Capability Maturity Model of Software Engineering Design Process (SDP^{CMMI})

Abdullah Al-Shishani
Faculty of Information Technology
Department of Software Engineering
Hashemite University
Zarqa, Jordan
abdullah.asendarz@gmail.com

Khalid T. Al-Sarayreh
Faculty of Information Technology
Department of Software Engineering
Hashemite University
Zarqa, Jordan
khalidt@hu.edu.jo

***Abstract*—Software design is a very important phase of the software development life cycle, It includes problem solving and planning a software solution. Which includes both low and high component design, such as algorithm and architecture design. If software design is done correctly, productivity and quality of the system can be improved on the later stages of the software development life cycle reducing the cost and time needed in the implementation phase.**

This paper presents a new process model to standardize software engineering design process, based on the on the Capability Maturity Model Integration for Development. The intention is to provide a generic maturity model for software engineering design process with quality standards to insure productivity and quality of the project under development.

Keywords*—*Software Design, software engineering, capability maturity model, capability maturity model for development.

I. Introduction

Software design is the process by which an agent creates a specification of a software artifact, intended to accomplish specific goals, using a set of primitive components and subject to constraints [15]. As defined by Freeman et al. [7] it is *the activity following requirements specification and before programming*, which defines how the software is going to be implemented. It is the process of finding an optimal or near-optimal solution of a problem which is defined in the requirements engineering phase. However, finding a solution of a problem is not an easy task, Curtis et al. [4] said that **"Writing code is not the problem, understanding the problem is the problem"**. Meaning that if a poor design is implemented, it may affect the later stages of software development life cycle. A field study of the software design process done by Curits et al. [4], showed that a bad software design can lead to productivity and quality problems, recorded by Weinberg [19], Brooks [3] and Fox [6], these problems have survived for several decades in spite of serious effort at improving software productivity and quality. Thus, it is really important to design the software with a controlled process and based on quality standards that can insure the quality of the design, hence, the quality of the software.

Software design process has many principles, which represents some kind of goals to achieve whilst implementing the design model of the software. Meyer et al.[10] defined some principles for object-oriented software. Some of these principles are: *Open Close Principle, Dependency Inversion Principle, Interface Segregation Principle, Single Responsibility Principle and Liskov's Substitution Principle*. However, these principles are dedicated to object-oriented software, thus, more general and generic principles are needed. Davis et al.[5] suggested more general principles:

- The design should be traceable to the analysis model.
- The design should minimize the distance between the software and the problem as it exists in the real world.
- The design should be structured to accommodate change.
- The design should be assessed for quality as it is being created, not after.
- The design should be reviewed to minimize conceptual (semantic) errors.

these principles should be taken into consideration while constructing the design of the software. However, under certain conditions these principles can be dropped, based on Curtis et al. [4] field study, a typical statement heard from the participants was that, *you've got to understand, this isn't the way we develop software here*. Such comment shows that developers where affected by conditions surrounding their project, and may not consider such principles under certain conditions. Thus, the need for a controlled process and a general model can not be ignored.

Najjar and Al-Sarayreh [11] proposed a generic model to insure the quality of requirements engineering, the model is called Capability Maturity Model of Software Requirements Process and Integration (SRP^{CMMI}). It is based on CMM integration for development (CMMI-DEV) [16] and consists of three maturity levels.

Inspired from SRP^{CMMI} and motivated by the need of a process model to control the quality of the design phase of software development life cycle, this paper proposes **Capability Maturity Model of Software Engineering Design Process (SDP^{CMMI})** based on CMMI-DEV. The model's goal is to insure the quality of the software design process, thus, to improve the quality of the software.

CMMI is the base because it is widely used [11] and it could be certified as compliant with IS0 9001 [12]. Moreover, there are over 5,000 businesses that use CMMI models from over 70

countries, and using it showed slightly improvement in many fields. Table I summarizes the quantitative performance results of Gibson et al. [8] in terms of percentage changes [1].

The remainder of this paper is organised as follows: Section 2 describes related work; section 3 shows an overview of the CMMI and CMMI-DEV; section 4 shows an overview of the Software Engineering Design Process; section 5 describes the proposed model; section 6 describes conclusion and future work.

II. Related Work

Several CMMI based models were used as the base for the software development process. Pulk et al.[14] discussed how to use CMMI in any business environment. The research showed that the issues associated with interpreting the Software CMMI for the small project or organization may be different in degree, but they are not different in kind, from those for any organization interested in improving its software processes. However, they used CMMI for the development process in general and did not focus on the design process.

Pulk et al.[13] also discussed extreme programming from a CMMI perspective. They summarized both extreme programming and the SW-CMMI to show how extreme programming can help organizations realize the SW-CMMI goals.

Lee et al. [9] applied CMMI, Total Quality Management (TQM) and ISO 9001 in Knowledge Management for software development process improvement. They say that traditional software development process emphasises testing techniques, but show weakness in planning, and does not satisfy the requirement of the user to cause failure cost heavily. But again, they did not focus on the design phase.

Najjar et al.[11] proposed Capability Maturity Model of Software Requirements Process and Integration *(SRP^{CMMI})*, the main goal of the model was to help in improving the requirement engineering process. This paper is inspired by their work, to apply CCMI on the software engineering design process.

Based on a research we made, no one applied CMMI in software engineering design process to improve the quality of the software.

III. Introduction To Capability Maturity Model For Development

CMMI (Capability Maturity Model Integration) models are collections of best practices that help organizations to improve their work. Inspired by Watts S. Humphrey, Software Engineering Institute (SEI) developed Capability Maturity Model Integration for Development (CMMI-DEV). CMMI-DEV is meant to help software building grow up from being ad-hock to professional engineering discipline. CMMI-DEV is not a guideline, set of process or procedures to follow, mandate for improvement or a particular way of doing work. It is a reference model and a structured description of proven practices that if implemented appropriately will improve the work of the organization, it describes what to do not how to it. However, it is incomplete, there are some areas it doesn't cover, such as disaster recovery and marketing. CMMI-DEV is one of the most commonly used maturity frameworks [11] by organizations from many industries including aerospace, banking, computer hardware, software, defense, automobile manufacturing, and telecommunications[16].

CMMI-DEV is divided into categories of practises, these categories of practices are called process areas (PAs), *a group of related practices that satisfies a set of goals that are important to make improvement in that area*. PAs can be seen as labels that serves organizers with a set of practices within the category that have been shown effective when implemented appropriately.

CMMI-DEV defines 22 PAs[16], and to support continuous representation, process areas are categorized into: *Process Management, Project Management, Engineering, and Support*. PA defines how to look at the work, the work is the same, but how we look at it can provide different perspectives on how to make work more effective of efficient. Some of the PAs that CMMI-DEV defines are: *requirements development and requirements management*, which deal with elicitation, development and validation of requirements as well as the changes occur during the project, and *technical solution and product integration* , which deal with the design, development along with insuring that all the parts of the product fit together when delivered to the customer.

how a single PA is structured

All PAs are clusters of practices organized by goals that summarizes the outcome of practise implementation. Practices are categorized into:

- Specific Practice: practices with specific to the doing of the work *(ex. estimating a project and requirements elicitation)*, these practices are descriptive not prescriptive, which means they describe what the project will do not how to do it. Specific practices are linked with specific goals.
- Generic Practice: practices that sustain project over time *(ex. any work activity should have a plan, appropriate resources and knowledge needed to do the work)* they are called generic because they are generic across the model. Generic practices are linked with generic goals.

There are some components found in each PA of CMMI-DEV, figure 1 shows these components and the relationship between them. These components are grouped into three categories:

- Required: essential components to achieving process improvement in a given process area.
- Expected: describes the activities that are important in achieving a required CMMI component.
- Informative: helps model users understand CMMI required and expected components.

The CMMI-DEV is organized into two types of levels *capability levels and maturity levels*. To reach a particular level, an organization must satisfy all of the goals of the process area or set of process areas that are targeted for improvement, regardless of whether it is a capability or a maturity level.

CMMI-DEV defines many processes that can be applied to software design phase, such as *Decision Analysis and Resolution (DAR), Validation (VAL) and Verification (VER)*.

TABLE I. QUANTITATIVE PERFORMANCE RESULTS OF GIBSON IN TERMS OF PERCENTAGE CHANGES

Performance Category	Median Improvement	Number of Data Points	Lowest Improvement	Highest Improvement
Cost	34%	29	3%	87%
Schedule	50%	22	2%	95%
Productivity	61%	20	11%	329%
Quality	48%	34	2%	132%
Customer Satisfaction	14%	7	-4%	55%
Return on Investment	4.0 : 1	22	1.7 : 1	27.7 : 1

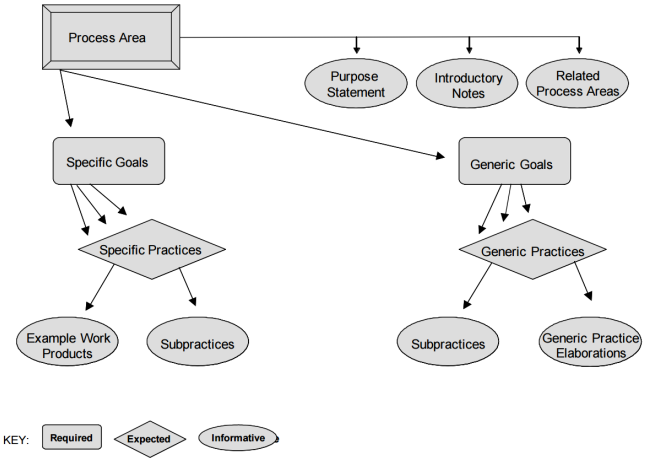

Fig. 1. CMMI Model Components

DAR, a PA at Maturity Level 3, is meant to analyze candidate solutions using a formal evaluation process that evaluates identified alternative solutions to the problem, this can be applied to software design process by analyzing and evaluating alternative design decisions made using formal evaluation process. DAR has one specific goal, that is *Evaluate Alternatives*, that has the following specific practices:

SG 1 Evaluate Alternatives
SP 1.1 Establish Guidelines for Decision Analysis
SP 1.2 Establish Evaluation Criteria
SP 1.3 Identify Alternative Solutions
SP 1.4 Select Evaluation Methods
SP 1.5 Evaluate Alternative Solutions
SP 1.6 Select Solutions

VAL, a PA at Maturity Level 3, is meant to demonstrate that components of the product fulfills its intended use when placed in its intended environment, this can be applied to software design process by demonstrating that design decisions made fulfills their intent. VAL has two specific goals, which are *Validate Product or Product Components and Prepare for Validation*, that have the following specific practices:

SG 1 Prepare for Validation
SP 1.1 Select Products for Validation
SP 1.2 Establish the Validation Environment
SP 1.3 Establish Validation Procedures and Criteria
SG 2 Validate Product or Product Components
SP 2.1 Perform Validation
SP 2.2 Analyze Validation Results

VER, a PA at Maturity Level 3, is meant to ensure that selected work products meet their specified requirements, similarly, ensuring that design decisions made meet their specified requirements. VER has three specific goals, which are *Prepare for Verification, Perform Peer Reviews and Verify Selected Work Products*, that have the following specific practices:

SG 1 **Prepare for Verification**
SP 1.1 Select Work Products for Verification
SP 1.2 Establish the Verification Environment
SP 1.3 Establish Verification Procedures and Criteria
SG 2 **Perform Peer Reviews**
SP 2.1 Prepare for Peer Reviews
SP 2.2 Conduct Peer Reviews
SP 2.3 Analyze Peer Review Data
SG 3 **Verify Selected Work Products**
SP 3.1 Perform Verification
SP 3.2 Analyze Verification Results

IV. SOFTWARE DESIGN PROCESS

An engineering design is a model of the product or structure to be engineered. The model is used to *Evaluate suitability of proposed product and Communicate proposed product to others*[2].

Choosing a design for an engineering problem goes into a loop of steps, figure 2 shows these steps, which are[18]: *Understand the Needs*: understand the problem, what to accomplish, requirements, limitations, goals. *Brainstorm Different Designs*: investigate existing technologies and methods to use and analyze candidate solutions. *Select a Design*: based on the needs identified, select the most promising candidate solution. *Plan*: how the selected design will work, what are the materials and tools needed and how to test to make sure it works. *Create*: Build a prototype and test it against your design objectives. *Improve*: how the product can be improved.

The engineering design process is a series of steps that engineering teams use to guide them as they solve problem, a designer takes to go from first, identifying a problem or need to, at the end, creating and developing a solution that solves the problem or meets the need. The steps of the engineering design process are:*Define the Problem, Do Background Research, Specify Requirements, Create Alternative Solutions, Choose the Best Solution, Do Development Work, Build a Prototype, Test and Redesign*.

Fig. 2. Choosing a design for a problem

Software design process as defined by Andrews et al. [2] has relatively similar steps, these are:

- Recognition of Need.
- Definition of the Design Problem.
- Design Criteria and Constraints.
- The Design Loop *(Synthesis → Analysis → Decision-Making)*.
- Optimization.
- Evaluation.

However, Sommerville et al. [17] defined the steps for software design that are more general and can be applied to any software project. These steps are used as the base for software design process in the proposed model, which are:

- Problem understanding.
- Identify one or more solutions.
- Describe solution abstractions.
- Refine.

one should note that these are the stages of the design process rather than the phases, the phases of the design process as defined by Sommerville et al. [17] are *Architectural design, Abstract specification, Interface design, Component design , Data structure design and Algorithm design*.

V. Capability Maturity Model of Software Engineering Design Process *(SDP^{CMMI})*

This paper propose a new model, called Capability Maturity Model of Software *(SDP^{CMMI})*,Engineering Design Process to standardize software engineering design process. Proposed model is based on CMMI-DEV and inspired by Najar et al. [11] Capability Maturity Model of Software Requirements Process and Integration *(SRP^{CMMI})*.

A. Process Areas

SDP^{CMMI} defines a set of process areas, and each process area is assigned to a maturity level. Each one of the process areas has a software design goal and practices. Just like the CMMI-DEV, each practice of the SDP^{CMMI} consists of the following components:

- Purpose Statement: describes the purpose of the process area.
- Subpracties: guidance for interpreting and implementing a specific or generic practice.
- Technique: set of all techniques that will be used to perform the practice.
- Work product: the sample outputs from the practice.

B. Maturity Levels

Maturity levels consists of a set of process areas, and as stated by CMMI-DEV, to reach any maturity level, all the targeted practices should be satisfied.

SDP^{CMMI} consists of three maturity capability levels, just like SRP^{CMMI}, which are adapted from CMMI capability levels, these levels are: *Incomplete, Performed and Managed*, numbered from 0 to 2 respectively. Each one of the maturity levels of the SDP^{CMMI} provide description on how to improve the software engineering design process.

1) ***Maturity Level 0 - Incomplete****:* Incomplete process indicates to a process that is partially performed or not performed at all. Which means, one or more of the goals of the process area are not satisfied, thus, this level has no generic goal, since there is no reason to institutionalize a partially performed process.

2) ***Maturity Level 1 - Performed****:* Maturity level 1 process is characterized as a performed process. A performed process is a process that satisfies the goals assigned to that process. SDP^{CMMI} defines four process areas at this maturity level: *Problem understanding, Identify one or more solutions, Describe solution abstractions and Refine*. Each one of the process areas at this level is assigned to a goal as the following:

PA.1. **Problem understanding**

G.1. *Understand the problem, what is exactly needed to be done. Look at the problem from different angles to discover the design requirements.*

P.1.1. Identify problems *(using requirements)*.
P.1.2. Discover the design requirements.

PA.2. **Identify one or more solutions**

G.1. *Evaluate possible solutions and choose the most appropriate depending on the designer's experience and available resources.*

P.1.1. List candidate design solutions.
P.1.2. Evaluation.

P.1.3. Choose most appropriate design.

PA.3. **Describe solution abstractions**

G.1. *Use graphical, formal or other descriptive notations to describe the components of the design.*

P.1.1. Describe the components of the design.
P.1.2. Evaluation.

G.2. *Describe design phases.*

P.2.1. Architectural design.
P.2.2. Interface design.
P.2.3. Component design.
P.2.4. Data structure design.
P.2.5. Algorithm design.

PA.4. **Refine**

G.1. *Repeat process for each identified abstraction until the design is expressed in primitive terms.*

P.1.1. Express the design primitive terms.

3) ***Maturity Level 2 - Managed****:* A managed process is planned, performed, monitored, and controlled process which is meant to achieve given objectives, such as cost, schedule, and quality. As the title of this level indicates, it manages the way thing are done in the organization. At this maturity level, SDP^{CMMI} the exact same processes referred to by *(*SRP^{CMMI}*)* and CMMI-DEV, which are: *Project Planning, Organizational Training, Project Management and Control, and Process and Product Quality Assurance*.

VI. Conclusion and Future Work

This paper proposed a new model called Capability Maturity Model of Software Engineering Design Process SDP^{CMMI}, based on CMMI-DEV , to improve the productivity and the quality of software engineering design process. SDP^{CMMI} is inspired by Capability Maturity Model of Software Requirements Process and Integration *(*SRP^{CMMI}*)* introduced by Najar et al. [11] which was a model based on CMMI-DEV for requirements engineering improvement.

SDP^{CMMI} just like SRP^{CMMI}, consists of three maturity levels: *Incomplete, Performed and Managed*. Each one of the maturity levels has goals in order to reach that maturity level.

Future work would be to develop a generic model for implementation phase of software development life cycle.

References

[1] Maged Abdullah et al. "Benefits of CMM and CMMI-Based Software Process Improvement". In: *Software Process Improvement and Management: Approaches and Tools for Practical Development: Approaches and Tools for Practical Development* (2011), p. 224.

[2] Gordon C Andrews et al. *Introduction to professional engineering in Canada*. Toronto: Pearson Prentice Hall, 2009.

[3] Frederick P. Brooks Jr. *The Mythical Man-month (Anniversary Ed.)* Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 1995. ISBN: 0-201-83595-9.

[4] Bill Curtis, Herb Krasner, and Neil Iscoe. "A Field Study of the Software Design Process for Large Systems". In: *Commun. ACM* 31.11 (Nov. 1988), pp. 1268–1287. ISSN: 0001-0782. DOI: 10.1145/50087.50089. URL: http://doi.acm.org/10.1145/50087.50089.

[5] Alan M. Davis. *201 Principles of Software Development*. New York, NY, USA: McGraw-Hill, Inc., 1995. ISBN: 0-07-015840-1.

[6] Joseph M. Fox. *Software and Its Development*. Upper Saddle River, NJ, USA: Prentice Hall PTR, 1982. ISBN: 0138220980.

[7] Peter Freeman and David Hart. "A Science of Design for Software-intensive Systems". In: *Commun. ACM* 47.8 (Aug. 2004), pp. 19–21. ISSN: 0001-0782. DOI: 10.1145/1012037.1012054. URL: http://doi.acm.org/10.1145/1012037.1012054.

[8] Diane L Gibson, Dennis R Goldenson, and Keith Kost. *Performance results of CMMI-based process improvement*. Tech. rep. DTIC Document, 2006.

[9] Ming-Chang Lee and To Chang. "Applying TQM, CMM and ISO 9001 in knowledge management for software development process improvement". In: *International Journal of Services and Standards* 2.1 (2005), pp. 101–115.

[10] Bertrand Meyer. *Object-Oriented Software Construction*. 1st. Upper Saddle River, NJ, USA: Prentice-Hall, Inc., 1988. ISBN: 0136290493.

[11] Sireen Kamal Najjar and Khalid T. Al-Sarayreh. "Capability Maturity Model of Software Requirements Process and Integration (SRPCMMI)". In: *Proceedings of the International Conference on Intelligent Information Processing, Security and Advanced Communication*. IPAC '15. New York, NY, USA: ACM, 2015, 68:1–68:5. ISBN: 978-1-4503-3458-7. DOI: 10.1145/2816839.2816856. URL: http://doi.acm.org/10.1145/2816839.2816856.

[12] Mark C Paulk. "Comparing ISO 9001 and the capability maturity model for software". In: *Software Quality Journal* 2.4 (1993), pp. 245–256.

[13] Mark C Paulk. "Extreme programming from a CMM perspective". In: *IEEE software* 18.6 (2001), pp. 19–26.

[14] Mark C Paulk. "Using the software CMM in small organizations". In: Pacific Northwest Software Quality Conference. 1998.

[15] Paul Ralph and Yair Wand. "A proposal for a formal definition of the design concept". In: *Design requirements engineering: A ten-year perspective*. Springer, 2009, pp. 103–136.

[16] CMMI SEI. *CMMI for development, Version 1.3, Software Engineering Institute*. 2010.

[17] Ian Sommerville. *Software Engineering (5th Ed.)* Redwood City, CA, USA: Addison Wesley Longman Publishing Co., Inc., 1995. ISBN: 0-201-42765-6.

[18] SP Tayal. "Engineering design process". In: *International Journal of Computer Science and Communication Engineering* (2013), pp. 1–5.

[19] Gerald M. Weinberg. *The Psychology of Computer Programming*. New York, NY, USA: John Wiley & Sons, Inc., 1985. ISBN: 0442292643.