DOKUZ EYLÜL UNIVERSITY

GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES

PROCESS DIVERSITY IN SOFTWARE DEVELOPMENT

by

Serra ŞAHİN

December, 2018

İZMİR

PROCESS DIVERSITY IN SOFTWARE DEVELOPMENT

A Thesis Submitted to the
Graduate School of Natural and Applied Sciences of Dokuz Eylül University
In Partial Fulfillment of the Requirements for the Degree of Master of Science
in Computer Engineering, Applied Computer Engineering

by
Serra ŞAHİN

December, 2018
İZMİR

M.Sc THESIS EXAMINATION RESULT FORM

We have read the thesis entitled **"PROCESS DIVERSITY IN SOFTWARE DEVELOPMENT"** completed by **SERRA ŞAHİN** under supervision of **ASST. PROF. DR. SEMİH UTKU** and we certify that in our opinion it is fully adequate, in scope and in quality, as a thesis for the degree of Master of Science.

[illegible]

Asst. Prof. Dr. Semih UTKU

Supervisor

[illegible]	[illegible]
Prof. Dr. Mehmet Ünlütürk	Prof Dr Alp Kut
(Jury Member)	(Jury Member)

[illegible]

Prof.Dr. Kadriye ERTEKİN

Director
Graduate School of Natural and Applied Sciences

ACKNOWLEDGEMENTS

I would like to thank to my supervisor, Asst. Prof. Dr. Semih UTKU, for his support, supervision and useful suggestions throughout this study. Also I would like to thank to Prof. Dr. Mehmet Süleyman ÜNLÜTÜRK for his support and suggestions.

Finally, I would like to offer my special thanks to my family, husband and friends for their support, patience and help. It would not have been able to complete this thesis without their support and help.

Serra ŞAHİN

PROCESS DIVERSITY IN SOFTWARE DEVELOPMENT

ABSTRACT

Software companies need to employ state-of-the-art technologies to meet customer requirements. Owing to the complexity of contemporary software products and uncertainty concerning the budget for the required resources, companies use process-oriented quality management techniques in order to guarantee appropriate product quality. In this light, standardized quality assurance system that uses the Capability Maturity Model Integration–Development (CMMI-DEV) program are incorporated into software development processes, and software companies aim to enhance productivity by acquiring CMMI-DEV certificates. In this study, an approach based on CMMI-DEV is used to develop software process diversity for an international company. The results show that the proposed software process diversity model can be used to assess and improve processes in middle size Software Company that has identical technical framework and similar business structure.

Keywords: Capability maturity-model integration-development (CMMI), hybrid methodologies, software development life cycle (SDLC)

YAZILIM GELİŞTİRMEDE SÜREÇ ÇEŞİTLİLİĞİ

ÖZ

Yazılım şirketleri, müşteri gereksinimlerini karşılamak için en son çıkan teknolojileri doğru bir uygulama sistemi ile kullanmalıdırlar. Bu nedenle şirketler, çağdaş yazılım ürünlerinin karmaşıklığında ki, gerekli kaynak kullanımının belirlenmesinde ki ve bütçeyle ilgili belirsizlikte ki sorunları aşabilmek için ve uygun ürün kalitesini garanti etmek için süreç odaklı kalite yönetimi tekniklerini kullanırlar. Bu kapsamda standartlaştırılmış kalite güvence sistemi olan Bütünleşik Yetkinlik Olgunluk Modeli – Geliştirme (CMMI-DEV), yazılım geliştirme süreçlerine de kullanılmış ve yazılım şirketleri bu CMMI-DEV sertifikalarını alarak üretkenliği artırmayı hedeflemiştir. Bu çalışmada ise, uluslararası bir şirketin bu kapsamda ki sorunları için CMMI-DEV tabanlı bir yaklaşım içinde yazılım süreci çeşitliliği kullanılmıştır. Sonuçlar, önerilen yazılım süreci çeşitlilik modelinin, aynı teknik çerçeveye ve benzer iş yapısına sahip olan orta ölçekli yazılım şirketindeki süreçleri değerlendirmek ve iyileştirmek için kullanılabileceğini göstermektedir.

Anahtar Kelimeler: Bütünleşik Yetkinlik Olgunluk Modeli (CMMI), melez metodolojiler, yazılım geliştirme yaşam döngüsü (SDLC)

CONTENTS

LIST OF FIGURES

Page

LIST OF TABLES

Page

CHAPTER ONE
INTRODUCTION

1.1 General

Software organizations define processes in the product development phase; they standardize these processes and continually improve the models to increase its quality and deliver the product on time e (Curtis, 2000). The choice of software development process used significantly influences the quality and on-time delivery of the product (Mathai, M.K., Venugopal, R., Abraham, 2016)(Aaen, Arent, Mathiassen, & Ngwenyama, 2001; Ganpatrao Sabale & Dani, 2012; Ramasubbu & Balan, 2009). There are numerous such software methodologies, development processes and frameworks available for use by software teams. Software processes guide the software team by partitioning the work into manageable tasks; therefore, it is necessary to elaborate on the process definitions. In particular, when defining the software process, the boundaries of work, objectives of the project, and number of resources allocated must be well defined. The number of resources and hardware to be used in these processes can hence be determined efficiently and accurately (Lindvall & Rus, 2000). Moreover, the flexibility of the organization should be considered in the stages of process improvement. In the current context, flexibility is the capacity to use existing capabilities and explore new opportunities. Furthermore, collaborative trust is based on institutional dialogue and common goals (Adler, 2013; Heckscher, Adler, & Paul, 2008; O'Reilly & Tushman, 2011). For these reasons software development firms must be able to standardize their development processes to ensure continual improvement in the software development lifecycle (SDLC). Standardization contributes to improvement in productivity, quality, and schedule planning (Agrawal & Chari, 2007; Chrissis, Konrad, & Shrum, 2003; Glazer, Dalton, Anderson, Konrad, & Shrum, 2008). To achieve standardization, the Capability Maturity Model Integration (CMMI) (a software process improvement (SPI) type) can be used (Kuhrmann, 2015; Niazi, 2015). The concept of CMMI-DEV is not concerned with implementation within processes, but with process improvement in development environments (Garzás & Paulk, 2013). The latest version of CMMI-DEV is widely used and can continually improve the maturity model of software development processes (Chrissis et al., 2003; J. et al., 2014; Kuhrmann, 2015; Niazi, 2015). Moreover, CMMI-DEV yields a dual benefit for managing, measuring and monitoring software development processes because CMMI provides continual improvement while eliminating the deficiencies of the SDLC processes.

In light of the abovementioned applicability and usefulness of process improvements, there is need for further collaboration between concepts in engineering and management for continual improvement of organizational processes. Therefore, it is necessary to use CMMI-DEV for management problems and SDLC methodologies to handle technical issues. To achieve this, process implementations, excluding infrastructure-related issues, should align key process areas of each methodology with those of CMMI-DEV. In particular, CMMI-DEV has been applied to key methodologies to ensure good engineering practices, management structures, and adequate institutionalization (Glazer et al., 2008; Paulk, 2001). In such a collaboration, the selected methodology determines to the steps used to improve key process areas; thus, it is necessary to determine the effects of these process areas on CMMI-DEV.

Resource-light methods have become increasingly popular; these methods are based on the type and size of business. In many software applications (Web, mobile), such methods are used to describe processes. In examining their effect on the development process, it is necessary to assess software development processes because such assessments are useful in guiding software development projects. Furthermore, they are more suitable for small- or medium-sized projects than for large-sized projects. For this reason, the design and selection of CMMI-DEV applications in management processes of large and risky projects in conjunction with lightweight methods is appropriate. The goals and practices of CMMI are rendered more effective when it is used with activities involving lightweight methods(Garzás & Paulk, 2013; Glazer et al., 2008; Paulk, 2001; Sutherland, Jakobsen, & Johnson, 2007). However, CMMI is similar to traditional model with strong documentation requirements and an iterative structure in its levels. Considering this, all the goals and practices of a given leveling process must be completed before moving on to a new level.

However, CMMI is not an SDLC-type but an SPI-type methodology. It is therefore used as a standardization rule for lightweight software methods. In other words, lightweight software methods are used to answer the “how” on CMMI practices and this opportunity let companies customize it easily (Glazer et al., 2008). However, some problems arise when CMMI is used in such a manner. Even lightweight software methods used as models are supported using CMMI; therefore, software development processes revert to purely lightweight methods when process improvement is neglected. Therefore, the model used in improvement processes must be developed or organized using robust structures (Hneif & Ow, 2009). Because traditional methodologies are considerably rigid, requiring substantial resources and documentation, and adapt slowly to change, companies have abandoned them (Cho, 2009; Mathai, M.K., Venugopal, R., Abraham, 2016; Stephen & Oriaku, 2014).

When examining the literature on the strengths and weaknesses of software development methodologies in light of the above discussion, any selected methodology needs to address the problem of quality and time owing to the cost and labor incurred in the development phase of a given project (Balaji, 2012; Madachy, Boehm, & Lane, 2006; Mohammed, Munassar, & Govardhan, 2010; Pawar, 2015; Preeti & Saru, 2014; Stephen & Oriaku, 2014; Stoica, Mircea, & Ghilic-Micu, 2013). In this case, instead of using purely lightweight methods or traditional systems, a model consisting of a combination of multiple software development processes is required to implement software process diversity (Ramasubbu, Bharadwaj, & Kumar Tayi, 2015). Hence,

- software teams can better adapt to changing user requirements and design specifications;
- with the diversity in the capabilities of processes, teams can address and resolve conflicting requests in the project;

- software teams using traditional methodologies can become flexible by adopting certain components of lightweight methods to process frameworks; and
- software teams using similar lightweight methods can use some structural elements and formal documentation linked to plan-based process approaches to improve productivity and overall predictability (Harris, Collins, & Hevner, 2009; Ramasubbu & Balan, 2009; Ramasubbu et al., 2015; Ramesh, Mohan, & Cao, 2012)(Subramanyam, Ramasubbu, & Krishnan, 2012).

This study aims to guide the workings of a software company experiencing problems and determine solutions for process improvement. Furthermore, the results of a software development improvement process, and the gains achieved by the software firm in transitioning from existing software processes to software process diversity with a flexible organizational structure are evaluated.

In this paper, Chapter 2 presents related works in the field. Chapter 3 and Chapter 4 show related topic definition and structure like SDLC and CMMI. The scope of study, population and study area, data collection instruments, and improvement process are given in Chapter 5. The results of a software development improvement process, and the gains achieved by the software firm in transition from existing software processes to software process diversity with a flexible organizational structure, are evaluated in Chapter 6, Chapter 7 presents the conclusion and scope for future work.

CHAPTER TWO
RELATED WORK

2.1 Literature Review

In this chapter, reports and published research related to CMMI-DEV, SDLC, and software process diversity models based on such factors as efficiency, timing, quality, and cost have been investigated. In a study on process improvement by Glazer et al. (Glazer et al., 2008), the relation between CMMI and lightweight software methodologies were defined. Similarly, Paulk (Paulk, 2001) investigated the Software Capability Maturity Model (SW-CMM) as basis for an XP model. In another study, using Spanish companies as examples, Garzás and Paulk (Garzás & Paulk, 2013) described a successful relationship between CMMI and the lightweight software method of Scrum. Sutherland et al. (Sutherland et al., 2007) discussed maturity level 5 of the CMMI and Scrum processes and claimed that integrity was obtained between them.

However, Glazer et al. (Glazer et al., 2008) have also argued that CMMI overcomes the differences in definitions between lightweight and conventional models. Certain issues have emerged relating to the "restrictions of lightweight methodologies" according to Hneif and Ow (Hneif & Ow, 2009). In this regard, the common opinion based on a comparison among software models, such as those proposed by Stephan and Oriaku (Stephen & Oriaku, 2014), Balaji (Balaji, 2012), Pawar (Pawar, 2015) Mohammed et al. (Mohammed et al., 2010), Preeti and Saru (Preeti & Saru, 2014), and Stoica et al. (Stoica et al., 2013) show that the disadvantages of each model are constrained by the project size. Sabale and Dani (Ganpatrao Sabale & Dani, 2012) showed the significance of product quality and timing in all SDLC models. Munassar and Govardhan (Mohammed et al., 2010) reported that the primary issues for all models, regardless of the wide range of software development models available, are cost and other resources; therefore, they proposed hybrid models.

Madachy et al. (Madachy et al., 2006) recommended extending the spiral model and using it as a lightweight spiral lifecycle model. Cho (Cho, 2009) reported on the strengths of a combination of the Rational Unified Process (RUP) and Scrum. Furthermore, in their study, Boehm and Turner implemented lightweight software methods (Boehm & Turner, 2005) in standard industry processes by maintaining their specific lightweight characteristics. In addition to these, Vinekar et al. (Vinekar, Slinkman, & Nerur, 2006) and Batra et al. (Batra, Xia, van der Meer, & Dutta, 2010) adopted a more scientific approach to assess the combined applicability of the conventional and lightweight methodologies.

Ramasubbu et al. (Ramasubbu et al., 2015) investigated software process diversity on a project with multiple software process development frameworks. O’Reilly and Tushman (O’Reilly & Tushman, 2011) defined flexibility as the capacity to simultaneously exploit existing capabilities and explore new opportunities, such as collaborative trust based on institutional dialogue and shared purposes (Heckscher et al., 2008).

When all these studies are examined, in summary; relationship and integrity between CMMI and the lightweight software method of Scrum are defined and provided. The advantages of CMMI over lightweight and conventional models are presented and hybrid solutions are proposed accordingly. Lightweight spiral lifecycle model is recommended, and conventional and lightweight methodologies are combined. However, It has not been mentioned how CMMI processes benefit hybrid solutions in software development processes. In this study, the software process diversity processes are provided for an international company and CMMI-DEV and Scrum were combined to enhance software productivity and creativity.

CHAPTER THREE
CAPABILITY MATURITY-MODEL INTEGRATION-DEVELOPMENT (CMMI)

This chapter mentions about definition, history, structure and level of the capability maturity model integration development (CMMI).

3.1 Introduction CMMI

Nowadays, companies want to deliver products and services better, faster, and cheaper. While this product is producing, created modules are increasingly complex. So requirement analysis and resource management are becoming a serious issue. (Software Engineering Institute, 2010)

Now more than ever, an integrated approach is improved so as enterprise-wide solutions are required these solution for solution of these issues. Especially, management entity of organizational, develop the service and product are critical effect to business success. Because, their development are needed management activities as part of business. (Software Engineering Institute, 2010)

The most influential method that is used to resolve the problems of software development is CMMI that is Capability Maturity Model Integration. CMMI provides an opportunity to avoid or eliminate these barriers thanks to continuously improvement are supplied on processes. CMMI consists of best practices that development activities applied to products and services. Therefore, CMMI as a concept of software development processes refers to the model and develop software process may be referred to maturity assessment model.(Software Engineering Institute, 2010)

3.2 History of CMMI

CMMI was budgeted by American department of defense (DoD) in 1984, was released by Software Engineering Institute at (SEI) Carnegie Mellon University in 1986, and the standards in force at that time were for the military and the government. Then CMM (SW-CMM) was published for software and successful. In 2002, this model was released with sector independent, called CMMI v1.1 model. In 2006, the latest version was released, called CMMI v1.2 and accepted by a wider audience. When this version was released, three structures were created due to independent sector. These are CMMI-DEV for Development, CMMI-SVC for Services and CMMI-ACQ. However, CMMI-DEV concept is not concerned with ‘How to be implemented within processes' (Stoica et al., 2013). Therefore, the last version of CMMI-DEV can be used for the continuous improvement in the software development processes maturity model, and is more widely accepted. So CMMI v1.3 is published to support the Agile Software development principles (Garzás & Paulk, 2013; Glazer et al., 2008; Hneif & Ow, 2009; Paulk, 2001),(“SDLC Overview”, 2016; Software Engineering Institute, 2010).

3.3 CMMI and Dimension

Software Engineering Institute has discovered many factors to provide quality products and services while doing research on this subject. The most critical ones of these factors people, procedures and methods, and tools and equipment. These dynamic items are shown in Figure 3.1 The three critical dimensions (Software Engineering Institute, 2010). When connections of between these elements are correctly organized, understanding of associated with the operation of the process, resource management and structures of the construction work to made quality work at less cost. Thus, development processes can be more resistant in features toward to changing the shape (Software Engineering Institute, 2010).

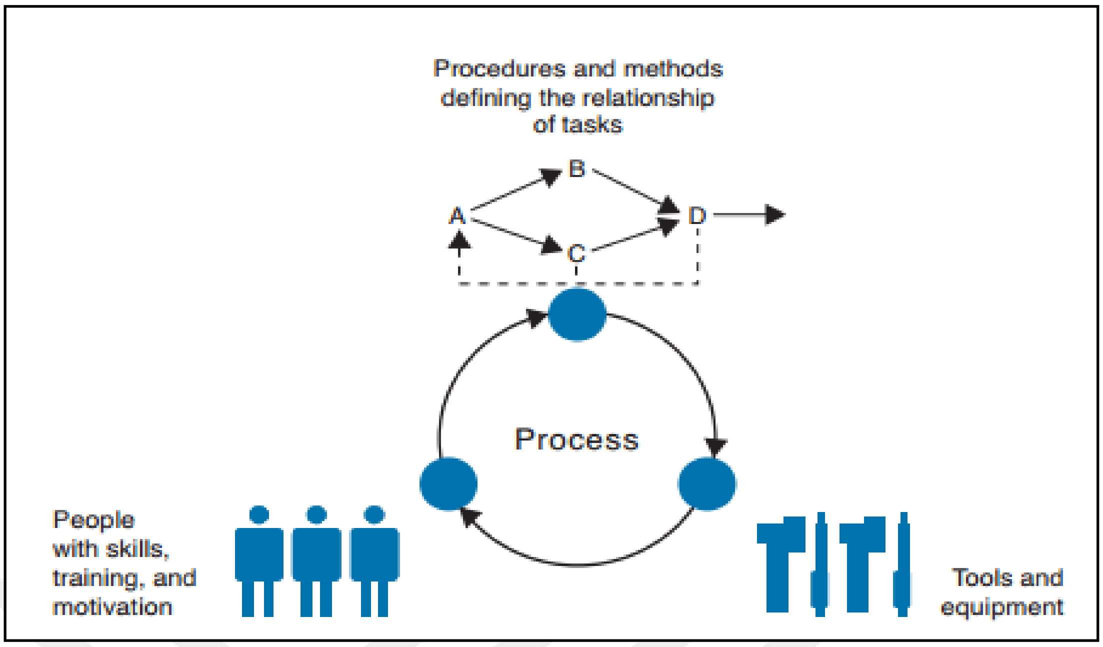

Procedures and methods defining the relationship of tasks
A
B
C
D
Process
People with skills, training, and motivation
Tools and equipment

Figure 3.1 The three critical dimensions (Software Engineering Institute, 2010)

Additionally, connection of these dimension are interested to process area category of CMMI that include project management, process management, systems engineering, hardware engineering, software engineering, and other supporting processes used in development and maintenance in process area category (Software Engineering Institute, 2010).

3.4 Structure of CMMI Model

In every CMMI models, major construction item is “Process Area (PA)” that are focus on the activities of the developer organization. It has 22 process areas and these are shown on Table 3.1 Process Area of CMMI.

Table 3.1 Process Area of CMMI (Software Engineering Institute, 2010)

Name	Abbreviation
Causal Analysis and Resolution	CAR
Configuration Management	CM
Decision Analysis and Resolution	DAR
Integrated Project Management	IPM

Table 3.2 continues

Measurement and Analysis	MA
Organizational Process Definition	OPD
Organizational Process Focus	OPF
Organizational Performance Management	OPM
Organizational Process Performance	OPP
Organizational Training	OT
Product Integration	PI
Project Monitoring and Control	PMC
Project Planning	PP
Process and Product Quality Assurance	PPQA
Quantitative Project Management	QPM
Requirements Development	RD
Requirements Management	REQM
Risk Management	RSKM
Supplier Agreement Management	SAM
Technical Solution	TS
Validation	VAL
Verification	VER

However, process areas are grouped three categories (requires, expected and informative) in structure illustration (“SDLC Overview”, 2016; Software Engineering Institute, 2010). These are;

- Required: In given process area, process improvement is accomplished that are the specific and generic goals
- Expected: In accomplishing a required CMMI component, important activities are described that are specific and generic practices
- Informative: In accomplishing and describing CMMI component, models help to user for understanding.

Each process area is grouped with practices. They are practices aggregately to set of goals to make improvement. Namely a process area has specific and generic goals and each specific and generic goals has own specific and generic practices. Also each goals are implemented with "Required" category, and each practices are implemented with "Expected" category. Goals and practices structure of a process area are shown in Figure 3.2 CMMI model components (Software Engineering Institute, 2010).

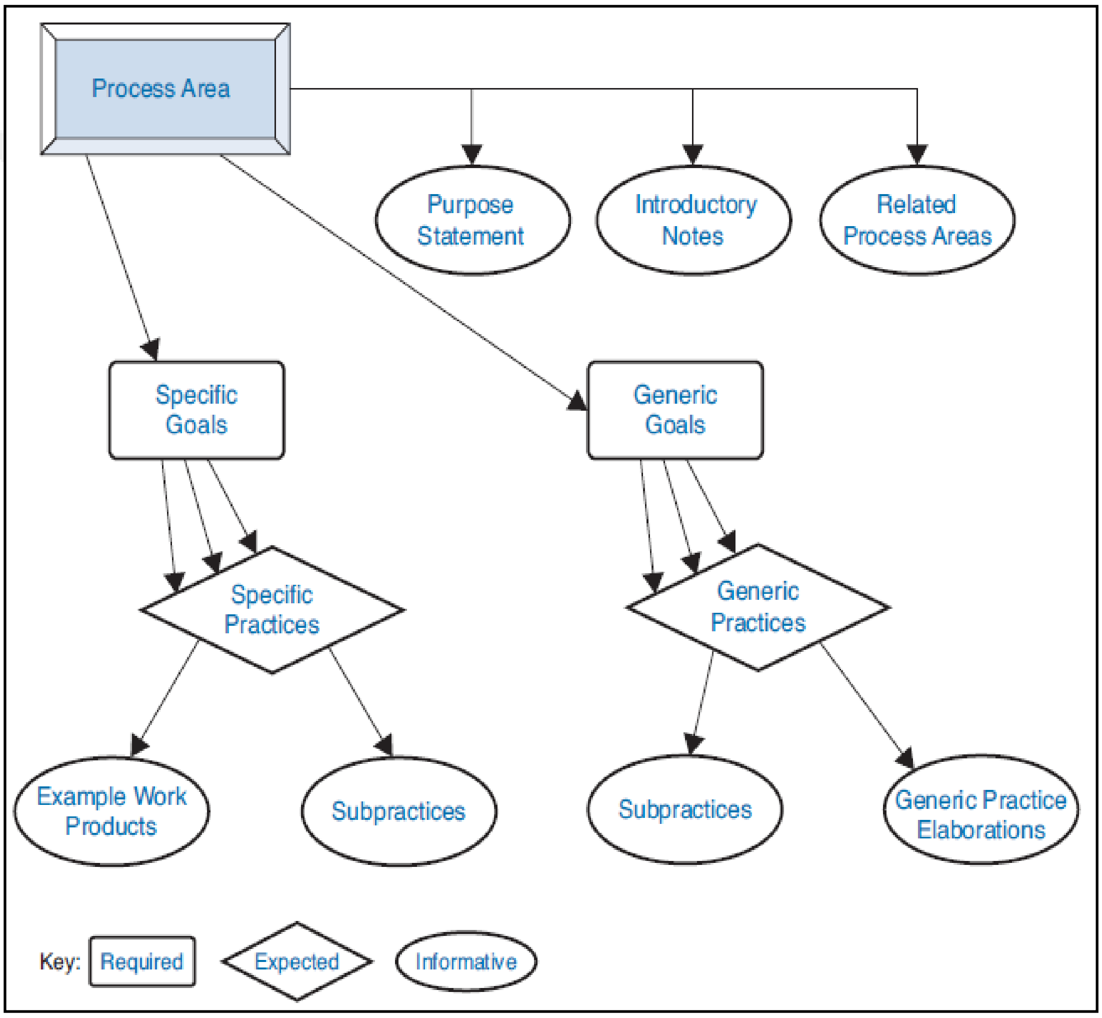


Figure 3.2 CMMI model components (Software Engineering Institute, 2010)

According to CMMI model components definitions;

- Purpose Statements: This component defines aim of process area and it is 'Informative' category
- Introductory Notes Statements: This component defines basic concept in process area and it is 'Informative' category.
- Related Process Areas: This component shows the related process area reference ad relationship among them and it is 'Informative' category.
- Specific Goals: This component defines the unique goals to supply the process area. It is obligatory definition and it is 'Requirement' category.
- Generic Goals: This component defines the goals that are applied on single or multiple process area to institutionalize the implement process area. It is obligatory definition and it is 'Requirement' category.
- Specific Practices: This component define processes for achieving the associated specific goals. It is 'Expected' category.
- Generic Practices: This component defines processes that are applied on single or multiple process area for achieving the associated generic goals. It is 'Expected' category.
- Sub practices: This component defines the details of interpreting and implementing for specific or generic practice. It is 'Informative' category.
- Generic Practices Elaborations: This component define the guidance after generic practices for applying uniquely on process area. It is 'Informative' category.
- Example Work Product: this component lists of sample output from a specific practice. It is 'Informative' category.

3.5 CMMI Representation

While active product or service is developing in CMMI-Dev, processes need to improve. Improvement process of organization is performed with levels. CMMI model is unified under two representations. These are; (“SDLC Overview”, 2016; Software Engineering Institute, 2010)

- Capability Level (continuous representation): Capability levels are a means for continuously improving the process that is selected individual or group of processes area by company. Capability levels are achieved by “continuous representation”. these levels are numbered 0 through 3. These are shown on Table 3.2 Comparison of Capability and Maturity Levels.
- Maturity Level (staged representation): Maturity levels are a means for improving the processes that are selected related set of processes by company. Maturity levels are achieved by “staged representation”. These levels are numbered 1 through 5. These are shown on Table 3.2 Comparison of capability and maturity levels.

Table 3.3 Comparison of Capability and Maturity Levels (Software Engineering Institute, 2010)

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

On the other hand these levels are characterized according to specification. Maturity levels are used to describe relative to model for general state of the organization's processes by the staged representation. It concentrates general maturity that is measured by maturity levels Capability levels are used to describe relative to individual process area for state of organization’s processes by continuous representation. It concentrates a process area that is measured by capability levels.

Further, when correlating between level of CMMI and component of CMMI is define, maturity level is related with process area and capability level is related with generic goals. This explanation is shown on Figure 3.3 Structure of the continuous and staged representations (Software Engineering Institute, 2010).

- Components of Continuous Representation of CMMI
    - Capability Level (continuous representation)
        - General Goal and Generic Practices
    - Categorically Process Areas
    - Profile
    - Other Component

- Components of Staged Representation of CMMI
    - Maturity Levels (Staged Representation)
        - Capability Level (continuous representation)
        - Process Area
        - Generic Goal and Specific Goal
        - Generic Practices and Specific Practices

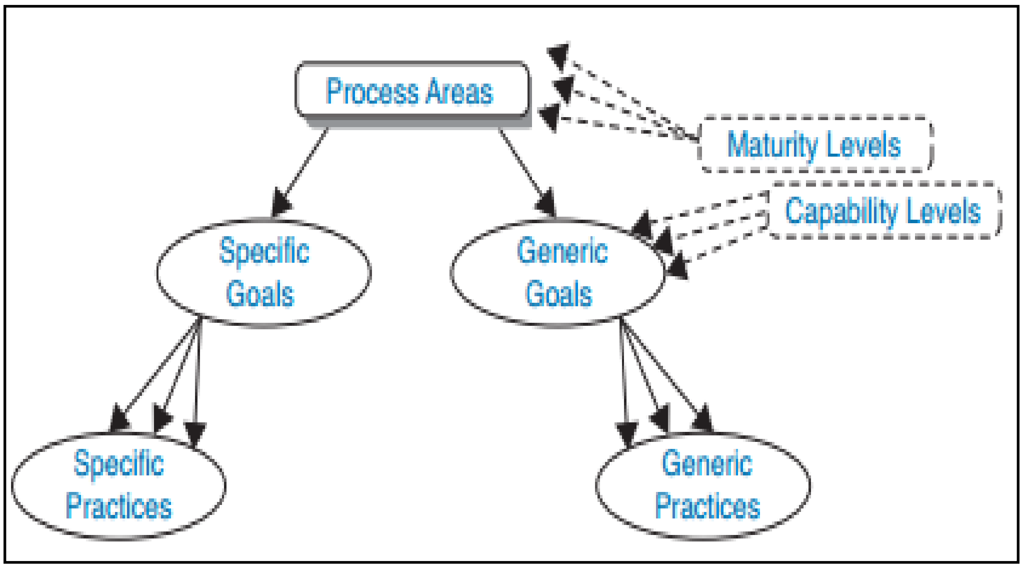


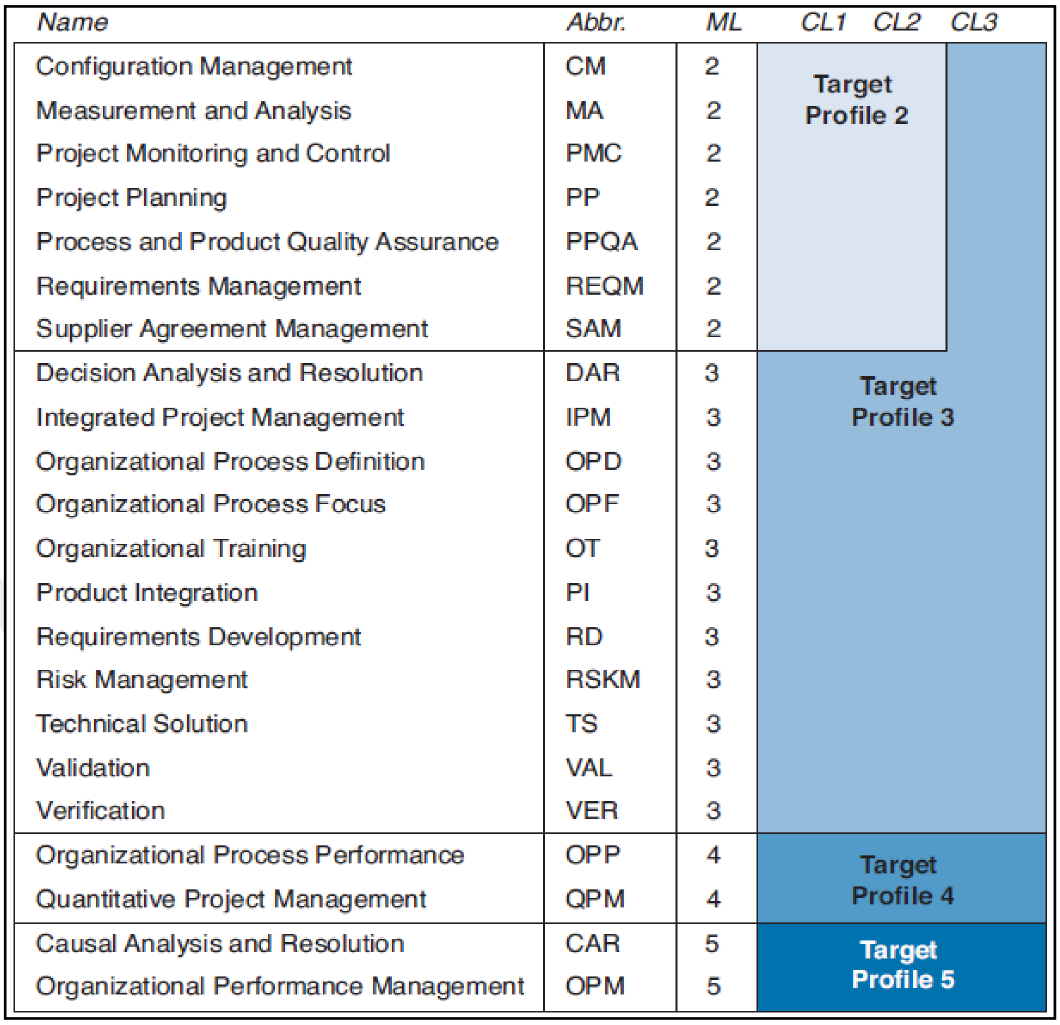
Figure 3.3 Structure of the continuous and staged representations (Software Engineering Institute, 2010)

Maturity level 2 is applied owing to empirical study is limited to level 2. So technical definition is concentrated on maturity level 2 that is staged representation in this study. But before maturity level 2 explanation, maturity levels should be explain to understand the concept of maturity.

3.6 Maturity Level (staged representation)

Maturity level is used to characterize organization's process areas according to all model. The staged representation doesn’t concerned with individual process areas that is status like complete or incomplete. It interests in multiple process areas that are specified with in maturity level. These combinations are shown Figure 3.4 maturity level with process areas (Software Engineering Institute, 2010). Staged representation provide to perfectly, accurately and timely complete work. While it doing, requests, resources, risk and stockholders are planned to provide a way measurement value, perfuming area to improve the processes of organization (Software Engineering Institute, 2010).

As mentioned before, five maturity levels are numbered 1 through 5. First level does not need specific or special effort for improving process. Because each company processes are level 1 from the first time it was established. So starting point is named "initial". But companies does not has any CMMI certificate. Company is expressed like mature means that it has CMMI certificate at least level 2. At that time development processes of company are improvement. It can be measurable so directed (Software Engineering Institute, 2010).



Name	Abbr.	ML	CL1	CL2	CL3
Configuration Management	CM	2	Target Profile 2		
Measurement and Analysis	MA	2			
Project Monitoring and Control	PMC	2			
Project Planning	PP	2			
Process and Product Quality Assurance	PPQA	2			
Requirements Management	REQM	2			
Supplier Agreement Management	SAM	2			
Decision Analysis and Resolution	DAR	3	Target Profile 3		
Integrated Project Management	IPM	3			
Organizational Process Definition	OPD	3			
Organizational Process Focus	OPF	3			
Organizational Training	OT	3			
Product Integration	PI	3			
Requirements Development	RD	3			
Risk Management	RSKM	3			
Technical Solution	TS	3			
Validation	VAL	3			
Verification	VER	3			
Organizational Process Performance	OPP	4	Target Profile 4		
Quantitative Project Management	QPM	4			
Causal Analysis and Resolution	CAR	5	Target Profile 5		
Organizational Performance Management	OPM	5			

Figure 3.4 Maturity levels with process areas (Software Engineering Institute, 2010)

3.7 Level -2 of Maturity (Managed)

As the name suggests, requirements are managed consequently processes are planned and performed in level-2 expedient. In Level 2;

- Generate the managed outputs with skilled employee who are chosen for project.
- Involve associated stakeholders that are controlled, reconsidered, monitored and evaluated.
- Maintain the existing practice during misfortune
- Manage and perform the project with their documents

- Demonstrate the management of status for work product with defined points. (Major tasks, major milestones etc.)
- Establish the commitments among associated stakeholders
- Control convenience of work product.
- Establish their standards, descriptions, processes and procedures on work products.

When the mentioned maturity values is detailed, it has 22 “Process Area (PA)” to focus on the activities of the developer organization in every CMMI models. Each process area is clustered with related practices. These process areas are shown in Table 3.3 Maturity Level 2 Process Area below.

Table 3.4 Maturity Level 2 Process Area (Software Engineering Institute, 2010)

Name	**Abbreviation**
Requirements Management	REQM
Project Planning	PP
Project Monitoring and Control	PMC
Measurement and Analysis	MA
Process and Product Quality Assurance	PPQM
Supplier Agreement Management	SAM
Configuration Management	CM

CHAPTER FOUR
SOFTWARE DEVELOPMENT LIFE CYCLE (SDLC)

Software development processes' set of activities that are significant combined to generate the software product. These are combined different ways in different set of activities according to different software processes. Besides that, every software process includes set of well- defined activities that have common definition. Thus, maturity' levels of software development are identified in stage of process and activities and functions fit into particular methodology ("SDLC Overview", 2016).

4.1 Stages of Software Development Life cycle (SDLC)

SDLC include set of general stage that shows major step, schedule and information of software development processes. Generally, each stage is built on the other and associated. Main stages are following; (Cortellessa, Vittorio, Di Marco, Antinisca, Inverardi, 2011; "SDLC Overview", 2016)

- Planning and Requirement Stage
- Definition Stage
- Analysis Stage
- Design Stage
- Development or Building Stage
- Testing or Implementation Stage
- Maintenance Stage

4.1.1 Planning and Requirement Stage

Requirement analysis is most important fundamental stage of SDLC to determine to proceed or not feasibility of project. Stage is continued by specified member of team with sales department, notice from customer and. Feasibility study is occurred with this information that is mean produced high level overview document, to survey plan project and determine to budget, schedule and technical specification ("SDLC", 2009; “SDLC Overview”, 2016). Figure 4.1 SDLC Stages shows planning and Requirement Stage on main structure.

In planning stage, quality assurance of software start to build and risks of project are defined. Thus, project is implemented with minimum risks according to outcome of technical feasibility study (“SDLC Overview”, 2016).

4.1.2 Definition Stage

Definition stage includes the when, who, and how the project expression. With the aid of expression, high level outline of project is generated. Thus Software Requirement Specification document that includes the all of product requirement to develop the product is associated with the project plan (Cortellessa, Vittorio, Di Marco, Antinisca, Inverardi, 2011; “SDLC Overview” 2016). Figure 4.1 SDLC Stages shows Definition Stage on main structure.

4.1.3 Analysis Stage

Analysis Stage is used to define to current and feature users, to execute the business processes in detail and to replicate the data on works. Various tools are used in this step like flow – charting. According to the information gathered, user’ requirements are followed and re- documented for system. Correspondingly, project plan should be updated with these changing that are budget, schedule, technical detail, data and stockholder … etc.) (“SDLC”, 2009). Figure 4.1 SDLC Stages shows Analysis Stage on main structure.

4.1.4 Design Stage

Design stage is used to build the system with the best architecture and to specific the technical requirements. Also, technical specifications are produced for minutia. Software Requirement Specification is reference to design for this architecture but sometimes, more than one design approach is suitable for product. So Design

Document Specification that is reviewed by all risk parameter, stockholders, design specification, budget, schedule and product stability, is produced for each design. But one must be chosen that is best for users and needs of business (“SDLC”, 2009; “SDLC Overview”, 2016). Figure 4.1 SDLC Stages shows Design Stage on main structure.

The purpose of system design is to create blueprint for the system that will satisfy all documented system design: (“SDLC”, 2009)

- Input
- Interface
- Processes
- Output

4.1.5 Development or Building Stage

Development is started in this stage. Code is programmed and starts to generate the product according to Design Document Specification. If required, development, unit test, integration test, screen and report and data replication are deal distributed. Also user documentation and development of user procedure is performed parallel (“SDLC”, 2009; “SDLC Overview”, 2016). Figure 4.1 SDLC Stages shows Development or Building Stage on main structure.

4.1.6 Testing or Implementation Stage

In this stage, test activities that are subset of all the staged and are involved in all stage are established. System is installed, tested and rolled out with program that is suitable training with business environment. If every feedback is ok, product is released, otherwise bug are fixed, necessary updated are coded then re-testing is maintained. Until user requirements are supported, these processes are repeated. However, carefully the project deadline, schedule and budget during fix bug and re-testing activities ("SDLC", 2009.; "SDLC Overview", 2016). Figure 4.1 SDLC Stages shows Testing or Implementation Stage on main structure.

4.1.7 Maintenance Stage

System is continuously improved according to feedbacks of users in this stage. The product is released in limited function and tested in real business environment. Then missing functions or modules are added to product with user's feedback. Figure 4.1 SDLC Stages shows Maintenance Stage on main structure.

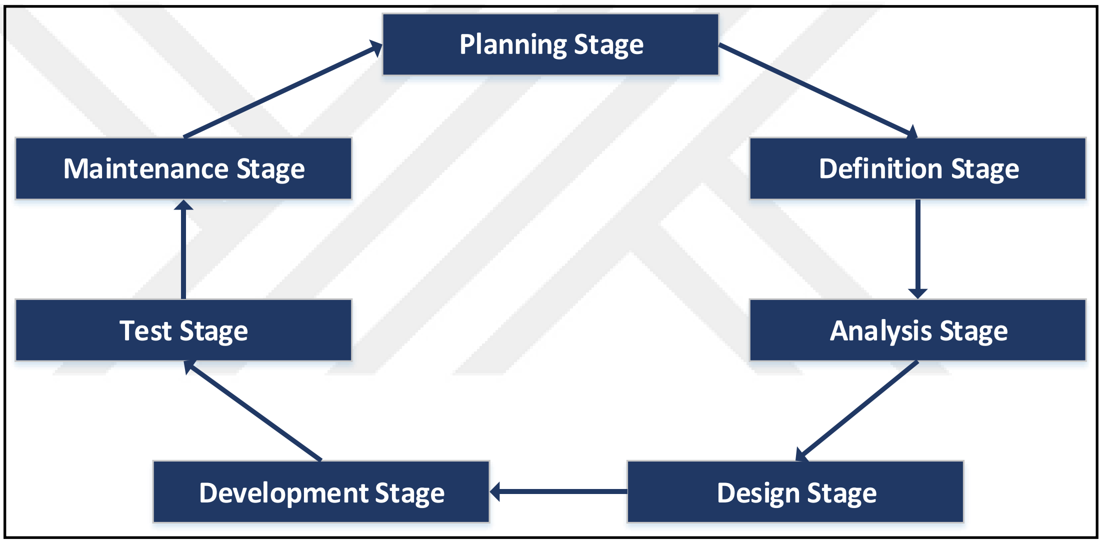


Figure 4.1 SDLC Stages

4.2 Models of SDLC

4.2.1 V – Model

The V- Model was defined by the Paul Rook in towards the end of 1980's to improve the efficiency and effectives of software development. It is like modified waterfall that concentrate between each phase of the development life cycle and its associated phase of testing. Thus phases of SDLC are verified by testing variation (Isaias & Issa, 2015).

To verification, each step is implemented with the aid of documentation of the previous step and steps are checked and confirmed with these documentations then it can move to a new step. Thus testing and development will be continued in parallel (Isaias & Issa, 2015).

V- Model includes process phases like waterfall models. The project begins with collecting requirement analysis and defining the specification then detail design is done and project is implemented with design documentation. At the same time, while system is moving down like descends down the ladder, the system move up like upwards up the ladder to verify with testing phases. It is called V model that like "V" shape (see Figure 4.2 V- Model) (Isaias & Issa, 2015; "v-model-final," 2013)

If system is detailed, implementation is tested by unit test, system design is tested by integration testing and specifications are tested by system testing and finally, all system is tested by acceptance testing (Isaias & Issa, 2015; "v-model-final," 2013).

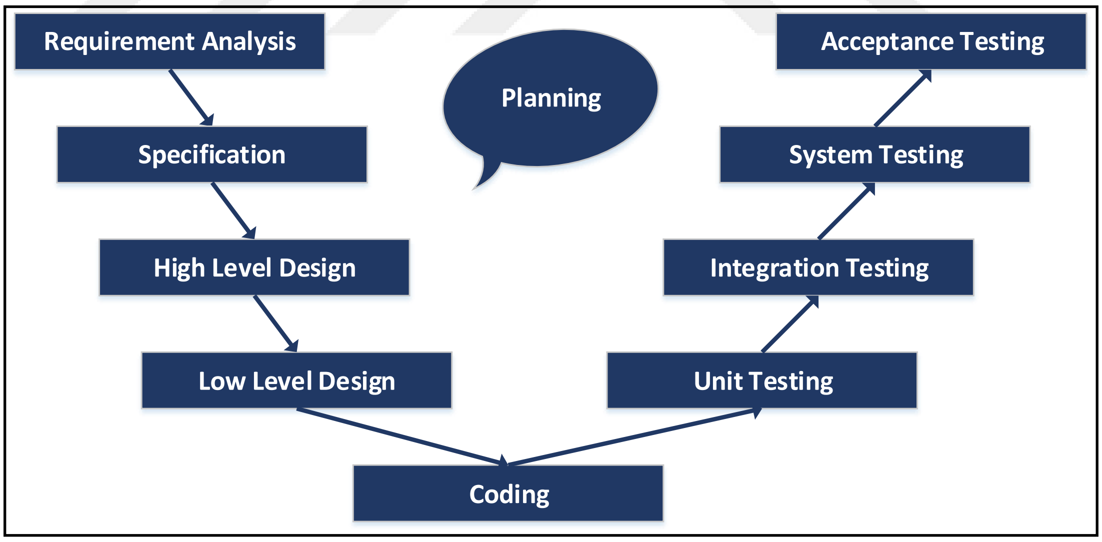


Figure 4.2 V- Model

4.2.1.1 Advantage of V-model

- Improved quality and reliability.
- Improved Risk Management.
- Reduced the amount of re-work.
- Reduced the amount of fault. Because defects don't flow down
- Offered simple and easy implementation

4.2.1.2 Disadvantage of V – Model

- Very rigid and less flexible
- Prototype is not created early. Because software is developed during the implementation step
- It is not suitable to large project.
- Required lots of resources.
- When any requirements are changed at middle of project, attached documents, that are analysis and test documents, are updated (Isaias & Issa, 2015; Munassar & Govardhan, 2010).

4.2.2 Scrum Model

The scrum is the model to solve the complex issue and provide quick adaptation, while highest valuable products that are owned from company, are development with productively and creatively (Madachy et al., 2006).

Since early 1990, processes of complex product have been managed with Scrum Model and Scrum is not technique or process of development. So Scrum is used to improve the product management and development process. Scrum model include the Scrum Teams and own roles, events, and rule. Each process serves specific purpose and is essential to Scrum's success and usage for continuity and success in

Scrum Model. Thus Scrum Model arranges interaction between processes and bind together (Madachy et al., 2006).

4.2.2.1 Theory of Scrum

Theory of Scrum uses the iterative and incremental approach. Thus risk is controlled and predictability is optimized. For this scrum uses empirical process control theory, or empiricism. Every implementation of processes is supported three items that are; (Madachy et al., 2006).

- Transparency
- Inspection
- Adaptation.

4.2.2.2 Advantage of Scrum

- Easy processes tracking
- Consistently customer feedback.
- Applied rhythm to easy understand and learn stakeholders
- Observable productiveness (with team velocity, burn down charts etc.)
- Easy organize stakeholders in different processes
- Low strict rules (documentation, procedures etc.)
- Importance for communication via face-to-face (Madachy et al., 2006)

4.2.2.3 Disadvantage of Scrum

- If there is not a definite end date, the project management stakeholders will be used to keep demanding new functionality to be delivered.
- If a task is not well defined, estimating project costs and time will not be accurate.

- If the team members are not committed, the project will either never complete or fail.
- It is suitable for small organization (fast moving projects).
- Need experienced stakeholders.
- Scrum works well when the Scrum Master trusts the team they are managing.
- If any of the team members leave during a development it can have a huge inverse effect on the project development
- If the test teams are not able to conduct failure testing after each sprint, project quality management is hard (Madachy et al., 2006).

CHAPTER FIVE
METHODS AND MATERIALS

In this chapter, materials and methods that are used in CMMI study and SDLC structure of software and affiliated departments, were explained. scope of study, population and study area, improvement process are explained and detailed with dialogs

5.1 Scope of Study

This study describes the effects of software process diversity on the CMMI level 2 process in a company.

5.2 Population and Study Area

The company being examined was located in Izmir, Turkey, with offices in Istanbul as well as Amsterdam. Software solutions in sales and distribution, logistical systems, labeling applications, and automatic defining and data collecting (AD/DC) technologies were developed by the company. The solutions were delivered to national and international customers by more than 140 employees using channels established by the solution partners. The solutions were grouped under different headings according to focus and specialization. The most important solution was Mobile Sales and Distribution Management (MSDM), used by senior management to control points of sale to increase. In particular, more than 4,000 enterprises across 12 countries relied on this software solution in their strategic decision making processes.

CMMI level 2 processes were applied to the software and affiliated departments. Software improvement processes were implemented on the primary product MSDM and its sub-product Quest. Accordingly, the study area and population of the research were:

- Product: Mobile Sales and Distribution Management, and Quest
- Location: İzmir
- Participants;
    - Software Department
        - Business developers: 15
        - Framework developers: 3
        - Bug fix developers: 3
        - Quality assurance: 1
        - Analysts: 2
    - Test Department
        - Testers: 5

5.3 Data Collection Instrument

As mentioned earlier, the improvement processes were examined in CMMI level 2. Issues in the various process areas were resolved using global and specific goals and practices within the structure of CMMI. Problems were identified using independent interviews with each stakeholder as well as the teams they are part of. These problems were then mapped to the key process areas of CMMI maturity level 2. Detailed information concerning the problems is listed in Tables 5.1 Detail for "Why are processes slow?" and 5.1 Detail for "Why do processes have faults?"

Table 5.1 Detail for "Why are processes slow?"

Issues	**Solutions**	**Process Area**	**SDLC Area**
Unaware of slowness	✓ Defining speed and speed targets, and measuring the size, time, and realized speed.	PPQA, MA	Not yet
No motivation to be fast	✓ Establishing a premium system to award achievement, training system to prevent failure, and punishment system to prevent continual failure.	PPQA, MA	Not yet

Table 5.2 continues

Poor time/resource planning	✓ Creating a capacity calculation system and establishing a size calculation system to calculate the size of needs or faults, a cost calculation system to calculate the time and resources needed to improve needs or eliminate faults, a software development and planning system to meet the requirements appropriately in optimum time based on the scope of the requirements, cost, and available capacity. ✓ Creating a dashboard for follow-up on daily and weekly proceedings.	PP, PC, PPQA, MA	Req.WI, Spec.WI
Incorrect issues affecting plans	✓ Defining and improving "urgent" issues to distinguish actually urgent issues from false alarms.	PPQA	Not yet
Lack of knowledge and experience	✓ Preventing helpers from simultaneously performing their tasks and providing help. ✓ Establishing a training system describing who requests help.	PPQA, CM	Design control, Code Inspection
Tool support	✓ Establishing a software support tool map to asses and realize compatibility among tools, support issues, and suggested solutions according to the overall aims. ✓ Obtaining support from consultants and vendors for tools.	PPQA, CM, SAM	Not yet

Table 5.3 Solution to “Why do processes have faults?”

Issues	Solution	Process Area	SDLC Area
Misunderstood needs / requests	✓ The **risks** of incomplete understanding or a misunderstanding of the needs are considered and necessary measures are taken. ✓ Business review before coding. ✓ Accepting no new needs’ analysis at least two weeks before IPC date. ✓ Making changes to validated needs’ analyses through **change request.** ✓ Writing needs/technical analyses in the form of **scenarios** using **display drawings** in needs’ analyses and preparing **test cases for technical analysis** in technical analysis.	REQM, PP, PC	Req. WI, IPC, Business Review, Finished Phase, Phase Review
Lack of knowledge and experience	✓ Commencing **Saturday** trainings to improve knowledge of the staff. Everyone attends **Saturday** trainings at least once. Taking **exams** after the trainings. ✓ Holding **ATCM** meetings and preparing a **design document** before coding. ✓ Performing **code inspection.**	PP, PC, PPQA	Business Review, ATCM, Design Control, Code Inspection

Table 5.4 continues

Poor time/resource planning / incorrect design/coding decisions	✓ Preventing interruption of work of the software development team due to the faults in the field. ✓ Allocating a **fault team** for this purpose. ✓ Working on **urgent** definitions of issues. ✓ Ensuring that all customers have migrated to the **latest version** to reduce faults in the field. ✓ Reconsidering the decision to not assign DLL in pilot and set transitions. Constitution of separate sets for the 5 major customers. Service Pack logic, etc. ✓ Improvement of environment and resource on the mobile side.	PP, PC, REQM	Developm ent
Complexity of code	✓ Manage program parameters. ✓ Keeping **online help** content updated. ✓ Implementing principles to balance the **packaged** product and **customer demands.**	REQM, PPQA, CM	Developm ent, Code Inspection
Web, business, database compatibility	✓ Checking for conformance to the layered structure, modular, and object-oriented (**LEGO**) principles. ✓ Documenting the **big picture** describing the infrastructure/framework.	PPQA, CM	Not yet

Table 5.5 continues

New technology	✓ Accelerating the **new framework** project.	PPQA	Not yet
No pre-test verification checks	✓ Software test, technical analysis test. ✓ Code inspection.	PPQA	Code Inspection, Development Test
Insufficient number of test cases and content	✓ Increasing the back office and mobile test cases. ✓ Reviewing test cases for content quality.	PPQA, CM	ISR Test
Test environment and test data issues	✓ Determining test environment issues, test environment improvement plan, and realizing the test environment improvement plan. ✓ Determining test data issues, test data improvement plan, and realizing these plans	PP, PC, PPQA, CM	ISR Test
Poor Time/resource planning / improper fault detection	✓ Focusing on test **automation**. ✓ **Planning** testing activities better. Calculating and allocating time required to conduct tests efficiently. Measuring the tested percentage (%) of the code ✓ Test training. ✓ Preparing a **test dashboard.** ✓ Establishing a **test process.**	PP, PC, MA, PPQA	ISR Test, Acceptance Test

Table 5.6 continues

Impairing another module/code while fixing one	✓ **Saturday** trainings. ✓ **Big picture** document describing the framework. ✓ Assigning **points** to each fault and performing an **impact analysis.** Performing **code inspection** and holding an **ATCM** meeting for major faults.	PPQA, CM, MA	Developm ent, Code Inspection
Excessive time for fault solutions	✓ **Saturday** trainings. ✓ Reducing mobile **build** time. ✓ Holding an **ATCM** meeting for major faults. ✓ Establishing a **fault-handling process.**	PPQA, CM, MA	ATCM
Inter-team communication problem within the software teams	✓ Allocating a special **fault team** to prevent staff from resolving faults detected in the approved **set test** to be interrupted with faults from the field. ✓ Ensuring that no **old version** is used in the field.	PPQA,	ATCM, Design Control, Code Inspection
Poor time/resource planning / solving of the detected faults	✓ Written and verbal communication **training** will be provided within the team. ✓ An **orientation program** will be scheduled.	PP, PC, PPQA, MA	ATCM, Design Control, Code Inspection

Following these operations, reports were created and measurements conducted to assess the improvement. A “back-log” system was designed to record the daily activity of each stakeholder. In this phase, a customized CMMI template of the Team

Foundation Server (TFS) system was used. Each user filled specific areas of the work item on the TFS.

5.4 Improvement Process

The company had used Scrum to solve complex issues and quickly adapt their software products to address those issues. Therefore, CMMI-DEV and Scrum were combined to enhance software productivity and creativity. Furthermore, even though the requirements changed quickly, Scrum was effective in managing them. Therefore, software products were continually released at a specified frequency and uninformed assumptions were eliminated through face-to-face communication. The following items were observed for the company:

- Sprint frequency was set to every four months.
- The number of iterations for each sprint was set to three.
- The Scrum groups and Scrum master were well organized.
- Iteration, regression, and acceptance tests were conducted on time.
- An assessment meeting was held at the conclusion of the operation of Scrum.

The complexity of software development increased within the company and issues arose with resource and scheduled management. New versions were delayed, and the number of bugs increased following a release. The company had difficulty in implementing and measuring the quality of the software project. As a solution, the company began using traditional methodologies to improve quality, reliability, and risk management. The aim was to reduce the amount of re-work and number of bugs. This was also done to ensure simple and easy implementation. However, as traditional methodologies are rigid and inflexible, requiring substantial resources and documentation, and cannot quickly adapt to changes, the company abandoned them. Thus, neither traditional nor lightweight software methodologies were suitable for the company’s software products. As a result, a system for software process diversity among software methodologies was required. When a software process diversity model using combined software methodologies was employed, the disadvantages of each were compensated for.

At the outset, the company applied a combination of CMMI level 2 and Scrum to its software department; however, it sought a better software process diversity system to replace Scrum. An investigation into this process revealed two primary questions that had to be addressed:

- Why are processes slow?
- Why do processes have faults?

The SPI was operated using CMMI-DEV and Scrum for a year. During this time, the following observations were made:

- Requirements' analysis is unsuitable for quickly moving processes except in the cases of urgently needed changes for bugs reported by customers.
- Product versions across customers were not the same. A new version was released every four months and some customers preferred not to use them. For such customers, the number of bugs was higher.
- Each Scrum needed experienced team members, but software teams in the company lacked the required number of expert developers.
- Risk was not managed appropriately during the iteration process within the Scrum,
- There was insufficient documentation

Based on these observations, Scrum was converted into a software process diversity method that combines Scrum and the V-Model. The development process was thus extended, the release period between versions was more appropriate, and the requirements' analysis was fully documented. To achieve these, the development process was divided into six parts and each part was defined according to the methodology of the V-Model. Thus, a software process diversity model was implemented by combining the V-Model with Scrum. The new configuration to address the abovementioned issues is shown in Figures 5.1 and 5.2.

In particular, Figure 5.2 shows the software process diversity model used during the software processes, while Figure 5.1 shows the V-model used during software development. The processes and procedures of this model are described below.

5.4.1 Requirement work item (Req. WI)

The primary objective in this process is based on the collection and management of customer needs. For this purpose, project managers collect customer requests from the customer services department, entered as work items on the tool used by the company. The product manager then examines these work items. If he/she approves a needs' analysis, the work item is directed to be evaluated by the software. Otherwise, the project manager returns to rectify the item. Once the relevant work item reaches the software, the required person days are entered by experienced software developers and administrators. In this manner, the needs' analyses with their number of person days are determined and collected. However, the person days spent on the processes in the V-Model software process were subsequently added to the person days' calculation of the relevant analysis. Moreover, the sprints to which the areas of the analyses were allocated were determined and categorized before the IPC (Integrated Plan Committee) process. These areas were handled by the back office and mobile areas.

5.4.2 Integrated plan committee (IPC meeting)

This process is based on the establishment and transmission of the project plan. The sprints and iteration for the collected needs' analyses are determined by software, quality assurance, and product and project managers in the IPC meeting. Start and end dates are determined by establishing a project plan based on the total number of person days, urgent customer needs person days, and holidays and leave days for the selected needs' analysis. This content is transmitted to the stakeholders of each department by the managers attending the meeting.

5.4.3 Specification work item (Spec WI)

The Spec WI process is based on writing the technical analysis to render the needs' analyses in the plan into a format that can be understood by software developers. A technical analysis is not written for every needs' analysis within the scope of the plan; this determination is made by software and product managers. These analyses are conducted by business analysts who obtain approval from the product manager or project manager of the relevant analysis. If the technical analysis is approved, it is associated with the relevant work item in the system. Otherwise, business analysts perform a reconfiguration for the analyses returned. However, the Cross Check and Business Preview processes of the V-Model were configured in this process.

5.4.4 Development

The development process was reconfigured with processes of the V-Model. As shown in Figure 5.1, this process divided the development process of the Scrum model into parts, and restructured it into a control and verification mechanism. There were three basic procedures catering to the following: what to do before coding, what to do during coding, and what to do after coding. Before coding, the Analyst, Tester, Coder meeting was conducted, and the Design and Design Control processes from the V-model were included. During coding, the Buddy Check and Preview Coding sub-processes were added, as explained in the V-model. Finally, after coding, Developer and Test and Code Inspection were added to build the control and verification mechanisms.

5.4.4.1 Business review

This process consists of a review of the technical documents by project managers, which are drafted by business analysts and project managers. In this process, the analysis is approved by the project manager in prototype or text format.

5.4.4.2 Analyst, tester, coder meeting (ATCM)

The primary objective of this process is to determine a road map with the stakeholders from every department before commencing coding analysis. As in the title, this meeting involves an analyst, a tester, and a coder, and is planned by Scrum masters.

5.4.4.3 Design and control

The primary objective of this process is to combine the technical analysis algorithm with the software by utilizing the relevant technical data and diagrams before commencing coding. Thus, software developers design the relevant technical document and obtain the approval of an experienced software developer. There is no software-related or other rule that must be observed in this process.

5.4.4.4 Coding

Coding is the fault, fault-free development of the product by the software developer. Hence, software developers cooperate with experienced software developers and business analysts. This cooperation is achieved as follows:

- Preview: For two days each week, business analysts check for the compliance of the analyses they are responsible for against the business criteria during development.
- Buddy Check: The technical accuracy of the code flow is checked twice daily by experienced software developers.

5.4.4.5 Developer testing

This step involves checking during unit testing before proceeding with the test stage. Hence, software developers cooperate with either testers or business analysts.

5.4.4.6 Code inspection

The objective of this process is to check for the compliance of the code with coding standards as well as ensuring the technical accuracy of the code by experienced software developers. Although this process appears similar to Buddy Check, there is a significant difference. A critical error detected at this point requires that the software developer redesign and reconfigure the code from scratch.

5.4.5 Finished phase

The finished phase is based on an estimation of the finished phase date. Scrum teams and the relevant masters determine this date.

5.4.6 Phase review

The assessment of the finished phase and risk management for deficiency, if any, constitutes this phase. Finished phases are assessed by Scrum teams and masters, testers, and product managers.

5.4.7 Integration, system and regression test (ISR Test)

Test processes are managed and controlled in the ISR test process. Testing starts from the smallest area. In this workplace, the testing department performs its smallest test using the integration test. Large areas are then tested with the system and regression tests. The integration test is performed until the end of the relevant phase as analysis ends, following which testers test the analyses assigned to them. An error found here is forwarded to the corresponding software developer who developed the code. Following the integration test phase, regression tests begin. Regression testing is also performed at the end of a relevant phase. The integrity of the analyses performed with respect to the other analyses of the module containing the product is tested. Any error occurring here can be resolved by any software developer. Following this part, the final type of system test, i.e., product test, is performed. Again, any software developer can resolve any mistake at this stage. Once this three-part test is complete, acceptance tests are performed at the end of the sprint. These tests are conducted in a separate process. The testing department follows a certain method for them. The test cases created earlier are first determined followed by the new test cases. The cases are prioritized and checked by the test manager. The results are reported by a test specialist (test case report). Based on this report, the outcome criteria are decided. The lifecycle of the test for the relevant analysis is then modified. The resulting document is shared with the test department. Accordingly, test environments are prepared. Following this process, analyses are assigned in a work distribution between test managers and testers. Each tester tests his/her own analysis. Completed test cases are overwritten in the system. If there are no errors, the analysis is complete. However, this process is conducted based on four new rules:

- executing the processes capable of being resumed in the test case by automatic testing tools;
- inspecting and configuring insufficient test cases at the end of each sprint, and adding missing test cases, if any;
- documenting the test cases and results;
- If there is an incorrect analysis during testing, this fault must return to the first process called the “requirement work item process.”

5.4.8 Acceptance test

This process is implemented in the final phase by the testing department. In this process, the system is tested in its entirety. Software developers primarily address any fault at this stage. Following this, the product is ready to be released on the market.

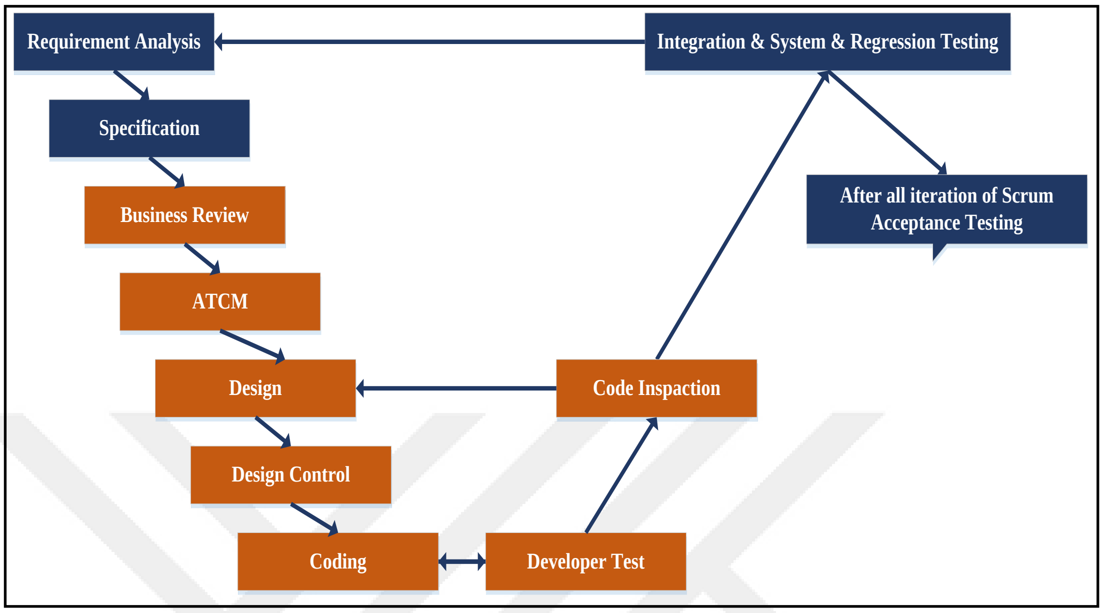


Figure 5.1 V-model of selected company according to study of SDLC

In summary, the CMMI research and results, and research on the SDLC and the transition from the old system to the new one are described in this chapter. The following chapter describes the solutions based on the above discussion, their implementation using the modified SDLC, and the advantages of the implemented SDLC.

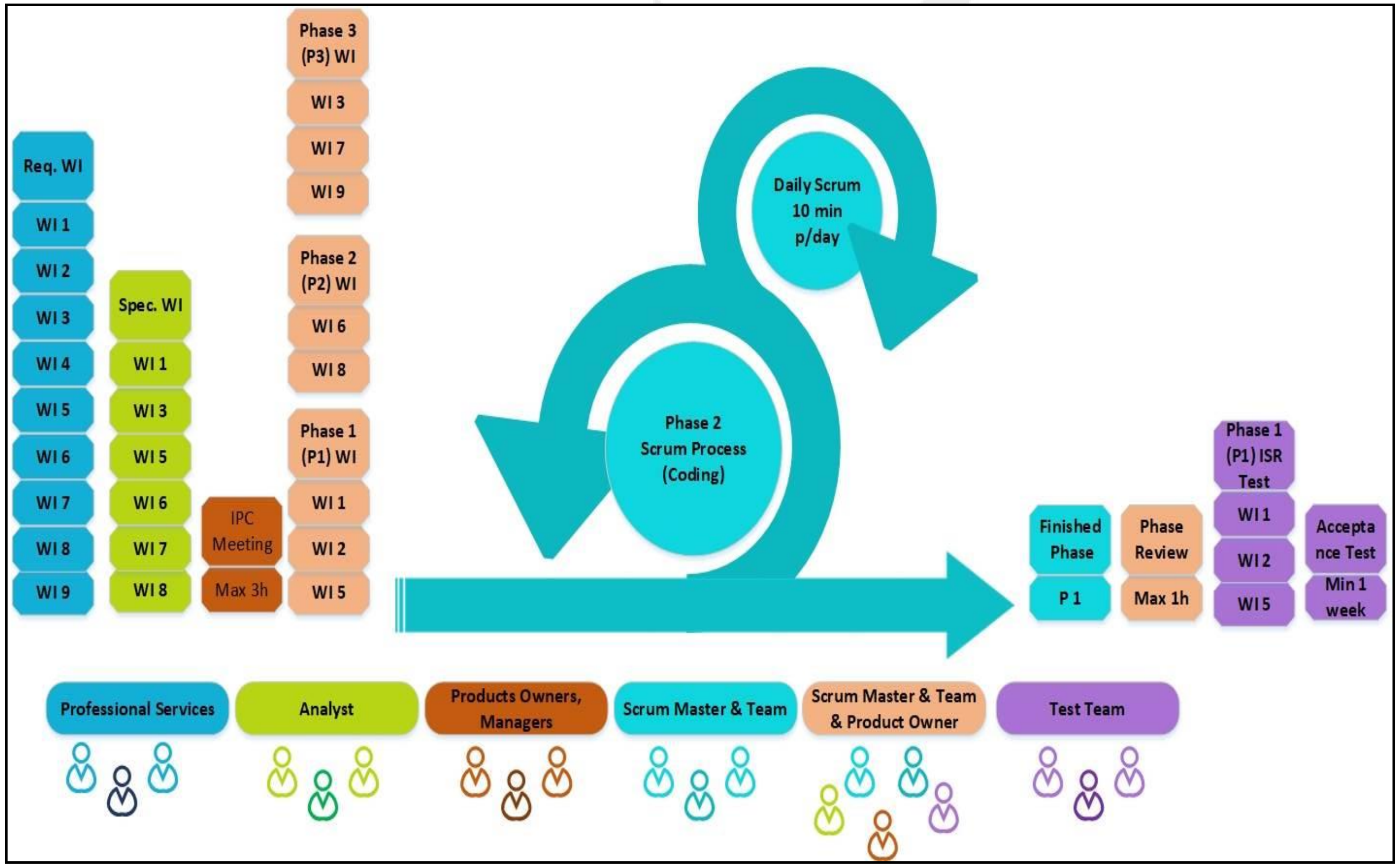


Figure 5.2 Hybrid model of selected company according to study of SDLC

CHAPTER SIX
RESULTS AND DISCUSSION

In this chapter, the solutions obtained from the CMMI processes, relation among the solutions, and SDLC processes are evaluated. In particular, the version plans used by the Scrum and software process diversity models as well as the results of these version plans, and comments obtained by addressing the two primary issues examined using CMMI are considered. Each issue is detailed and answered. Each solution is mapped to the key area of the related CMMI process (see Table 3.3 in Chapter Three).

The actions to be taken for the two issues were agreed upon by all stakeholders from the software department (chief technology officer, quality assurance department, software manager, software leader, team leaders, and all developers) as well as the test department. At this stage,

- the company was committed to CMMI practices and these were explained in detail to its employees;
- the necessary CMMI process training was provided to follow-up on the improvements;
- by adopting human-focused processes to manage this change and build CMMI process awareness, employees were given the necessary motivation (like rewards); and
- experienced stakeholders were assigned to improve SDLC, find CMMI solutions, and report the benefits.

In addition to the above, changes to the software process were accepted by the relevant departments, but the improvement processes were not fully executed on all products of the company. The firm evaluated CMMI over one year. Following the assessment, it was determined that the stakeholders working on different products should apply flexibility and diversity in the improvement processes of products with few dependencies. Because of the use of key areas of the CMMI process, the application time might have increased. For this reason, as company policy, improvements were made to two software products with the same business processes and structures. The other software products were excluded from the process until CMMI migration problems were overcome. Even though the improvement processes were not homogeneously distributed on all products, resource assignment was used to do so for each selected project. As listed in Table 6.1, a total of 16 stakeholders (two stakeholders were fully integrated and 14 stakeholders shared integrated) were assigned to the product "Quest." A total of 27 stakeholders (13 stakeholders were fully integrated and 14 shared integrated) were assigned to the product “MSDM”.

Table 6.1 Organizational flexibility

Organizations	**No. of stakeholders**	
	Product 1: MSDM	**Product 2: Quest**
Web developer	6	1
Mobile developer	7	1
Framework developer	3	
Bug-fix developer	3	
Quality assurance	1	
Analyst	2	
Tester	5	

As listed in Table 6.2, 13 of the 20 sub-problems were originally selected for CMMI improved with SDLC processes (see Tables 5.1 and 5.2 in the Chapter Five for issues and solutions). The 65% improvement rate is relatively high. When the SDLC processes were used in the process improvement stages, half the processes were improved using the appropriate method. In particular, the organizations were very effective in reducing the number of faults and solving the speed-related problem. Given the records of the issues of speed and faults, a 78.5% improvement was obtained in reducing faults and 33.3% in increasing speed. Therefore, reducing the number of faults and speeding the processes up were the basic criteria in this context.

Table 6.2 Percentages of the issues resolved with SDLC in total CMMI issues

	Issue Counts	**Implementation of Solution Counts**	**%**
Slowness	6	2	%33.3
Fault Count	14	11	%78.5
Total	20	13	%65

In A numerical representation of the methods shows that it is possible to measure the fundamental basic processes in CMMI level 2. A conclusive summary is presented in Table 6.3.

Table 6.3 Version plans and their evaluation

	software versions 1	**software versions 2**	**software versions 3**
Total number of days	110 days	103 days	152
Number of days' delay	53 days	14 days	3 days
Planned person days	1013	678.17	1400
Realized person days	1398	678.17	1249.73
Daily average closed person days		6.04	10.5
Number of analysis test faults		325	371
Number of approved set faults		711	654
Approved set workdays		45	19
Number of approved set faults per day		15.80	34.42

In particular, Table 6.3 shows the number of faults at the end of the project and improvement in speed on the basis of version for the company. The model used in software version 1 was Scrum, and software process diversity models were used in software versions 2 and 3.

6.1 Speed Increase Based Results

Inferences;

- In software version 1, 1398 person days of work were performed for a sprint of 110 days, and with 1013 planned person days.
- In software version 2, 678.17 person days of work were performed for a sprint of 103 days in the software process diversity model.
- In software version 3, 1249.73 person days of work were performed for a sprint of 152 days in a software process diversity model, with 1400 planned person days.

These observations show that the numbers of person days used for software version 1, the Scrum model, and the two software process diversity models were quite close; however, the work was completed in fewer days in software version 2. The processes emerged in addition to production because the V-Model is a component of the software process diversity model based on a verification and control mechanism (Business Review, ATCM, Design and Design Control, Development Test and Code Inspection). If higher production is required in software version 3 by verifying this process, its business plan takes longer. Consequently, software version 3 took two months longer than version 2; nevertheless, software version 3 involved higher production.

Results;

- The release date was delayed by 53 days owing to the complexity of and excess work in software version 1, planned using the Scrum model.
- Using the software process diversity model, the set was released with a delay of 14 days owing to insufficient improvement in the model in software version 2. In this version, the daily average number of closed person days was 6.04 when measurements started in this version.

- Using the software process diversity model, the process proceeded smoothly and the set was released with a negligible delay in software version 3. Thus, this version was an improved one, and the average number of person days almost doubled to 10.5.

6.2 Number of Faults

Inferences;

- In software version 1, real numbers could not be obtained as the records could not be transferred to the modified system appropriately.
- In software version 2, testers detected 325 faults when testing 678.17 person days of work, and they were resolved by software developers until the approved testing period. The testers detected 711 faults in the approved testing period, and it took 45 days to resolve the process and its faults.
- In software version 3, the testers detected 371 faults when testing 1249.73 person days of work, and they were resolved by software developers until the approved testing period. The testers detected 654 faults in the approved testing period, and it took 19 days to resolve the process and its faults.

These observations indicate that the number of faults detected in the test stage continued to decrease and were resolved in a shorter period as the version developed. At first glance, although it seems that fewer faults were detected in software version 2 compared with version 3, when ratio of person days spent is examined in the relevant version, there were fewer faults in the product development process in software version 3. If 325 analysis tests occurred in the daily plan of 678.17 person days in software version 2, they might have been expected to detect a maximum of 598.90 faults in testing 1249.73 person days. However, only 371 faults were detected in software version 3. Similarly, if 711 faults were detected in 678.17 person days in the approved test set, they might have been expected to detect a maximum of 1310 faults in testing 1249.73 person days. However, only 654 faults were detected in the approved test of software version 3. Moreover, 45 days were spent on 711 faults in the approved set test of software version 2, whereas only 19 days were spent on 654 faults in for software version 3.

Results;

- Using the software process diversity model, the approved test period was longer and more faults were resolved in software version 2 than software version 3.
- Using the software process diversity model, the process proceeded smoothly in software version 3, and the approved test period was processed with fewer faults and in a shorter period of time.

In summary, based on our observations, the expected results were obtained by the software department along with the estimated development process. Consequently, improvements were made in every version to achieve the desired speed and improvement, considering the change in the number of the faults resolved in parallel with the proposed process.

CHAPTER SEVEN
CONCLUSION AND FUTURE WORK

Both cost and time are difficult to calculate when a single process was performed uniformly on products that have an identical technical framework and similar business structure. To address this problem, continual improvement processes are required that are inherent to the software. In this study, the applicability of the CMMI model to SDLC was verified by showing that a company's Scrum model was inadequate. As CMMI and SDLC are complementary, they are more appropriate to assess results. Considering the impact on CMMI, because the company evaluated in our study used CMMI level 2, only some questions concerning how SDMI processes and CMMI should be implemented were addressed.

For this reason, detailing SDLC processes used in the Scrum model appeared unsuitable for products requiring verification, control, and documentation. The Scrum model is tailored to the software department's advantage. The software diversity process and flexibility were used for improvement processes on several products with varying organizational structures. As a result, stable and more feasible structures were established.

According to the results obtained during our study of the software development process of the company, these processes, implemented within a year, were successful in terms of increasing speed and reducing the number of faults; however, they did not achieve a successful outcome in the case of productivity.

However, the solutions applied in our study involve a routine model, which does not solve the problems experienced by several companies. In particular, administrative problems such as time constraints, lack of resources, and limitation of resource usage can cause the process to fail in the long term. Ensuring effective change in a live system requires effort and determination. It is thus considerably important to reuse experience and make marked improvements in measurements.

As future work, two issues will be addressed: specifying how the SDLC models can implement CMMI, and whether it is possible to obtain the desired software processes by implementing the CMMI model in the cases where the existing SDLC models are inadequate. Thus, future research will focus on detailing software process diversity models to mitigate the disadvantages of models that are separately configured and widely used in recent times. Another area of focus may be the advantages of software process diversity models in the cases where they are used together with other SPI types than CMMI, when compared with other single models. CMMI level 3 and higher might be investigated with respect to improving project management in terms of person and risk management, quality management using process and software metrics, early recognition of deviations in budget and time targets through continual improvement in measures, and necessary improvements and continual improvement of function policies in the process.

REFERENCES

Aaen, I., Arent, J., Mathiassen, L., & Ngwenyama, O. (2001). A conceptual map of software process improvement. *Scandinavian Journal of Information Systems*. Retrieved February 18, 2017 from http://aisel.aisnet.org/sjis/vol13/iss1/8

Adler, P. (2013). The Collaborative , ambidextrous enterprise. *Universia Business Review, 40*, 34-51.

Agrawal, M., & Chari, K. (2007). Software effort, quality, and cycle time: A study of CMM level 5 projects. *IEEE Transactions on Software Engineering*, *33*(3), 145-156. https://doi.org/10.1109/TSE.2007.29

Balaji, S. (2012). Waterfall vs V-model vs Agile : A comparative study on SDLC. *International Journal of Information Technology and Business Management*, *2*(1), 26-30.

Batra, D., Xia, W., van der Meer, D., & Dutta, K. (2010). Balancing agile and structured development approaches to successfully manage large distributed software projects: A case study from the cruise line industry. *Communications of the Association for Information Systems, 27*(1), 379-394.

Boehm, B., & Turner, R. (2005). Management Challenges to implementing agile processes in traditional development organizations. *IEEE Software*, *22*(5), 30-39.

Cho, J. (2009). A Hybrid software development method for large-scale projects: Rational unified process with Scrum. *Issues in Information Systems*, *10*(2), 340-348.

Chrissis, M. B., Konrad, M., & Shrum, S. (2003). *CMMI: Guidelines for process integration and product improvement* (2nd ed.). Boston MA: Addison-Wesley

Cortellessa, Vittorio, Di Marco, Antinisca, Inverardi, P. (2011). *Model-based software performance analysis*. Heidelberg: Springer.

Curtis, B. (2000). Global pursuit of process maturity. *IEEE Software*, *17*(4), 76-78.

Ganpatrao Sabale, R., & Dani, A. (2012). Comparative study of prototype model for software engineering with system development life cycle. *IOSR Journal of Engineering*, *2*(7), 2250-3021.

Garzás, J., & Paulk, M. C. (2013). A case study of software process improvement with CMMI-DEV and Scrum in Spanish companies. *Journal of Software: Evolution and Process*, *25*(12), 1325-1333.

Glazer, H., Dalton, J., Anderson, D., Konrad, M., & Shrum, S. (2008). CMMI or Agile: Why not embrace both !. *SEI in Software Engineering*. Retrieved November 11, 2008 from https://resources.sei.cmu.edu/library/asset-view.cfm?assetid=8533

Harris, M. L., Collins, R. W., & Hevner, A. R. (2009). Control of flexible software development under uncertainty. *Information Systems Research*, *20*(3), 400-419.

Heckscher, C. C., Adler, P. S., & Paul, A. (2008). *The Firm as Collaborative Community: Reconstructing Trust in the Knowledge Economy*. Oxford: Oxford University Press.

Hneif, M., & Ow, S. H. (2009). Review of agile methodologies in software development. *International Journal of Research and Reviews in Applied Sciences, 1*(1), 2076-734.

Isaias, P., & Issa, T. (2015). *High level models and methodologies for information systems*. New York: Springer.

J., S., R.J., K., J.J.M., T., A.J.M.M., W., Samalikova, J., Kusters, R. J., … Weijters, A. J. M. M. (2014). Process mining support for Capability Maturity Model Integration-based software process assessment, in principle and in practice. *Journal of Software: Evolution and Process*, *26*(7), 714-728.

Kuhrmann, M. (2015). Crafting a software process improvement approach - a retrospective systematization. *Journal of Software: Evolution and Process*, *27*, 114-145.

Lindvall, M., & Rus, I. (2000). Process diversity in software development. *IEEE Software*, *17*(4), 14-18.

Madachy, R., Boehm, B., & Lane, J. A. (2006). Spiral lifecycle increment modeling for new hybrid processes. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics),* (167-177).

Mathai, M.K., Venugopal, R., Abraham, J. . (2016). Hybrid model for software development. *International Journal of Research in Engineering and Technology, 5,* 198-202.

Mohammed, N., Munassar, A., & Govardhan, A. (2010). A comparison between five models of software engineering. *International Journal of Computer Science*, *7*(5), 94-101.

Munassar, N. M. A., & Govardhan, A. (2010). *Hybrid model for software development processes*. Retrieved November 13 2017 from http://itpapers.info/acit10/Papers/f256.pdf

Niazi, M. (2015). A comparative study of software process improvement implementation success factors. *Journal of Software: Evolution and Process, 27*(9), 700-722.

O’Reilly, C. A., & Tushman, M. L. (2011). Organizational ambidexterity in action: How managers explore and exploit. *California Management Review, 53*(4), 5-22.

Paulk, M. C. (2001). Extreme programming from a CMM perspective. *IEEE Software, 18*(6), 19-26.

Pawar, R. P. (2015). A Comparative study of agile software development methodology and traditional waterfall model. *IOSR Journal of Computer Engineering,* (1-8).

Preeti, R., & Saru, D. (2014). Impact of different methodologies in software development process. *International Journal of Computer Science and Information Technologies*, *5*(2), 1112-1116.

Ramasubbu, N., & Balan, R. K. (2009). The impact of process choice in high maturity environments: An empirical analysis. *In Proceedings - International Conference on Software Engineering* (529-539).

Ramasubbu, N., Bharadwaj, A., & Kumar Tayi, G. (2015). Software process diversity: Conceptualization, measurement, and analysis of impact on project performance. *MIS Quarterly, 39*(4), 787-807.

Ramesh, B., Mohan, K., & Cao, L. (2012). Ambidexterity in agile distributed development: An empirical investigation. *Information Systems Research*, *23*(2), 323-339.

SDLC (2009). Retrieved September 7, 2016 from http://www.slideshare.net/orlandomoreno/sdlc-system-development-life-cycle-sdlc

SDLC Overview (n.d.). Retrieved September 7, 2016 from http://www.tutorialspoint.com/sdlc/sdlc_overview.htm

Software Engineering Institute (2010). CMMI for development, version 1.3. *Carnegie Mellon University*. Retrieved November 1, 2010 from https://resources.sei.cmu.edu/asset_files/TechnicalReport/2010_005_001_15287.pdf

Stephen, O., & Oriaku, K. (2014). Software development methodologies: Agile model vs V-model. *International Journal of Engineering and Technical Research (IJETR)*, *2*(11), 108-113.

Stoica, M., Mircea, M., & Ghilic-Micu, B. (2013). Software development: Agile vs. traditional. *Informatica Economică*, *17*(4), 64-76.

Subramanyam, R., Ramasubbu, N., & Krishnan, M. S. (2012). In search of efficient flexibility: Effects of software component granularity on development effort, defects, and customization effort. *Information Systems Research*, *23*(3), 787–803.

Sutherland, J., Jakobsen, C. R., & Johnson, K. (2007). Scrum and CMMI level 5: The magic potion for code warriors. *In Proceedings - AGILE 2007* (272–277).

V-model-final. (2013). Retrieved September 7, 2016 from http://www.slideshare.net/suhasreddy1/v-model-final

Vinekar, V., Slinkman, C. W., & Nerur, S. (2006). Can agile and traditional systems development approaches coexist? An ambidextrous view. *Information Systems Management*, *23*(3), 31-42.