ASSESSMENT OF SOFTWARE MAINTENANCE CAPABILITY: A Model and Its Architecture

(Alain April[1], Alain Abran[1], Reiner R. Dumke[2])
[1]École de Technologie Supérieure, Montréal, Canada, aapril@ele.etsmtl.ca , aabran@ele.etsmtl.ca
[2]Otto von Guericke University of Magdeburg, Germany, dumke@ivs.cs.uni-magdeburg.de

Abstract

Maintaining and supporting the software of an organization is not an easy task, and software maintenance managers do not currently have access to tools to evaluate strategies for improving the specific activities of software maintenance. This article presents the new architecture (version 2.0) of the software maintenance capability maturity model (**SM^{CMM}**). The contributions of this paper are: 1) to present a categorization of the software maintenance processes using a representation similar to that in ISO12207; and 2) present the new architecture of the model, which highlights the unique processes of the maintainers.

Key Words

Software maintenance, maturity model, software improvement, software quality

1. Introduction

Using the SW-CMM (and, more recently, the CMMi) to assess a software maintenance organization is always difficult, since this model is based mostly on a project point of view and does not include the maintenance-unique activities typical of software maintenance duties. This leaves maintainers without any guidance for either the key processes or the activities that are unique to their domain [1], for example:

- Transition: a controlled and coordinated sequence of activities during which a system is transferred progressively from the developer to the maintainer;
- Service Level Agreements (SLAs) and specialized maintenance contracts: negotiation and management of maintenance-specific SLAs and domain-specific contracts;
- Help Desk handling of MRs and PRs: Maintainers use a problem-handling process to prioritize, document and route the requests they receive;
- Acceptance/rejection of MRs: Maintainers will not accept modification request work over a certain size/effort/complexity, and will reroute these requests to a developer;
- Impact Analysis: describes how to conduct, cost effectively, a complete analysis of the impact of a change to existing software.

From 1994 to 1996, our software engineering research laboratory, sponsored by research grants from major Canadian telecommunications companies, developed an initial maturity model specific to software maintenance [2]. This model was used as a complement to the SW-CMM. We acknowledged then that further work would be required to increase its coverage and to map it to upcoming versions of the many ISO standards and models..

We are often questioned about the need for such a model. Do we need another maturity model, considering the rise of agile development methods? Xtreme's Ron Jeffries states that his methodology cuts a vertical slice in the CMM in levels 2 to 5 [3]. He also mentions that processes are important, in agile and other software engineering methodologies, and that, when followed properly, his methodology has practices up to CMM level 5 [4]. An experiment with Xtreme maintenance is presented in [5] which demonstrates this point of view. By contrast, Mark Paulk states that some of the Xtreme practices are still controversial when used outside the Internet development domain [6]. We are still convinced that maintainers, more than ever, need support in, and guidance on, improving their own processes, and that a maturity model can help maintenance organizations of all sizes and maturity levels.

This paper presents the new version of the software maintenance capability maturity model (**SM^{CMM}**), including its architecture and the key process areas that are distinct and common to this model and the Capability Maturity Model Integration for Software Engineering (CMMi). The label Software Maintenance-Capability Maturity Model is usually abbreviated to SM-CMM; however, in this text the label **SM^{CMM}** will be used for greater readability and to avoid confusion with the CMMi abbreviation. The model's purpose is presented in section 3, the key maintenance processes in section 4, its foundation and architecture are presented in sections 5 and 6 respectively, followed by the conclusion in section 7.

2. The Model's Purpose

SM^{CMM} was designed as a customer-focused benchmark for either:

- Auditing the software maintenance capability of a software maintenance service supplier or outsourcer; or
- Internal software maintenance organizations.

The **SM^{CMM}** has been developed from a customer perspective, as experienced in a competitive, commercial environment. The ultimate objective of improvement programs initiated as a result of an **SM^{CMM}** assessment is increased customer (and shareholder) satisfaction, rather than rigid conformance to the standards referenced by this document.

A higher capability, in the **SM^{CMM}** context, means, for customer organizations:

a) Reaching the target service levels and delivering on customer priorities;
b) Implementation of the best practices available to software maintainers;
c) Obtaining transparent software maintenance services and incurring costs that are competitive;
d) The shortest possible software maintenance service lead times.

For a maintenance organization, achieving a higher capability can result in:

a) Lower maintenance and support costs;
b) Shorter cycle time and intervals;
c) Increased ability to achieve service levels; and
d) Increasing ability to meet quantifiable quality objectives at all stages of the maintenance process and services.

3. Model Scope

Models are often an abstract representation of reality. For a better mapping with the maintainers' reality, an **SM^{CMM}** must include many of the essential perspectives of the software maintainer, and as much as possible of the maintainer's practical work context (see Figure 1 below).

The model is not intended to describe specific techniques or all the technologies used by maintainers. The decisions pertaining to the selection of specific techniques or technologies are specific to each organization.

Users of the model must instantiate the generic model in the context of their user organization. To achieve this, professional judgment will be required to evaluate how an organization benchmarks against the generic model.

4. Key Maintenance Processes

Key software maintenance processes have been grouped into three classes (Figure 1). This provides for easier alignment with the ISO 12207 standard [7]:

a) Primary processes (operational);
b) Support processes (supporting the primary processes); and

c) Organizational processes offered by the IT unit or by other departments of the organization (for example: finance, human resources, purchasing, etc.).

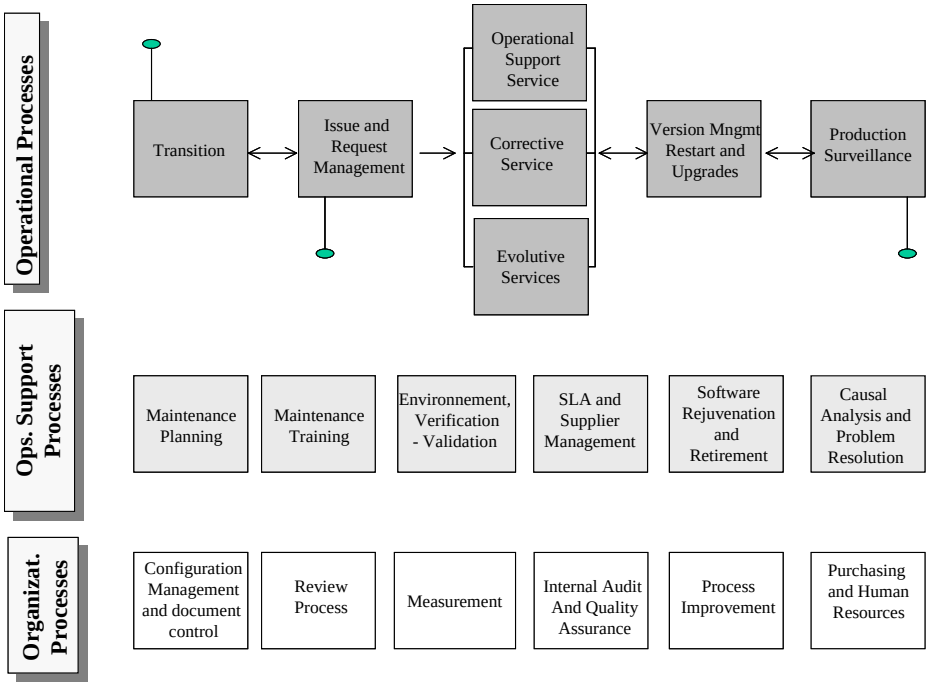

Figure 1: A classification of the Software Maintainer's Key Processes

The key operational processes (also called primary processes) that a software maintenance organization uses are initiated at the start of software project development and then maintained subsequently, beginning with the transition process. The *Transition process* ensures that the software project is controlled and that a structured and coordinated approach is used to transfer the software to the maintainer. In this process, the maintainer will focus on the maintainability of this new software.

Once the software has become the responsibility of the maintainer, the *Issue and Service Request Management process* handles all the daily issues, problem reports, change requests and support requests. These are the daily services that must be managed efficiently. The first step in this process is to assess whether a request is to be addressed, rerouted or rejected (on the basis of the service-level agreement and the nature of the request and its size) [8]. Accepted requests are documented, prioritized, assigned and processed in one of the service categories: 1) *Operational Support process* (*which typically does not necessitate any modification of software*); 2) *Software Correction process*; or 3) *Software Evolution process*.

Note that certain service requests do not lead to any modification to the software. In the model, they are referred to as 'operational support' activities, and these consist of: a) replies to questions; b) provision of information and counselling; and c) helping customers to better understand the software, a transaction or its documentation.

The last two main operational processes concern the *Version Management process*, which will move items to production, and the *Production Surveillance process*, which will ensure that the operational environment has not been degraded. Maintainers always oversee the behavior of the operational system and its environments for signs of degradation. They will quickly warn other support groups when something unusual happens (operators, technical support, scheduling, networks and desktop support) and judge whether or not it is an instance of service degradation which needs to be investigated.

A process which is used, when required, by an operational process is said to be an operational support process. In this classification, we include: a) the many maintenance planning processes; b) the maintainer's education and training; c) the maintenance environments and testing; d) management of the contractual aspects and service-level agreements; e) rejuvenation or retirement of software; and, finally, f) resolution of problems. These are all key activities which are available to support some operational process activities.

Organizational processes are typically offered by the IT department and by other departments in the organization (for example: human resources, finance, quality assurance and ISO9001). While it is important to measure and assess these processes, it is often easier for the maintainer to start defining the operational processes and operational support processes.

This generic model should help users understand and represent the various key software maintenance processes. What is important is that these processes be explicitly listed and classified based on their type (operational, support or organizational). The list presented here is provided only to serve as a model (one that the companies that helped develop the **SM**$^{\mathbf{CMM}}$ have found useful), and other **SM**$^{\mathbf{CMM}}$ users should create their own list based on the terminology and classifications that suit their particular organization.

5. SM^{CMM} Foundation

The **SM**$^{\mathbf{CMM}}$ is based on the Software Engineering Institute (SEI) Capability Maturity Model Integration for Software Engineering (CMMi), version 1.1 [9] and *Camélia* [10]. The **SM**$^{\mathbf{CMM}}$ must be viewed as a complement to the CMMi, especially for the processes that are common to developers and maintainers, for example: a) process definition; b) development; c) testing; d) configuration management; and e) QA practices are also used by the software development organization.

The architecture of the **SM**$^{\mathbf{CMM}}$ (further described in section 6) differs slightly from that of the CMMi version 1.1. The most significant differences are:

1. A roadmap category to further define the Key Process Areas,
2. Detailed references to papers and examples on how to implement the practice.

The **SM**$^{\mathbf{CMM}}$ incorporates additional practices from the following topics:

1. Event and Service Request Management;
2. Maintenance Planning activities specific to maintainers (version, SLA, impact analysis);
3. Service Level Agreement;
4. Software Transition;
5. Operational Support;
6. Problem Resolution Process with a Help Desk;
7. Software Rejuvenation, Conversion and Retirement.

6. SM^{CMM} Architecture

The CMMi has recently adopted the continuous representation that has been successfully used in the past by other models such as: Bootstrap [11], *Camélia* and ISO/IEC TR15504 (Spice) [12]. This model uses a continuous representation, as it helps to: a) conform to Spice recommendations; b) obtain a more granular rating for each roadmap and domain; and c) identify a specific practice across maturity levels and identify its path from level zero (absent) to a higher level of maturity.

The **SM**$^{\mathbf{CMM}}$ is also based on the concept of a roadmap. A roadmap is a set of related practices which focuses on an organizational area or need, or a specific element within the software maintenance process. Each roadmap represents a significant capability for a software maintenance organization. Within a given roadmap, the level of a practice is based on its respective degree of maturity. The most fundamental practices are located at a lower level, whereas the most advanced ones are located at a higher level. An organization will mature through the roadmap levels. Lower-level practices must be implemented and sustained for higher-level practices to achieve maximum effectiveness.

Each of the 6 maturity levels can be characterized, in the SM^{CMM} model, as follows (Figure 2):

Lvl	Level Name	Risk	Interpretation
0	Non-existent	Highest	no sense of process
1	Initial	Very high	ad hoc maintenance process
2	Repeatable but intuitive	High	basic request-based process
3	Defined Process-'Oriented'	Medium	state-of-the-art maintenance process
4	Managed and Measurable	Low	generally difficult to achieve now
5	Optimized	Very low	technologically challenging to attain

Figure 2: **SM^{CMM}** Capability Levels.

Domains are organized into Capability Areas, each representing a significant capability for a software maintenance organization. There are 4 Process Domains, which can span the six **SM^{CMM}** Capability levels. A representation of the span of each Capability Area is shown in Table 1. Roadmaps are not presented here in order to lighten the text and figures. Each Process Domain incorporates one or more Key Process Areas. Each Key Process Area comprises Roadmaps with one or more Practices which span several **SM^{CMM}** levels. The complete version 2.0 model has 4 Domains, 18 Key Process Areas, 74 Roadmaps and 443 Practices.

Process Domain	Key Process Area	Roadmap
Process Management	-Process Focus	Responsibility/Communications, Information Gathering, Findings, Action Plans.
	-Process/Service Definition	Documentation, Standardizing Adaptations, Communication, Repository.
	-Training	Requirements, Plans, Personal, New hire, Projects, Users.
	-Maint. Process Performance	Measure definition, Baselines, Quantitative Mgmt., Prediction models.
	-Innovation and Deployment	Identification, Analysis, Piloting, Deployment, Benefit measurement.
Maintenance Request Management	-Event/Service Request	Contact structure and communications, Events/Service Requests Management.
	-Planning	Strategic, Project Transition, Disaster Recovery, Capacity, Versions, Impact Anal.
	-Monitoring/Control of Events /Service Request	Follow-up, Review progress, corrective action.
	-SLA/Supplier Agreements Management	Account Mgmt., SLA's & Contracts, Execute agreements, Report & bill.
Software Evolution	-Software Transition	Involvement, Surveillance, Knowledge Transfer, Documentation, Acceptance.
Engineering	-Operational Support	Monitoring, Support, Functional, ad-hoc Requests.
	-Software Evolution/Correction	Detailed Design, Construction, Unit & Integration Testing, Documentation.
	-Software Verification/Validation	Reviews, Acceptance tests, move to prod.
Support to Software Evolution Engineering	-Software Configuration Management	Change Mngmt., Baseline config., control of Components.
	-Process/Product QA	Object. Evaluation, Identify/Document non-

	conformances, Communicate, Follow-up.
-Measurement/Analysis	Define measurement prog., Collect & Analyze, Use of measures, Communication.
-Causal Analysis/Prob. Resolut.	Investigate, Identify, Analyze, Propose solutions.
-Software Rejuv., Conversion and Retirement of Software	Re-documentation, Restructuration, Reverse Engineering, Re-engineering, Migration, Retirement

Table 1: Domain/Key Process Area Relationships of the **SM^{CMM}**

7. Next steps

The next experimental release of **SM^{CMM}** will be version 2.1, which will be based on feedback and lessons learned from its use in some organizations. The full version is to be published in 2004, initially in a French edition of the model, to be followed shortly thereafter by its English translation.

References

[1] A. Abran and J.W. Moore Executive Editors; P. Bourque and R. Dupuis: Editors; L.L. Tripp Chair of IAB, Guide to the Software Engineering Body of Knowledge - Trial Version, IEEE Computer Society Press, Dec. 2001.

[2] M. Zitouni, A. Abran, A Model to Evaluate and Improve the Quality of the Software Maintenance Process," *Proc. 6th International Conference on Software Quality Conference*, Ottawa, Canada, ASQC-Software Division, 1996.

[3] Jeffries, R., Anderson, A., Hendrickson, C. (2000). Extreme Programming Installed, Addison Wesley.

[4] Jeffries, R. (2002). Extreme Programming and the Capability Maturity Model, XPMagazine, [Online]. www.xprogramming.com/xpmag/xp_and_cmm.htm. (checked on December 5, 2003).

[5] C. Poole, W. Huisman. *Using Extreme Programming in a Maintenance Environment*, IEEE Software November/December 2001, pp.42-50.

[6] Paulk, M. (2002). Agile Methodologies and Process Discipline, Crosstalk, October issue.

[7] ISO/IEC 12207, Information Technology – Software Life Cycle Processes, International Organization for Standardization, Geneva, 1995.

[8] A. April, J. Bouman, A. Abran, D. Al-Shurougi , Software Maintenance in a Service Level Agreement. *Proc. 4th European Software Conference*, FESMA, Heidleberg, Germany, May 2001.

[9] SEI, "Capability Maturity Model Integration for Software Engineering (CMMi), Version 1.1", CMU/SEI-2002-TR-028, ESC-TR-2002-028, Software Engineering Institute (SEI), Carnegie Mellon University, Pittsburgh, USA, 2002.

[10] Camélia, Modèle d'évolution des processus de développement, maintenance et d'exploitation de produits informatiques – Version 0.5, Projet France-Québec, Montréal, Canada, 1994.

[11] Bootstrap, Esprit project #5441, European Commission, Brussels, Belgium, 1991.

[12] ISO/IEC TR 15504, Information Technology – Software Process Assessment (parts 1-9), International Organization for Standardization, Geneva, Switzerland 1998.