between the developed model and the waterfall is the delivery scheme, this adds the following advantages:

- The customer need not to think about what will be delivered in the next phase, since the system is delivered in a single shot, so all the requested features should be their from day one.

- The output of the project can be expected.
- Delivering a rigid product to the customer which will increase his confidence in the system.

Even this feature is considered as a weakness point for the waterfall model, we argue that this is no longer a weak point in our model. The cause of the problem on the water fall is that the user is not involved in the project, so when they receive the product they will have a vague idea about its operation. But this dilemma is solved in our model, since the customer is involved on all the phases of the project including development.

The incremental model shares a lot of properties with the developed SDLC, the main advantages of this model are:

- Generates working software quickly and early during the software life cycle.
- More flexible – less costly to change scope and requirements.
- Easier to test and debug during a smaller iteration.
- Easier to manage risk because risky pieces are identified and handled during its iteration.

For the first advantage, the incremental approach differs from the developed approach; this is because even this advantage could be a very big disadvantage for the incremental delivery. The incremental delivery could be misleading to the customer, were they expect to see a feature on a certain phase, but this features could be delayed to later phases. Also this kind of delivery could lead to loss the customer faith in system performance, because the incremental nature of development will tend the developers to focuses on service rather that reliability in the first few delvers.

The last three advantaged is the same for the developed model if applied to modules rather than iterations.

The main disadvantages of the incremental model comes from the customer side, since its very hard for them to prioritize their needs in the first few increment which will lead to change requirements over the phases. Also, the customer does not have a feeling over the development time, so, they may request a feature to be delivered that could take the time of other three but more important increments from the developers perspective.

The agile methods are proven to be the most successful among development methods, especially with small to medium projects. But, due to its incremental delivery approach, it inherits the same disadvantages from incremental approaches related to delivery, which the developed model solves.

The developed model focus in customer involvement as the agile methods, so, both of them face the challenge of keeping the customer interested in working in the project. The developed model solves this problem by assigning a customer side project manager, who focuses in managing the customer side teams. On the other hand, the main reason of the success of agile methods is there focus on the customer involvement, which is the main target of the developed approach.

One of the most related works to the developed model is the Modular-Model proposed In (Maheswar 2002); the life cycle proposed on this model focuses on distributing the requirement into modules to do the development concurrently trying to achieve parallelism and hence faster delivery. The first phase of this model is a “Developer Phase Involving Customer”, the phase name is misleading, because the customer job here is to participate in providing requirement and the developer will collect them validate and then distribute them into modules, so the customer is not involved nor participate in the development work. The same applies for the last phase of this cycle –i.e. “Testing Phase Involving Customer”- since it is an ordinary acceptance test on the customer environment, so there is no new contribution that this work offered. The main disadvantages of this work are:

- Specifications are built based on prototyping mechanism which needs a lot of time to achieve the customer target. In contrast, our proposed model relies on JAD meeting to collect the requirements which proved to be very successful.
- The model focuses on modularity to achieve parallelism, but no tool provided to achieve this. On the other hand, our model distributes the requirement over a behavior tree, so modularity will be achieved based on that tree.

Chapter 6

Contributions, Conclusions and Future Work

The contribution of the presented software development methodology will be illustrated in two main perspectives. At the outset, the benefits that the model has presented will be argued. Then, the effectiveness of the model will be measured. To conclude the study, conclusion and the viewed future work will be discussed in the end of this chapter.

6.1 Model Analysis

The means of this analysis is to show the strengths in the proposed model, together with presenting the benefits from the adopted methods.

6.1.1 *A Five-Phases Model*

The Model contains five well defined phases, in which the boundaries between each phase are obvious, and the role of responsibility is clearly identified. Recognizing the boundaries help the project members to follow the project, knowing their role in the current phase and the expected outcome from them at the end of each phase.

The customer preparation phase was presented to let the customer team understand what they really want, and build comprehensive knowledge about the main goal from adopting the system. Along side with this, this phase will prepare the customer team to work on software projects.

Upon the end of the phase, the customer will have a clear idea about the strengths and weaknesses in the proposed solution. Which will make them value there supplier and understand any incompatible requirements that was not feasible during the brief feasibility study.

The project charter will identify the roles of each party, the customer and the supplier, to remove any confusion between decision makers and visibly identify the goals and the expected deliverables of the software.

The model also offered the customer preparation phase isolated from the requirement phase, because:

- It contains activities that do not need to go into iteration.
- Its process can't be interleaved with other activities, for example defining the project goals can not be interleaved with the development of some requirement, those are mutually isolated, and the development of any requirement is fully dependent on the project goals.

The requirement engineering phase, in the proposed model, is an iterative process, to make sure that the output requirements are as correct as possible, not conflicting, and clearly describe the customer needs.

The requirements phase needs the longest time in the proposed model in comparison with the other phases, because:

- Reworking requirements defects on most software development projects costs between 40% (Firesmith 2003) and 80% (Wiegiers 2001) of total project effort.
- Requirements defects may cost between 10 to 200 times as much if detected in a fielded systems or 10 times as much if detected during testing compared to detection at the requirements stage (Firesmith 2003).
- As much as 60% of all defects in a system lifetime originate from deficient requirements (Berry 2002).
- Any mistake in requirement phase effects the whole cycle, and eventually increase the time to deliver and increase the development cost.

The model requirement, design and development, and testing phases are interleaved, in which, some of the requirements could be processed in the requirement phase, other could be developed and some of them could be tested, all at the same time, the benefits of interleaving includes:

- All the teams -i.e. the requirement team, the development teams and the testing teams- are working concurrently, and no team is setting idle.
- All the teams will keep in touch with the system and will never loss synchronization or familiarity with the project, because they are dedicated to it.
- Increase the time to deliver, which will lead to increase customer and supplier benefits.

The most significant advantage of the closure phase is the project assessment document, which can be used for future reference as a feedback for both the customer and the supplier to identify the factor of success and/or failure of the projects. This will help them to focus on the success factors and reduce the failure factors, for example, the effect of the change in requirements.

6.1.2 *A Tree-Based Data flow*

In addition to its effective rule in the requirement phase, in terms of defect detection - as mentioned in section 3.4- , representing the system in a behavior tree could help in the following aspects:

- Simplifies tracing of requirement –i.e. the phase of each node.
- Simplifies nodes (requirement) reporting between the customer and the supplier.
- Modulating the system into subtrees that can be developed simultaneously.
- Simplifies the selection of testing cases in the acceptance test, and the expected behavior will be obvious.

6.1.3 *A Call-Based Work Flow*

Both the customer and the supplier along with the whole development life cycle will benefit from the proposed calling system:

- It clearly identifies the decision maker in each phase of the project.
- The responsibility comes with the decision; hence, call maker will make deep analysis before issuing any call.
- Controlling the process flow.

- A clear declaration of moving nodes between phases to make all the teams synchronized.
- Clearly identify customer participation in the process, which will force him to be more committed to it, since the mistake will be obvious and documented if it was from the customer end.

6.1.4 *Interactive Customer and Supplier Teams*

The customer and the supplier will work as a single team in the project, from the requirement definition to the project management and until the system testing. Benefits of this method include:

- The customer and the supplier keep each other dedicated to the project by creating a competitive environment and each member will work against putting any mistakes on his side.
- The customer will have a better understanding in the value of change in requirements.
- Any delay in project delivery will be justified from the customer perspective.
- Simplifies the testing for the customer team.
- Flexible adaptation to the software after delivery.

The ultimate goal of this method is to make sure of gaining satisfaction of the customer, which will lead to long term profit of the supplier.

The roles and responsibilities are distributed between the customer and the supplier, which will do well to both ends. For example, having a customer project manager will:

- Increase and guarantee customer's team dedication to project.
- Speed up deliveries from customer end, like requirement clarifications.
- Organize and control the customer teams.
- Provide help to the supplier project manager, because of his wider knowledge on the hosting environment, and interfaces with third party systems.
- Making sure that the supplier project manager and his team is fully focused in the project goals.
- Making sure the project time plan is followed and gives early alarm in case of a proper delay.
- Identify risks from the customer end.
- Reporting the project progress to the customer end stakeholders.

6.2 The Model Effectiveness

Measuring the effectiveness of the proposed model needs a reliable and trusted method that comprehensively analyzes the factors of success and failure in projects. We will measure our model effectiveness based on comparisons with the CHAOS report and the CMMI model.

6.2.1 CHAOS Report Confirmation

One of the measuring references for this model will be the “Standish Group CHAOS Report 2004”; The Standish Group, is a globally respected source of independent primary research and analysis of IT project performance. The CHAOS report - produced by the group- comprises 12 years of research, done through focus groups, in-depth surveys and executive interviews, on project performance of over 50,000 completed IT projects.

The objectives of CHAOS research are to document the scope of application software development project failures, the major factors for failure, and ways to reduce failure. In 1994, The Standish Group made public its first CHAOS Report, documenting the billions of dollars wasted on software development for projects that were never completed. That report is among the most oft-quoted in the industry since then. (Hartmann 2006)

For purposes of the study, the projects were classified in the report into three resolution types:

- Project success: The project is completed on-time and on-budget, with all features and functions as initially specified.
- Project challenged: The project is completed and operational but over budget, over the time estimate, and offers fewer features and functions than originally specified.
- Project impaired: The project is canceled at some point during the development cycle.

The resolution of projects in 2004 is illustrated in figure 38.

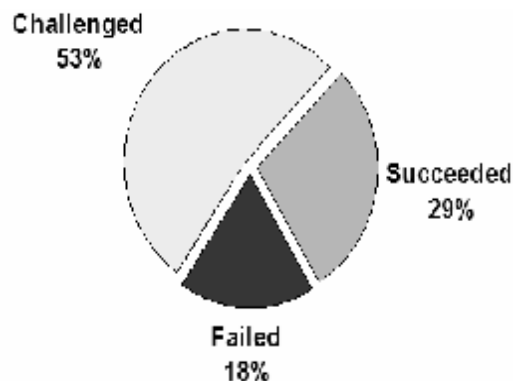


Figure 38: CHAOS 2004 projects resolution

Figure 39 shows the change in projects resolution from 1994 until 2004.

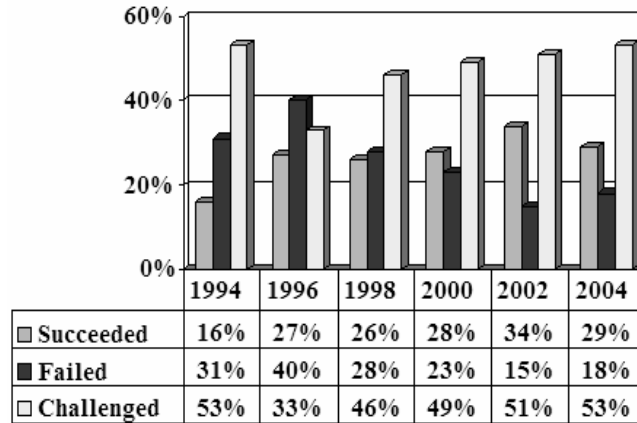


Figure 39: Change in projects resolution (1994-2004)

Figure 40 shows the average percentage of cost overrun over 1994 till 2004, and figure 41 illustrate the average percentage in the time overrun during the same period of time.

The most important aspect of the research is discovering why projects fail. To do this, The Standish Group surveyed IT executive managers for their opinions about why projects succeed. The three major reasons that a project will succeed are user involvement, executive management support, and a clear statement of requirements.

There are other success criteria, but with these three elements in place, the chances of success are much greater. Without them, chance of failure increases dramatically. The project top ten success factors was:

1. User involvement 15.9%
 2. Executive management support 13.9%
 3. Clear statement of requirements 13.0%
 4. Proper planning 9.6%
 5. Realistic expectations 8.2%
 6. Smaller project milestones 7.7%
 7. Competent staff 7.2%
 8. Ownership 5.3%
 9. Clear vision & objectives 2.9%
 10. Hard-working, focused staff 2.4%
- Other 13.9%

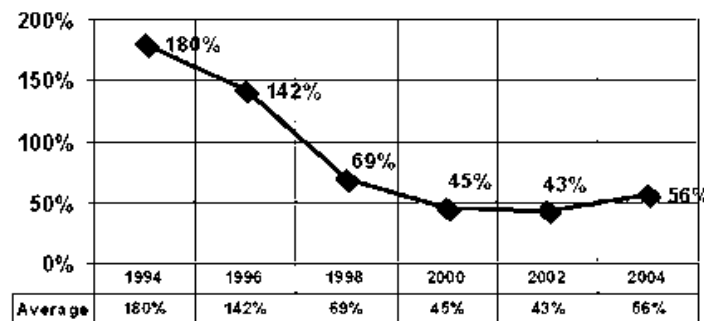


Figure 40: Average percentage of cost overrun (1994-2004)

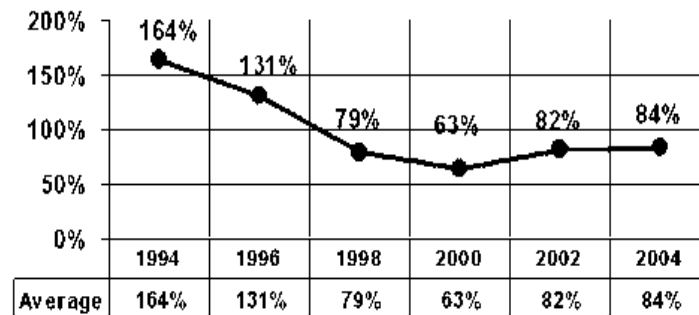


Figure 41: Average percentage of time overrun (1994-2004)

The survey participants were also asked about the factors that cause projects to be challenged. Top ten factors for Project failure was:

1. Lack of user input 12.8%
 2. Incomplete requirements & specifications 12.3%
 3. Changing requirements & specifications 11.8%
 4. Lack of executive support 7.5%
 5. Technology incompetence 7.0%
 6. Lack of resources 6.4%
 7. Unrealistic expectations 5.9%
 8. Unclear objectives 5.3%
 9. Unrealistic time frames 4.3%
 10. New technology 3.7%
- Other 23.0%

Opinions about why projects are impaired and ultimately canceled ranked incomplete requirements and lack of user involvement at the top of the list. The main factors for impaired projects are:

1. Incomplete requirements 13.1%
 2. Lack of user involvement 12.4%
 3. Lack of resources 10.6%
 4. Unrealistic expectations 9.9%
 5. Lack of executive support 9.3%
 6. Changing requirements & specifications 8.7%
 7. Lack of planning 8.1%
 8. Didn't need it any longer 7.5%
 9. Lack of IT management 6.2%
 10. Technology illiteracy 4.3%
- Other 9.9%

Major factors that effect software development can be derived by analyzing the results from the above figures; it can easily noted that the most effective factors in project - listed from the most important to the least important- are:

1. User involvement.
2. Clear statement of requirements.
3. Executive management support.
4. Realistic expectations.
5. Staffing.

6. Proper planning.
7. Clear vision & objectives.
8. Technology.
9. Smaller project milestones.
10. Ownership.
11. Didn't need it any longer.
12. Unrealistic time frames.
13. Lack of IT management.

The results amazingly comply with the process of the proposed model; from its intensive focus on customer involvement, passing by the long and comprehensive requirement phase, along side with enhancing the management role –the stakeholders- in projects.

The model also peruses realistic expectations by involving customer in the development process, and clearly identifies the role of each project team member –the staff- in the process. It can be noticed that, the main factors that affect projects are actually the points of focus of the developed model.

6.2.2 *CMMI Model Measurement*

Since 1991, CMMs have been developed for a large number of disciplines. Some of the most notable include models for systems engineering, software engineering, software acquisition, workforce management and development, and Integrated Product and Process Development. The CMM-IntegrationSM project was formed to sort out the problem of using multiple CMMs. The CMMI Product Team's mission was to combine three source models, the Capability Maturity Model for Software (SW-CMM) v2.0 draft C, the Electronic Industries Alliance Interim Standard (EIA/IS) 731, and the Integrated Product Development Capability Maturity Model (IPD-CMM)SM Into a single improvement framework for use by organizations pursuing enterprise-wide process improvement (Kalayci 2005).

The CMMI assets a model by placing it in a certain maturity level; Maturity levels consist of a predefined set of process areas. The maturity levels are measured by the achievement of the specific and generic goals that apply to each predefined set of process areas. In CMMI models with a staged representation, there are five maturity levels, each layer in the foundation for ongoing process improvement, designated by the numbers one through five as shown in figure 42.

Figure 43 shows a summary of the target profiles that must be achieved when using the continuous representation to be equivalent to maturity levels 2 through 5. Each shaded area in the capability level columns represents a target profile that is equivalent to a maturity level (CMMI Product Team 2005).

Level	Focus	Key Process Areas
5 Optimizing	<i>Continual process improvement</i>	Defect Prevention Technology Change Management Process Change Management
4 Managed	<i>Product and process quality</i>	Quantitative Process Management Software Quality Management
3 Defined	<i>Engineering processes and organizational support</i>	Organization Process Focus Organization Process Definition Training Program Integrated Software Management Software Product Engineering Intergroup Coordination Peer Reviews
2 Repeatable	<i>Project management processes</i>	Requirements Management Software Project Planning Software Project Tracking & Oversight Software Subcontract Management Software Quality Assurance Software Configuration Management
1 Initial	<i>Competent people and heroics</i>	

Figure 42: An overview of the software CMMI levels (Paulk 2001)

Name	Abbr	ML	CL1	CL2	CL3	CL4	CL5
Requirements Management	REQM	2	Target Profile 2				
Project Planning	PP	2					
Project Monitoring and Control	PMC	2					
Supplier Agreement Management	SAM	2					
Process and Product Quality Assurance	PPQA	2					
Configuration Management	CM	2					
Organization Process focus	OPF	3	Target Profile 3				
Organization Process Definition	OPD	3					
Training program	TP	3					
Integrated Software Management	ISM	3					
Software Product Engineering	SPE	3					
Intergroup coordination	IC	3					
Peer Reviews	PR	3					
Software Process Management	OPM	4	Target Profile 4				
Quantitative Quality Management	QQM	4					
Defect Prevention	DP	5	Target Profile 5				
Technology Change Management	TCM						
Process Change Management	PCM	5					

Figure 43: Target Profiles and Equivalent Staging (Paulk 2001)

Analysis of the developed model based on the CMMI is shown in figure 44, from the figure it can be show that the model falls mainly in target profile 3.

ML 2	REQM					
	PP					
	PMC					
	SAM					
	PPQA					
	CM					
ML 3	OPF					
	OPD					
	TP					
	ISM					
	SPE					
	IC					
	PR					
ML 4	OPM					
	QQM					
ML 5	DP					
	TCM					
	PCM					
		CL 1	CL 2	CL 3	CL 4	CL 5

Figure 44: Analysis of the developed model based on the CMMI

6.3 The Model Drawbacks

Some of the identified disadvantages of the model are:

- If a system cannot be properly modularized, defining and building subtrees will be problematic.
- The behavior tree representation makes the method useful on small and middle size projects, but not on large scale projects.
- It has not been employed as much proven models and hence may prove difficult to confidence suppliers to adopt it.

6.4 Conclusions and Future Work

The main purpose of software development is supporting the business functions of some client on a certain field, the aim of this study was to find software development model with attendant methodologies and technologies that focuses on meeting the user requirements as well as improving software quality and productivity, which will increase customer satisfaction.

It was initially assumed that the outcome model aims to enhance the customer contribution to the development process, as well as trying to increase the customer side participation in project management activities. The model tried to accomplish this by building:

- A software development model
- A role model
- A dataflow model
- A work flow model

All of the above models identified clearly the parts of the customer and the supplier and the boundaries between them.

Analysis of the model effectiveness has showed that the proposed model follows the current needs of software development process. On the other hand, the model has some drawbacks, because of the limitation inherited from behavior tree usage.

In order to give a realistic assessment of the effectiveness of the proposed SDLC, the model must be adopted by software engineers and project managers in the field to verify its ability on the ground. The proposed model is still theory and needs actual projects in order to be measured and evaluated. Whatever the measure applied to verify this SDLC, no tool can give accurate answers as real life projects will do. Only years of adoption can answer the question, does this system really work?

Future work will include building software tools to support this development life cycle, these tools include, a behavior tree creation and tracing software, that will be used to create the behavior tree, check in errors in it, and trace the phase of each instant.

Along side with this, deep and thorough analysis must be done to check the ability of proposed role, dataflow and work flow models.

References

- [1] Abrahamson, P., Salo, O., Ronkainen, J., Warst, J. (2002), *Agile Software Development Methods, Review and Analysis*. VIT Publications 478.
- [2] ACM/IEEE-CS Joint Task Force on Software Engineering Ethics and Professional Practices (1999), *Software Engineering Code of Ethics and Professional Practice version 5.2*, ACM.
- [3] Arthur, J. D. and Nance, R. E. (2000), *Verification and Validation without Independence: a recipe for failure*, Virginia Polytechnic Institute and State University.
- [4] Augustine, S., Woodcock, S. (2003), *Agile Project Management*, CC Pace Systems.
- [5] Beckworth, G. and Garner, B. (1994), *An Analysis of Requirements Engineering Methods*, Deakin University (Australia).
- [6] Beck, K (1999a), *Embracing Change with Extreme Programming*. IEEE Computer 32(10).
- [7] Beck, K (1999b), *Extreme Programming Explained: Embrace Change*. Addison -Wesley.
- [8] Behavior Tree Group (2007), *Behavior Tree Notation v1.0*, ARC Center for Complex Systems.
- [9] Benediktsson O., Dalcher D., Thorbergsson H. (2006), *Comparison of Software Development Life Cycles: A Multiproject Experiment*, IEE Proceedings –software.
- [10] Berry, D. M. (2002), *Formal methods: the very idea - some thoughts about why they work when they work*, Science of Computer Programming, 42(1):11–27.
- [11] Boehm, B. and In, H. (1996), *Identifying Quality-Requirement Conflicts*, IEEE Software Vol. 13.
- [12] British Broadcasting Corporation (2007), *Scaling Product Ownership*, Agile Conference 2007.
- [13] Canfora, G. and Cimitile. A. (2000), *Software Maintenance*, University of Sannio.
- [14] CMMI product team (2005), *CMMI[®] for Development: Version 1.2*, by Carnegie Mellon Software institute.
- [15] Coad, P., Lefebvre, E. And De Luca, J. (200), *Java Modeling In Color With UML: Enterprise Component And Process*. Prentice Hall.
- [16] Cornelissen, B., van Deursen, A., Moonen L. and Zaidman, A. (2007), *Visualizing Test suites to Aid in Software Understanding*, 11th European Conference on Software Maintenance and Reengineering.
- [17] David Norton (2007), *Agile Essence: Dynamic System Development Method*, Gartner (G00150567).
- [18] Davis, A.M.; Bersoff, E.H.; Comer, E.R. (1988), *A strategy for comparing alternative software development lifecycle models*, IEEE Transactions on Software Engineering, Volume 14, Issue 10.
- [19] de Barros Paes, Carlos Eduardo Hirata, Celso Massaki (2007), *RUP Extension for the Development of Secure Systems*, Fourth International Conference on Information Technology.
- [20] ESA Board for Software Standardization and Control (BSSC) (1995), *Guide to Software Validation and Verification*, European space agency.

- [21] Firesmith, D. (2003), *The business case for requirements engineering*, SEI of Carnegie Mellon University.
- [22] Fowler, M. (2005), *The New Methodology*, <http://www.martinfowler.com/>
- [23] Gray, A. Jackson, A. Stamouli, I. Shiu Lun Tsang (2006), *Forming successful extreme programming teams*, Agile Conference 2006.
- [24] Greg Thomas and Neal Millett (2007), *Fairfax County Virginia GIS Planning, Design and Implementation - The Critical Steps*, [<http://gis2.esri.com/library/userconf/proc00/professional/papers/PAP416/p416.htm>], 16/oct/2007.
- [25] Haag, Cummings, Mccubbrey, Pinsonneult, and Donovan (2006), *Information Management Systems, For the Information Age. Phase 2: Analysis*. McGraw-Hill Ryerson.
- [26] Hartmann D. (2006), *Interview: Jim Johnson of the Standish Group*, InfoQ.com, Posted on Aug 25, 2006.
- [27] Highsmith, J.A (2000), *Adaptive Software Development: A Collaborative Approach to Managing Complex Systems*. New York, NY, Dorset House Publishing.
- [28] Hofstrand, D., Holz-Clause, M. (2006), *what is a Feasibility Study?*, IOWA State University
- [29] Hoh In; Rick Kazman and David Olson (2001), *From Requirements Negotiation to Software Architectural Decisions*, Texas A&M University.
- [30] IBM Corporation (2005), *Rational: Rational Edge ezine*, <http://www.ibm.com/developerworks/rational/rationaledge/index.html>.
- [31] IEEE-SA Standards Board (2004), *IEEE P1012/D12 Draft Standard for Software Verification and Validation*, IEEE Computer Society Press.
- [32] IEEE-SA Standards Board (1998a), *IEEE Std 830-1998 Recommended Practice for Software Requirements Specifications*, IEEE Computer Society Press.
- [33] IEEE-SA Standards Board (1998b), *IEEE Std. 1219-1998 Standard for Software Maintenance*, IEEE Computer Society Press.
- [34] IEEE-SA Standards Board (1990), *IEEE Std. 610.12 Standard Glossary of Software Engineering Terminology*, IEEE Computer Society Press.
- [35] IMF World Economy Forecast Report (2002), *World Information Technology Revolution*, United Nation Public Administration Network.
- [36] Jackson, M. (1995), *Software requirements & specifications: a lexicon of practice, principles and prejudices*, ACM Press/Addison-Wesley.
- [37] Jenkins, N. (2006), *A Project Management Primer*, <http://www.nickjenkins.net>.
- [38] Jennerich, Bill (1990), *Business Requirements Analysis for Successful Re-Engineering. Joint Application Design*. Unisphere.
- [39] Johnson, D., Sutton, P., & Harris, N. (2001), *Extreme Programming Requires Extremely Effective Communication: Teaching Effective Communication Skills to Students In An IT Degree*, The 18th Annual Conference of the Australian Society for Computers in Learning in Tertiary Education. Melbourne.
- [40] Kagan Erdil; Emily Finn; Kevin Keating; Jay Meattle; Sunyoung Park and Deborah Yoon (2003), *Software Maintenance As Part of the Software Lifecycle*, Tufts University.
- [41] Kalayci, O. (2005), *CMMI versus XP (eXtreme Programming)*, Nitelik SW Process Consultancy.

- [42] Kamata, M. I., Tamai, T. (2007), *How Does Requirements Quality Relate to Project Success or Failure?*, 15th IEEE International Requirements Engineering Conference.
- [43] Kelley Cyr (2002), *Parametric Cost Estimating Handbook*, NASA.
- [44] Kerzner, H. (2006), *Project Management: A Systems Approach to Planning, Scheduling, and Controlling (9th edition)*, John Wiley and Sons.
- [45] Korkala, M. Abrahamsson, P. Kyllonen, P. (2006), *A case study on the impact of customer communication on defects in agile software development*, Agile Conference 2006
- [46] Lanubile, F (2003), *A P2P Toolset for Distributed Requirements Elicitation*, University of Bari.
- [47] Laura Rose (2006), *Involving customers early and often in a software Development project*.
- [48] <http://www.ibm.com/developerworks/rational/library/jan06/rose/index.html#author>.
- [49] Maheswar, U., Sekhar, C., Rao, A.K. and Devsen, K. (2002), *A Software Development Life Cycle Model for Low Maintenance and Concurrency*, Chillarege Press.
- [50] Marek Rychl'y and Pavl'na Tich'a (2007), *A Tool for Supporting Feature-Driven Development*, Brno University of Technology.
- [51] Murugesan, S. (1994), *Attitude towards testing: a key contributor to software quality*, First International Conference on Software Testing, Reliability and Quality Assurance, Conference Proceedings.
- [52] Niessink and van Vliet (2000), *Software Maintenance from a Service Perspective*, Journal of Software Maintenance and Evolution: Research and Practice.
- [53] Palmer, S. R. And Felsing, J. M. (2002), *a Practical Guide to Feature Driven Development*. Upper Saddle River, NJ, Prentice Hall.
- [54] Paulk, M. (2001), *Extreme Programming from a CMM Perspective*, IEEE Software 2001 November edition.
- [55] Pollice, G. (2005), *teaching software development vs. software engineering*, <http://www.ibm.com/>.
- [56] R.Gibson (1992), *Managing Computer Projects*, Prentice-Hall.
- [57] Rising, L. And Janoff, N. S. (2000), *the Scrum Software Development Process for Small Teams*. IEEE Software 18(6).
- [58] Schwaber, K. And Beedle, M. (2002), *Agile Software Development with Scrum*. Upper Saddle River, NJ, Prentice Hall.
- [59] Sol, H. G. (1983), *A Feature Analysis of Information Systems Design Methodologies: Methodological Considerations*, Elsevier.
- [60] Somerville (2004), *Software Engineering 7th edition*, Addison Wesley.
- [61] Song, X. and Osterweil, L. J. (1991), *Comparing Design Methodologies Through Process Modeling*, 1st International Conference on Software Process, IEEE CS Press.
- [62] Song, X. and Osterweil, L. J. (1992), *Toward Objective, Systematic Design Method Comparisons*, IEEE Software 9(3): 43-53.
- [63] Standish Group (1995), *Standish Research Paper*, www.standishgroup.com/chaos.html.
- [64] Standish Group (2004), *Standish Research Paper*, www.standishgroup.com/chaos04.html.
- [65] State of North Dakota (2002), *Project Charter*, MAXIMUS.

-
- [66] Sulaiman, T. Barton, B. Blackburn, T. (2006), *AgileEVM - earned value management in Scrum Projects*, Agile Conference 2006
- [67] Takeuchi, H. And Nonaka, I. (1986), *the New Product Development Game*. Harvard Business Review Jan. /Feb.
- [68] Talby, D., Hazzan, O., Dubinsky, Y. and Keren, A. (2006), *Agile software testing in a large-scale project*, IEEE Software (Volume: 23 , Issue: 4)
- [69] Texas Project Delivery Framework (2007), *Project Charter Instructions*, Business Justification.
- [70] Thompson, A. (2005), *Business feasibility study outline*, <http://bestentrepreneur.murdoch.edu.au/>.
- [71] Don J. Wessels (2007), *The Strategic Role of Project Management*, Published in PM World Today.
- [72] Wickham, P. (2006). *Strategic Entrepreneurship Harlow*, FT Prentice Hall.
- [73] Wiegers, K. E. (2001), *Inspecting requirements*, StickyMinds.com Weekly Column, 30 July 2001.
- [74] Zheng, X. and Dromey, R.G. (2001), *Making Requirements Defect Detection Repeatable*, Software Quality Institute, Griffith University
- [75] <http://En.Wikipedia.Org/>, 22/nov/2007.
- [76] <http://www.bitpipe.com/tlist/Joint-Application-Development.html>, 13/Oct/2007.
- [77] <http://www.cs.odu.edu>, 9/Jan/2008.

Appendices

Appendix A1: Behavior Tree Notation

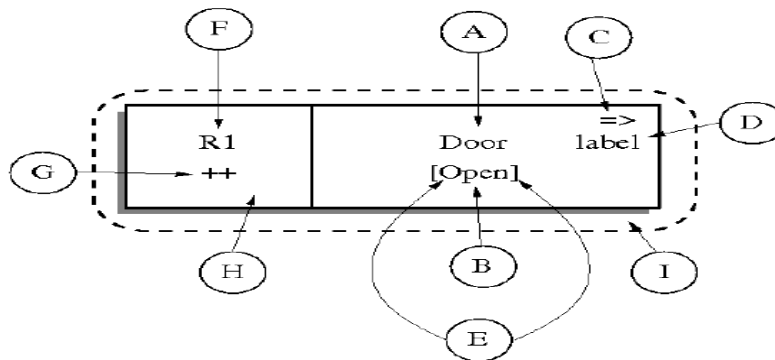
Reference: Behavior Tree Group (2007), *Behavior Tree Notation v1.0*, ARC Center for Complex Systems.

A1.1 Naming Conventions

Variable Naming Conventions

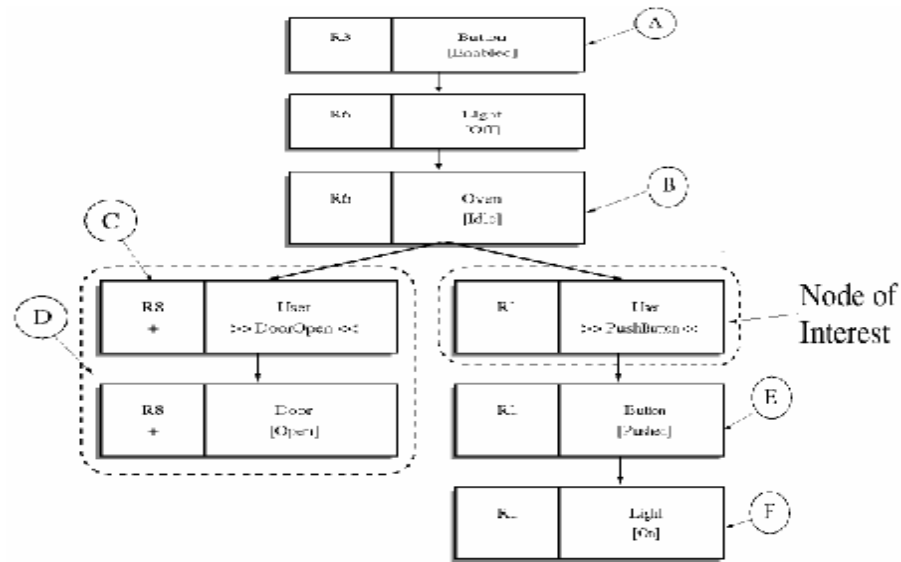
Variable	Description
N, N_i	Behavior Tree Nodes
T, T_i	Behavior Trees
C, C_i	Components
$C\#$	A Component Instance
s	A State of a Component
e	An Event
a	An Attribute of a Component
b	A Branching Condition of a Component

Node Concrete Syntax



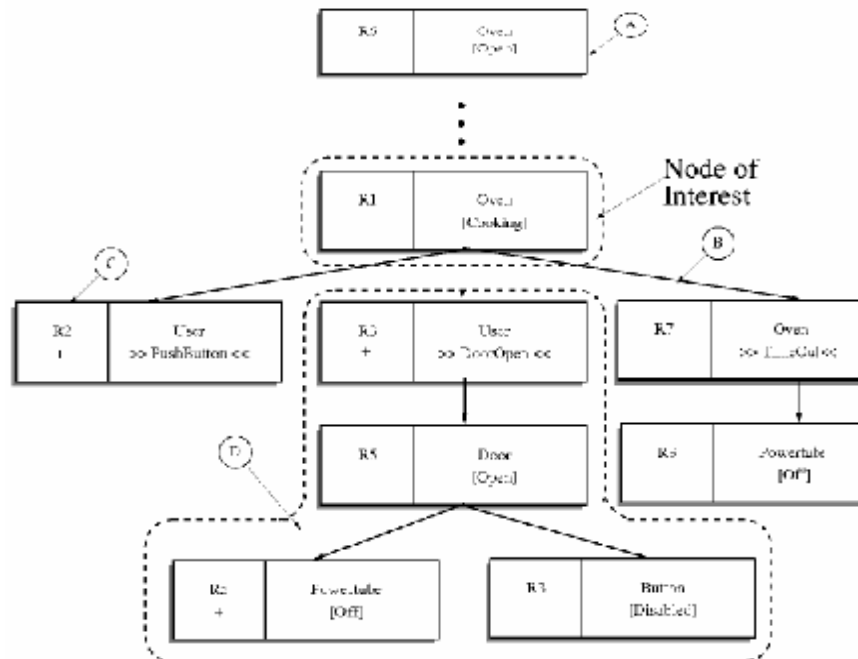
Label	Name	Description
A	Component Name	Specifies a component
B	Behavior	Specifies the behaviour associated with the component
C	Operator(s)	Indicates behaviour of this node is dependent on another node in the tree
D	Label	An optional label for disambiguation (in case a node appears elsewhere with the same component and behaviour)
E	Behavior Type	Delimiters on the behaviour indicate the type of behaviour involved
F	Traceability Link	A reference to the requirements document
G	Traceability Status	Indicates how the node relates to the link
H	Tag	The box on the left-hand side of the node (by default, contains traceability information, but may be used differently, or omitted, in different contexts)
I	Behavior Tree Node	

Tree Naming Conventions



Label	Name	Description
A	Ancestor Node	Any node which appears in a direct line between the node of interest and the root node of the tree
B	Parent Node	An immediate ancestor
C	Sibling Node	A node which shares the same parent
D	Sibling Branch	A (sub)tree with a sibling node as its root
E	Child Node	A node immediately below
F	Descendant	Any node appearing below

Tree Branch Naming Convention



Label	Name	Description
A	Root Node	The first node in a tree (does not have a parent)
B	Edge	
C	Leaf Node	A node with no children
D	Subtree	A tree contained within the tree rooted at the node of interest
	Branch	A synonym for subtree

A1.2 Behavior Tree Notation

Behavior Tree Composition

Type	General Definition	Example	Textual Notation	Description
Sequential Composition			$N; T$	Execute N , passing control to T . The behavior of component T may be tracked between N and T .
Atomic Composition			$N_1 \parallel N_2; T$	Execute N_1 immediately followed by N_2 , passing control to T . The behavior of component T may not be tracked between N_1 and N_2 .
Parallel Branching			$N_1(T_1, T_2)$	Execute N , passing control to both T_1 and T_2 .
Alternative Branching			$N_1(T_1, T_2)$	A condition exists in N and, between T_1 and T_2 , depending on which is ready to execute (see Method).

Basic nodes

Type	Graphical Notation	Textual Notation	Description
State Realisation		$C[s]$	Component C realises state s
System State Realisation		$C[s]$	This is a state realisation decorated with a double box to indicate the component is system component in the current context. There can only be one system component in each context.
Selection		$C?s?$	Special
Event		$C??e??$	Wait until event e is received
Guard		$C??s???$	Wait until C is in state S
Internal Output Event		$send\ C.e$	Generate event e internally to system
Internal Input Event		$recv\ C.e$	Wait for event e (from system)
External Output Event			Generate event e and send to environment
External Input Event			Wait for event e to be received from environment
Empty Node		$skip$	Empty Nodes when used with labels can be used as origins or destinations of node operators.

Node Operators







Type	Graphical Notation	Textual Notation	Description
Reference		$N \Rightarrow$	Behave as the destination tree. The destination node must appear in an alternative branch to the origin.
Reversion		$N \wedge$	Behave as the destination tree. The destination node must be an ancestor. All sibling behaviour is terminated.
Branch Kill		$N \dashrightarrow$	Terminate all behaviour associated with destination tree
Synchronisation		$N \equiv$	Wait for destination node (or nodes)
May		$N \%$	The node may execute normally, or may have no effect
Start new		$N \wedge \wedge$	As with reversion, but sibling behaviour is not terminated

Condition operators			$N_1 \text{ op } N_2$	The operator <i>op</i> may be one of $\&$, $ $, or <i>XOR</i> , corresponding to logical conjunction, disjunction and exclusive or
---------------------	--	--	-----------------------	--

Multiple Component Instances

Type	Graphical Notation	Description
For All		Execute an instance of <i>T</i> for every element in <i>CSET</i>
For Some		Execute an instance of <i>T</i> for some number (including 0) of elements in <i>CSET</i>
At Least One		Execute an instance of <i>T</i> for some number (but at least 1) of elements in <i>CSET</i>
For One Arbitrary		Execute an instance of <i>T</i> for one element in <i>CSET</i>

Node Tags

Type	Graphical Notation	Description
Original Behavior		No Traceability Status indicates that the behavior is stated in the original requirements. The color "green" is used for original requirements.
Implied Behavior		The "+" Traceability Status indicates that the behavior is not explicitly stated in the original requirement but is implied by the requirement. The color "yellow" is used for implied behavior.
Missing Behavior		The "-" Traceability Status indicates that the behavior is missing from the original requirements and is needed for completeness. The color "red" is used for missing behavior.
Updated Behavior		The "++" Traceability Status indicates that the behavior has been added in the post-development (PD) or maintenance phase. The color "blue" is used for updated behavior. Where there are different series of changes / upgrades we use ++V1.0, ++V2.0, etc to indicate the particular upgrade series.
Deleted Behavior		No Descriptions Yet
Design Refinement Behavior		The "+-" Traceability Status indicates that the behavior is a refinement of the original requirements, indicating that the behavior is implied but the detail to describe it is missing.

Appendix A2: Curriculum Vitae

Yazan Al-Masa'fah

Date & place of Birth: 10/Feb/1983, Amman

Marital Status: Married

Nationality: Jordanian

Home +962-64250122

Cell +962-788414063

Yazan.Masafah@gmail.com

Education

Hashemite University *Al-Zarqa, Jordan*

B.Sc. Degree in Electrical & Computer engineering, June 2005

Graduation project: **Building a Security System for the University LAN**,

gained a grade of **A+**

Graduation Rating: **Very Good**

Elite secondary schools *Amman, Jordan*

Altawjihi Examination with grade of **92.6 %**, July 2001

Professional Experience

Huawei Technologies Co., Ltd.

Amman, Jordan

A&S Pre-Sales Engineer (21/Oct/2007 - present)

(Full Time Employee)

Product management of IN and VAS services, including marketing, pre-sales, design and bidding activities.

Orange™ – Jordan Telecom Group (JTG)

Amman, Jordan

VAS Engineer (29/May/2006 – 20/Oct/2007)

(Full Time Employee)

VAS Projects Management, Design, Planning, Implementation and integration. In charge of VAS nodes (SMSC, IVR, Voice Mail, Auto dialer, MCA, cRBT and Voice SMS). Responsible of VAS SMS, MMS, WAP, content, J2ME services.

Accomplishments

- JTG Technical Representative of France Telecom Group Jordan TechnoCenter for Voice SMS service, Oct/2007.
- Implementing Voice SMS service for FT Africa & Asia countries (Project Manager), Oct/2007.
- Implementing Voice SMS service for JTG (Project Manager), Oct/2007.
- Implementing new IVR & Voice Mail system for JTG (Vise Project Manager), Oct/2007.
- JTG rebranding into Orange™ (Project Manager), Aug/2007.
- Implementing Skip DB feature on the SMS to reduce the load on the SMSC database, May/2007.
- Launching of MobileCom's GPRS Modem (Project Manager), Apr/2007.
- Upgrading the SMPP machines from UPU to Langley machines, with upgrading the SMPPs & SFE release from 2.6.14 to 2.6.146, in order to increase the number of available SMPP clients from 100 to 200 ports, Apr/2007.
- Launching a new auto-dialer for JTG & Upgrading the existing system for MobileCom (Project Manager), Mar/2007.
- Launching a Bulk SMS system for MobileCom (Project Manager), Feb/2007.
- Launching of MobileCom's cRBT service ph.2 (Project Manager), Aug/2006
- Launching more than 50 SMS, MMS, WAP, content, J2ME services.

MobileCom

Amman, Jordan

Roaming & Interconnect Engineer (6/Mar/2006 – 28/May/2006)

(Part Time Employee)

Taking part in making roaming agreements with other operators, Testing and verification of roaming services for GSM and GPRS.

Jordan Telecom Group

Amman, Jordan

IN & Voice Mail Engineer (5/Jan/2006 –28/May/2006)

(Full Time Employee)

Responsible for IN Operation & Maintenance, NetManager Operation & Maintenance, as well as Voice Mail, Pre-paid Cards, Pre-paid Telephone, and Flexible Routing & Charging services.

Accomplishments

- Upgrading JT IN platform from Alcatel to Ericsson, in order to increase the license & to add more flexible charging features, Apr/2006.
- Testing of Voice VPN for JTG over Alcatel IN platform, Feb/2006

Jordan Telecom Group

Amman, Jordan

Maintenance Computer Engineer (26/Sep/2005 – 4/Jan/2006)

(Part of Jordanian Engineers Organization Training Program)

Maintenance of personal computers, lap-tops and printers, along side with Windows XP maintenance.

Ardico of Jordan Company (UNISYS® computers agents)

Amman, Jordan

Networking Computer Engineer (22/Jun/2004 – 22/Aug/2004)

(Part of Hashemite University Training Program)

Taking part in Implementing and Troubleshooting IP networks in various locations.

Training Courses

- Object oriented design with C++ programming language, Hashemite University, Al-Zarqa, 2003.
- Microsoft® Windows® 2000 Network and operating system Essentials, Hashemite University, Al-Zarqa, 2003.
- Cisco Networking Academy Program (CCNA), Princes Eman's Center, Amman, 2004.
- C#.NET programming, Hashemite University, Al-Zarqa, 2005.
- Insight IVR and Voice Mail O&M, Global Learning Organization, Milan/Italy, 2007.

Skills & Qualifications

- Fluency in Arabic and English (Written and spoken).
- Exceptional communication & negotiation skills.
- Effective team working & leading skills.
- Excellent documentation, technical writing & presentation expertise.
- Proficiency in Microsoft Office® (Word, Excel, PowerPoint, Access, Project & Visio).
- Programming by C++, C#.NET & Perl languages.
- Maintenance of Personal Computers & Servers.
- Strong knowledge on UNIX, Linux & Windows Operating Systems.
- Working with Oracle & mysql Databases environments.
- Strong Background on TCP/IP Networking & Security.