Maturity Models in the Software Engineering Literature: An Analytical Overview

Rafa E. Al-Qutaish *rafa.alqutaish@gmail.com*
Associate Professor of Software Engineering
412-1682 Chemin du Tremblay,
Longueuil, QC J4N 1E1, Canada

Abstract

A maturity models are structured and systematic frameworks designed to outline the key characteristics, attributes, and features that define effective processes or products. These models serve as a reference point or benchmark, allowing organizations to assess, compare, and evaluate the relative capabilities and effectiveness of their processes or products. By providing a clear pathway for progression, maturity models help organizations identify areas of strength, opportunities for improvement, and strategies for achieving higher levels of efficiency and quality. In the context of software engineering, maturity models are widely recognized and utilized as essential tools for evaluating both the development processes and the resulting software products. They are particularly valuable in assessing how well processes or products align with industry standards and best practices. Generally, maturity models in this domain are categorized into two primary types: process maturity models and product maturity models. Process maturity models focus on evaluating the evolution, refinement, and optimization of software development processes. These models assess how systematically and efficiently an organization develops, tests, delivers, and maintains software, aiming to enhance process performance and reliability over time. On the other hand, product maturity models concentrate on evaluating the quality, reliability, and overall sophistication of the software product itself. These models examine factors such as functionality, usability, scalability, and maintainability, offering insights into the readiness and robustness of the product. This paper aims to provide a comprehensive overview of the role and significance of maturity models within software engineering. It highlights their importance as tools for performance evaluation, fostering continuous improvement, and driving standardization across the industry.

Keywords: Maturity Models, OSMM, SMM, SPQMM, TMM, CMMI-SW, ISO 15504, Nastro Model.

1. INTRODUCTION

A maturity model is a structured collection of elements that describe the characteristics of effective processes or products (Wendler, 2012). However, a maturity model can be used as a benchmark for assessing different processes or products for equivalent comparison (Golden, 2004). Also, a maturity model evaluates the development and performance of an individual or group. Unlike performance models, which compare an individual's activities and metrics to an objective standard, maturity models assess performance based on predefined stages of growth and expertise.

A maturity model is a leveled benchmark that offers organizations a framework to assess their current state. The goal of maturity models is to steer the focus of discussions. The main components of a maturity mode are key process area, Criteria, and levels (Torriano, 2022). However, hundreds of maturity models have been proposed by researchers and practitioners across multiple domains (Bruin et al., 2005; Weber, 2008). In addition, a maturity model is a conceptual framework that maps the development of an organization's capabilities within a specific domain. It is structured into stages, each representing a more advanced level of maturity

(*Maturity Model: A framework that assesses the level of maturity of an organization's processes and practices*).

Utilizing a maturity model brings significant benefits to organizations, including a structured approach to continuous improvement, the ability to benchmark against industry standards, and improved overall performance (*Maturity Model: A framework that assesses the level of maturity of an organization's processes and practices*).

- Continuous Improvement: Maturity models offer a clear framework for progression, enabling organizations to set achievable goals and monitor their development effectively. This structured approach fosters ongoing improvement.
- Benchmarking: By comparing their capabilities with industry standards, organizations can identify strengths and weaknesses. This benchmarking process helps align improvement efforts with best practices, driving better outcomes.
- Enhanced Performance: Advancing systematically through the maturity levels improves process capability and efficiency, ultimately leading to superior performance and greater operational effectiveness.

In the software engineering literature, there are many capability and maturity models. These capability and maturity models could be classified into two categories, that is, process and product capability and maturity models based on what aspect of the software they could be applied.

Maturity models are commonly applied to assess the as-is situation, to derive and prioritize improvement measures, and to control progress (Iversen et al., 1999). However, maturity models are expected to disclose current and desirable maturity levels and to include respective improvement measures.

This paper is structured as follows: Section 2 presents some examples of the many available process maturity models related to software engineering. By contrast, only five maturity models are identified as dealing with the software product. In Section 3, we discuss these some product maturity models, that is: the Open Source Maturity Model (OSMM) (Golden, 2004), SCOPE Maturity Model (SMM) (Jakobsen & Punter, 1999), Software Product Quality Maturity Model (SPQMM) (Al-Qutaish & Abran, 2011), and the Software product Maturity Model (Nastro, 1997). At the end of this paper – in Section 4 – we discuss the strengths and weaknesses of maturity models which are related to the software product rather than to the software process. Finaly, Section 5 concludes the paper.

2. SOFTWARE PROCESS MATURITY MODELS

The maturity models of software development processes are useful because they indicate different levels of process performance and, consequently, the direction in which a software process should improve (McBride et al., 2004). In this section we will discuss the following process capability and maturity models:

1. Capability Maturity Model Integration for Software Engineering – CMMI-SW (SEI, 1993, 2002a, 2002b).
2. Testing Maturity Model – TMM (Burnstein et al., 1996a, 1996b).
3. ISO Process Assessment Model – ISO 15504(ISO, 2003, 2004a, 2004b, 2004c, 2006).

2.1 Capability Maturity Model Integration for Software Engineering (CMMI-SW)

The Capability Maturity Model (CMM) was developed by the Software Engineering Institute (SEI) of Carnegie Mellon University in response to a request to provide the U.S. Department of Defense with a method for assessing its software contractors (SEI, 1993). Within the updated Capability Maturity Model Integration (CMMI) version are currently embedded four bodies of knowledge (disciplines) (SEI, 1993, 2002a, 2002b):

1. System Engineering.
2. Software Engineering.
3. Integrated Product and Process Development.
4. Supplier Sourcing.

In this subsection, we provide an overview of only the Capability Maturity Model Integration for Software Engineering (CMMI-SW).

The components of the CMMI-SW are process areas, specific goals, specific practices, generic goals, generic practices, typical work products, sub-practices, notes, discipline amplifications, generic practice elaborations and references (SEI, 2002a). The CMMI-SW can organize process areas in a staged or a continuous representation. The staged representation includes five maturity levels to support and guide process improvement; in addition, it groups process areas by maturity level, indicating which process areas to implement to achieve each maturity level (SEI, 2002a). Figure 1 presents the CMMI-SW maturity level architecture.

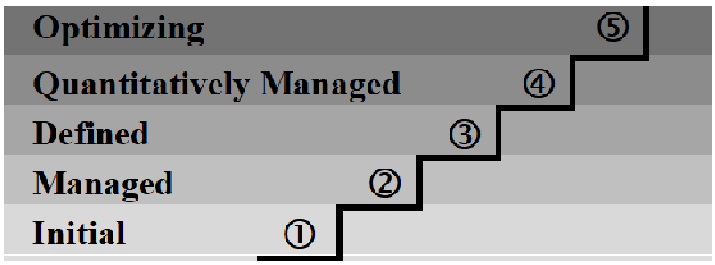

FIGURE 1: CMMI-SW maturity levels(SEI, 2002a).

Furthermore, Figure 2 shows the CMMI-SW model components in a staged representation, and illustrates the relationships between those components (SEI, 2002a).

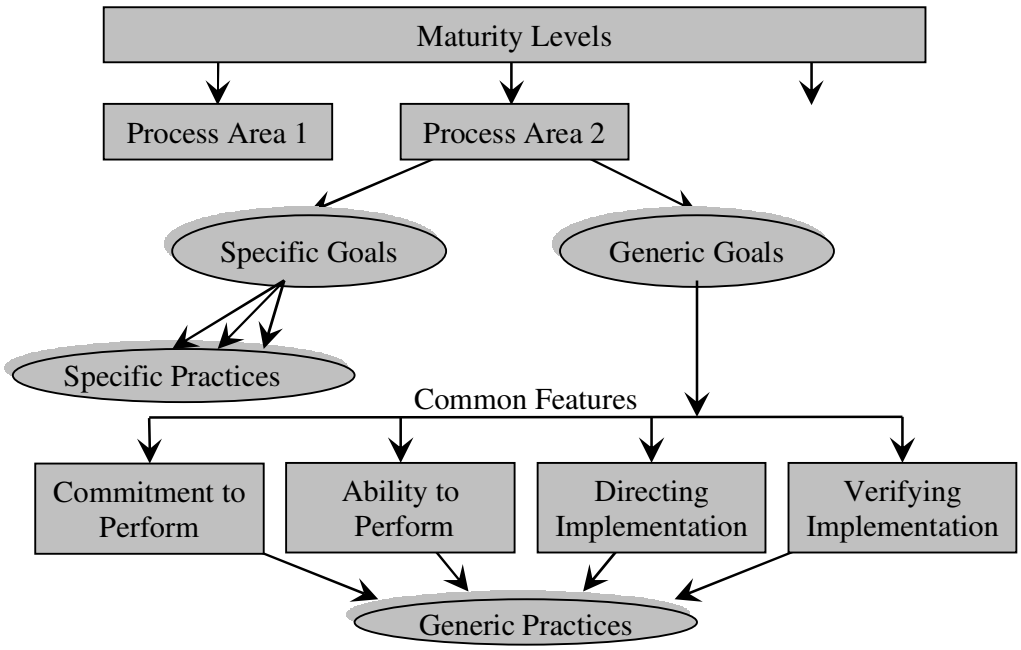

FIGURE 2: CMMI-SW components (SEI, 2002a).

2.2 Testing Maturity Model

The Testing Maturity Model (TMM) is a framework designed to assess and improve the maturity of an organization's software testing processes. It provides a structured approach to evaluate the effectiveness of testing practices, helping organizations improve their testing capabilities and align them with best practices. The model aims to guide teams from initial ad-hoc practices to more refined, optimized processes that ensure high-quality software delivery(Burnstein et al., 1996a, 1996b). However, the key Concepts of TMM(Burnstein et al., 1996a, 1996b)**:**

1. Maturity Levels: The model defines a set of maturity levels that an organization can progress through. Each level represents a different stage of testing process maturity, ranging from the absence of formal testing practices to a highly optimized and automated testing environment(Burnstein et al., 1996a, 1996b).
2. Process Areas: These are the core activities or focus areas for testing, such as test planning, test execution, defect management, test automation, and continuous improvement. The TMM evaluates how well an organization is performing in these areas(Burnstein et al., 1996a, 1996b).
3. (Burnstein et al., 1996a, 1996b)The model encourages organizations to adopt a mindset of continuous improvement. As organizations progress through the maturity levels, they refine and enhance their testing practices to reduce defects, improve test coverage, and increase the efficiency of the testing process(Burnstein et al., 1996a, 1996b).

While the exact number of levels may vary depending on the version or specific adaptation of the model, a typical TMM often has five – see Figure 3 - maturity levels(Burnstein et al., 1996a, 1996b):

1. Level 1: Initial (Ad-Hoc Testing)
   - Testing is chaotic, unstructured, and reactive.
   - No formal test processes or methodologies are followed.
   - Limited understanding of testing roles and responsibilities.
2. Level 2: Managed (Basic Process)
   - Basic testing processes are established and followed.
   - Test planning and execution are somewhat defined, but still reactive.
   - The focus is on delivering the product, with little emphasis on process improvement.
3. Level 3: Defined (Standardized Process)
   - Testing processes are well-defined, standardized, and integrated into the software development lifecycle (SDLC).
   - Test strategies, methodologies, and tools are chosen to fit project needs.
   - Documentation is improved, and testing teams are trained and skilled.
4. Level 4: Measured (Quantitative Management)
   - Metrics are used to manage and control testing processes.
   - Data-driven decisions are made to improve quality and efficiency.
   - Test coverage, defect density, and other metrics are used to monitor and guide test activities.
5. Level 5: Optimizing (Continuous Improvement)
   - Focus on continuous process improvement.
   - Testing processes are refined based on lessons learned and feedback.
   - Test automation is extensively used, and there is a proactive approach to managing risks and improving quality.
   - The organization fosters a culture of testing excellence and innovation.

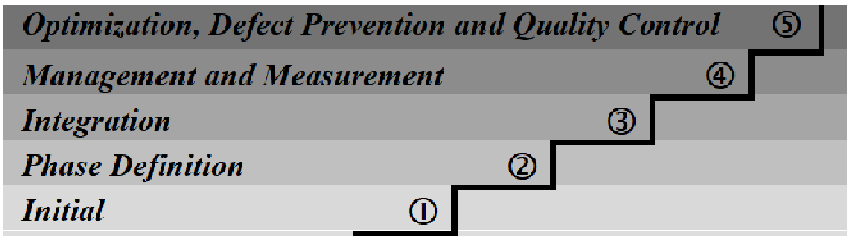

FIGURE 3: Testing maturity levels.

The Testing Maturity Model provides a roadmap for organizations to assess and improve their software testing capabilities. By following the maturity levels and focusing on key process areas, companies can move from unstructured, ad-hoc testing practices to a more strategic and data-driven approach to quality assurance(Burnstein et al., 1996a, 1996b). Thew following are the benefits of the TMM(Burnstein et al., 1996a, 1996b)**:**

- Improved Software Quality: As an organization matures in testing practices, the quality of the software improves due to more thorough and structured testing(Burnstein et al., 1996a, 1996b).
- Increased Efficiency: Well-defined and optimized testing processes help in better resource management and reduced time to market(Burnstein et al., 1996a, 1996b).
- Cost Reduction: With higher maturity, organizations can detect defects early, reducing the cost of fixing bugs later in the development lifecycle(Burnstein et al., 1996a, 1996b).
- Better Risk Management: By implementing metrics and continuous improvement practices, organizations can identify risks early and mitigate them effectively(Burnstein et al., 1996a, 1996b).

2.3 ISO 15504: Software Process Assessment

Also known as SPICE (Software Process Improvement and Capability Determination). ISO 15504 consists of a set of documents related to Software Process Assessment. It was first published in 1998 as a series of 9 Technical Reports. During 2003 to 2005, ISO has re-published this international standard as a 5-part series:

1. ISO 15504-1: Concepts and Vocabulary (ISO, 2004a).
2. ISO 15504-2: Performing an Assessment (ISO, 2003).
3. ISO 15504-3: Guidance on Performing an Assessment (ISO, 2004b).
4. ISO 15504-4: Guidance on use for Process Improvement and Process Capability Determination (ISO, 2004c).
5. ISO 15504-5: An Exemplar Process Assessment Model (ISO, 2006).

The first Part – Concepts and Vocabulary – is an entry point into ISO 15504. It gives an introduction to the concepts of this international standard, and defines a number of related terms (ISO, 2004a). In addition, this part describes how the other four parts fit together, and provides guidance for their selection and use (ISO, 2004a). Figure 4 shows a potential roadmap for users of this international standard (ISO, 2004a).

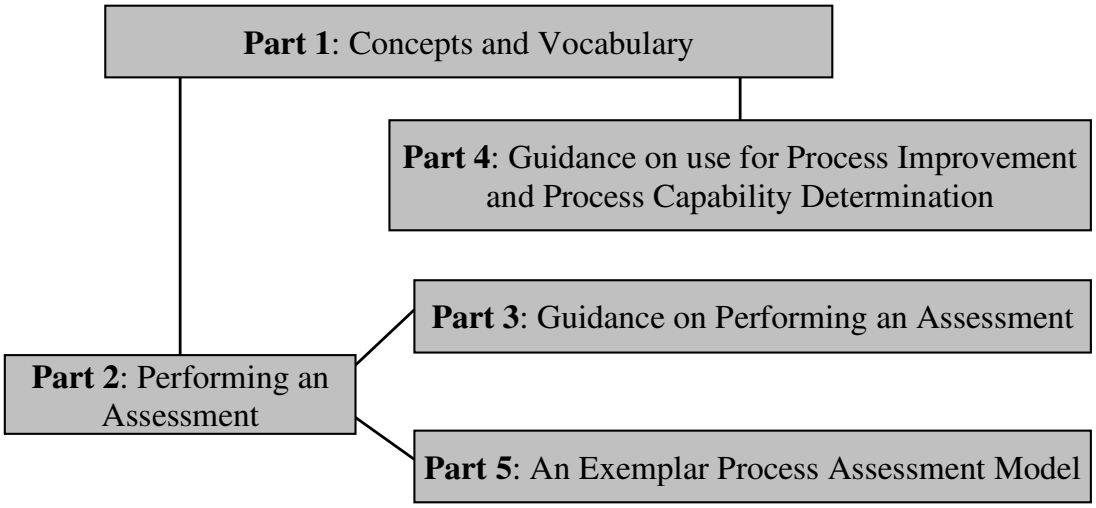

FIGURE 4: A potential roadmap for the users of ISO 15504 (ISO, 2004a).

The second Part – Performing an Assessment – of this international standard contains normative requirements for process assessment and for process models in an assessment, and defines a measurement framework for evaluating process capability. The measurement framework defines nine process attributes that are grouped into six process capability levels that define an ordinal scale of capability that is applicable across all selected processes. In addition, this part describes the relationships between the components of the process assessment model, as in Figure 5 (ISO, 2003).

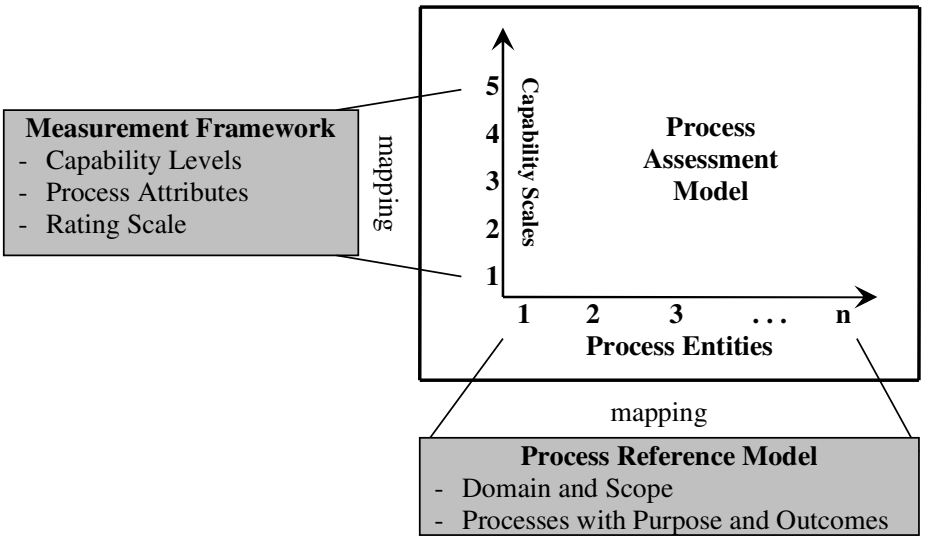

FIGURE 5: Process assessment model relationships (ISO, 2003).

Figure 6 illustrates the relationships between the process attributes and their ratings and the corresponding capability levels. In this figure 6, the capability levels start at level one, that is, level zero is excluded since it indicates that the process is not implemented, or fails to achieve its process purpose.

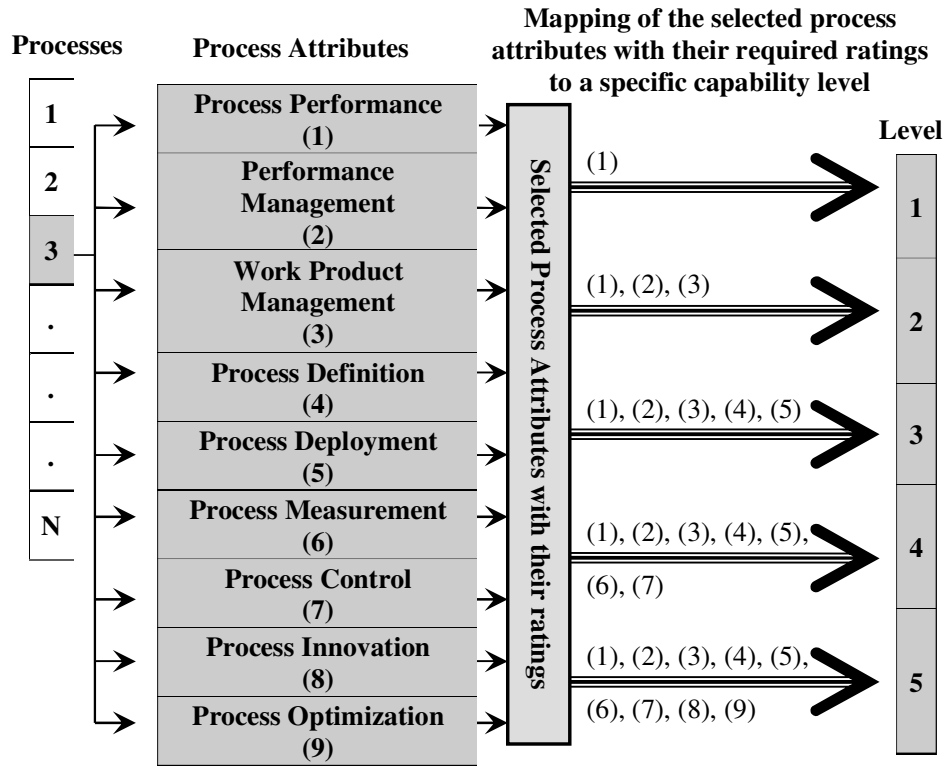

FIGURE 6: The relationships between the process attributes, their ratings and the corresponding capability levels.

Furthermore, this Part 2 – Performing an Assessment – of the ISO 15504 introduces the following rating categories to be used in order to rate each of the process attributes (ISO, 2003):

- N: Not achieved (0% - 15% achievement).
- P: Partially achieved (15% - 50% achievement).
- L: Largely achieved (50% - 85% achievement).
- F: Fully achieved (85% - 100% achievement).

The third Part – Guidance on Performing an Assessment – provides guidance on how to meet the minimum set of requirements for performing an assessment contained in the second part – Performing an Assessment – of this standard (ISO, 2004b). It provides an overview of process assessment and interprets the requirements through the provision of guidance on (ISO, 2004b):

1. Performing an assessment.
2. Measurement framework for process capability.
3. Process reference models and process assessment models.
4. Selecting and using assessment tools.
5. Competency of assessors.
6. Verification of conformity.

In addition, this part also provides an exemplar documented assessment process in its Annex A (ISO, 2004b).

The fourth Part – Guidance on use for Process Improvement and Process Capability Determination – provides guidance on how to utilize a conformant process assessment within a process improvement program or for process capability determination (ISO, 2004c). Within a

process improvement context, process assessment provides a means of characterizing an organizational unit in terms of the capability of selected processes. Analysis of the output of a conformant process assessment against an organizational unit's business goals identifies strengths, weaknesses and risks related to the processes. In addition, this can help determine whether the processes are effective in achieving business goals, and provide the drivers for making improvements. Process capability determination is concerned with analyzing the output of one or more conformant process assessments to identify the strengths, weaknesses and risks involved in undertaking a specific project using the selected processes within a given organizational unit (ISO, 2004c).

Finally, the fifth Part – An Exemplar Process Assessment Model – provides an exemplar model for performing process assessments that is based upon and directly compatible with the Process Reference Model in ISO 12207 Amendment 1 and Amendment 2 (ISO, 2006). The process dimension is provided by an external Process Reference Model, which defines a set of processes, characterized by statements of process purpose and process outcomes (ISO, 2006). The capability dimension is based upon the Measurement Framework defined in Part 2 – Performing an Assessment – of this standard. The assessment model(s) extend the Process Reference Model and the Measurement Framework through the inclusion of a comprehensive set of indicators of process performance and capability (ISO, 2006).

The potential users of this set of standards are the following (ISO, 2004a):

1- Assessors.
2- Acquirers.
3- Suppliers.

3. SOFTWARE PRODUCT MATURITY MODELS

In the software engineering literature, we find only the following two maturity models which are related to the software product:

1. Open Source Maturity Model – OSMM (Golden, 2004).
2. Software Product Maturity Model (Nastro, 1997).
3. SCOPE Maturity Model (SMM) (Jakobsen & Punter, 1999).
4. Software Product Quality Maturity Model (SPQMM) (Al-Qutaish & Abran, 2011).

It must be noted, however, that these two models of software product do not address the quality of these products. Within this section, we will provide a brief description for these two models.

3.1 Open-Source Maturity Model (OSMM)

The Open Source Maturity Model (OSMM) (Golden, 2004) is designed to help organizations successfully implement open-source software. The OSMM is a three-phase process, and performs the following tasks, as in Figure 7:

1. Assessment of the maturity element.
2. Assignment of the weighting factor.
3. Calculation of the product maturity score.

	Phase 1: Assess Maturity Element				Phase 2	Phase 3
	Define Requirements	Locate Resources	Assess Element Maturity	Assign Element Score	Assign Weighting Factor	Calculate Product Maturity Score
Product Software						
Support						
Documentation						
Training						
Product Integration						
Professional Services						

FIGURE 7: The OSMM three-phase evaluation process(Golden, 2004).

The first phase consists of the following steps:

1. Define the requirements,
2. Locate the resources,
3. Assess the element maturity
4. Assign the element score.

The second phase involves assigning the objective weighting factors that are provided as default weightings and that can be changed by individual organizations to reflect their particular needs (Golden, 2004).

The last phase involves calculating the product maturity score by multiplying the score of each element by its weight, and then summing the results to obtain the output of the OSMM assessment as a numeric score between zero and 100. This score may be compared against recommended levels for different purposes, which vary according to whether an organization is an early adopter or a pragmatic user of information technology (Golden, 2004), see Figure 8 for the recommended minimum OSMM scores.

Purpose of Use	Type of User	
	Early Adopter	Pragmatist
Experimentation	25	40
Pilot	40	60
Production	60	70

FIGURE 8: Recommended minimum OSMM scores (Golden, 2004).

Using the key software concept of maturity (i.e., how far along a product is in the software lifecycle, which dictates what type of use may be made of the product), the OSMM assesses the maturity level of the following key product elements (Golden, 2004):

1. Software.
2. Support.
3. Documentation.
4. Training.
5. Product integration.
6. Professional Services.

The OSMM is designed to be a lightweight process which can evaluate an open-source product's maturity in two weeks or less (Golden, 2004).

Software Product Maturity Model

In addition to the OSMM, Nastro (1997) developed a maturity model for the software product. His maturity model consists of three core elements and two sub-elements, the sub-elements may be applied to specific software applications (Nastro, 1997).

The core elements of Nastro's (1997) model are the following:

1. Product capability.
2. Product stability.
3. Product maintainability

And the sub-elements are:

1. Product repeatability.
2. Product compatibility.

Based on the computed maturity level of each of the core and sub-elements, Nastro (1997) proposed the following equation to calculate the product maturity level of an embedded, real-time or signal processing system (Nastro, 1997):

$$PC * (PS + PR + PM) / 3 \quad (1)$$

where:

- *PC* is the Product Capability maturity level,
- *PS* is the Product Stability maturity level,
- *PR* is the Product Repeatability maturity level, and
- *PM* is the Product Maintainability maturity level.

In the above equation, *PC* has the highest weight among all the core and sub-elements, because of its criticality for this application (Nastro, 1997).

3.2 SCOPE Maturity Model (SMM)

This SMM was created by members of the EuroScope consortium, a network of European evaluators established in 1993. The consortium comprises member organizations from Denmark, France, Hungary, Ireland, Italy, the Netherlands, Spain, and the United Kingdom. The consortium has explored the potential for software product certification. During these discussions, it became evident that the software market—comprising producers and acquirers/users—often fails to recognize the benefits of certification. Customers of certification services frequently lack clarity about what to expect, as well as how certification relates to testing or measuring software. To address this issue, the SMM was developed to contextualize software product evaluations.

Additionally, a certification service named SCOPEmark Level-2 has been established. This service, aligned with Level 2 of the SMM, is set to launch in April 1999 (Jakobsen & Punter, 1999).

The objective of the SCOPE Maturity Model (SMM) is to enable evaluation customers as well as evaluating bodies to use a framework for the design and execution of an appropriate software product evaluation (Jakobsen & Punter, 1999).

when a software product is assessed as a Level n product in the maturity model, it indicates that the product meets the requirements not only for Level n but also for all preceding levels, such as Level n-1. Each level is defined by a set of requirements for the evaluation process and its outcomes. Below, we outline the core concepts behind each of the 5 levels. While the lower levels are grounded in international standards, such standards are largely absent for the higher and more advanced levels of software quality evaluation (Jakobsen & Punter, 1999):

1. SMM-1: The initial level serves as the default level, meaning there are no explicit requirements regarding the product's quality. However, if a product is assessed as a Level 1 product, the implicit requirement is that it has undergone evaluation. This makes the initial level an awareness stage within the model (Jakobsen & Punter, 1999).
2. SMM-2: the repeatable level, Repeatable product quality ensures that the product meets expectations outlined in basic requirements, which are confirmed through testing with satisfactory results. The key component of Level 2 is the ISO 12119 standard, which provides users with confidence that the product performs as promised and documented (Jakobsen & Punter, 1999).
3. SMM-3: the defined level, at this level, software product quality is evaluated as quality in use (Bevan, 1997), rather than solely based on compliance with specifications or requirements. Usage scenarios are systematically analyzed, and the insights gained guide efforts to improve product quality. The key standard at this level is ISO 9126, which defines relevant quality attributes for the product, including functionality, reliability, usability, efficiency, maintainability, and portability (Jakobsen & Punter, 1999).
4. SMM-4: the managed level, at this level, the holistic approach to product quality is further developed by incorporating quantitative measures and risk analysis. Metrics are extensively collected to guide the planning and execution of evaluations and to assess both internal and external software attributes, using parts 2 and 3 of ISO/IEC 9126 (Jakobsen & Punter, 1999).
5. SMM-5: the optimising level, at this level, the evaluation process is continuously optimized to enhance software product quality. This involves implementing feedback mechanisms based on quantitative measures to refine the evaluation process and using evaluation results to drive product improvements. Unlike Level 4, where problem reports and coverage metrics are collected, Level 5 takes a proactive approach by leveraging these insights to guide further evaluations. The outcomes are then applied to improve the quality of the software product (Jakobsen & Punter, 1999).

3.3 Software Product Quality Maturity Model (SPQMM)

The SPQMM comprises three sub-models designed to assess and enhance quality not only after software delivery but throughout its entire lifecycle, that is, Software Product Internal Quality Maturity Model (SPIQMM), Software Product External Quality Maturity Model (SPEQMM), and Software Product Quality-in-Use Maturity Model (SPQiUMM). However, each of these maturity sub-models is composed of several sub-models, each based on the ISO 9126 quality characteristics and corresponding product quality measures (Al-Qutaish & Abran, 2011).

To implement the quantitative approach, the six-sigma methodology for assessing software product quality has been applied in the development of this quality maturity models (Al-Qutaish & Abran, 2011). However, the architecture of the proposed maturity model is grounded in two widely accepted concepts in the industry (Al-Qutaish & Abran, 2011):

- Levels of product quality
- A quantitative approach to product quality

Based on observations of general industry practices beyond the software field, the following five quality maturity levels have been identified for the SPQMM (Al-Qutaish & Abran, 2011):

1. Guaranteed
2. Certified
3. Neutral
4. Dissatisfied
5. Completely Dissatisfied

Sigma values are used to determine the quality maturity levels in the SPQMM, as shown in Figure 9. This maturity scale can be applied from two perspectives: it assesses not only the overall quality of the software product but also the quality at different stages of the lifecycle, including internal quality, external quality, and quality-in-use (Al-Qutaish & Abran, 2011).

FIGURE 9: The quality maturity levels and their scales for the Software Product Quality Maturity Model (SPQMM) (Al-Qutaish & Abran, 2011).

The SPQMM can be used to assess the maturity of a software product's quality. Specifically, it can be applied to (Al-Qutaish & Abran, 2011):

- Certify the quality maturity level of a new software product, potentially aiding its market promotion.
- Benchmark existing software products to help in selecting the best one based on its quality maturity level.
- Evaluate the quality of a software product throughout its development lifecycle (internally, externally, and in-use), exploring the relationships between these stages and identifying weaknesses to improve the product.
- Assess the maturity of the internal quality of a software product for reuse in other software products.
- Compare the maturity levels of quality at different lifecycle stages (i.e., internal, external, and in-use).

4. SOFTWARE PROCESS MATURITY MODELS LIMITATIONS

4.1 Capability Maturity Model Integration for Software Engineering – CMMI-SW

The Capability Maturity Model Integration (CMMI) for Software Engineering is a widely used framework for improving and assessing software development processes. However, like any methodology, it has several limitations:

1- Complexity and Cost of Implementation: Implementing CMMI can be resource-intensive, requiring significant time, effort, and financial investment, especially for small to medium-sized organizations. Also, the process involves extensive documentation, training, and changes to existing processes, which can be costly.

2- Focus on Process, Not Product Quality: CMMI primarily emphasizes process improvement rather than directly addressing product quality. While better processes can lead to improved

quality, it does not guarantee that the final product will meet customer needs or quality expectations.

3- Heavy Emphasis on Documentation: CMMI requires a significant amount of documentation, which may be seen as burdensome, especially for organizations that prefer lighter, more agile approaches to software development. This documentation-focused approach may slow down development and lead to inefficiencies.
4- Not Well-Suited for Agile Methodologies: CMMI's structured approach to process improvement can conflict with agile methodologies, which emphasize flexibility, iterative development, and minimal documentation. While adaptations of CMMI for agile environments exist (e.g., CMMI for Development), the integration can be challenging and may not fully align with the agile philosophy.
5- Rigidity and Overhead: Organizations may become overly focused on achieving specific maturity levels, leading to rigidity in processes. This can stifle innovation and flexibility within teams, especially if the focus is placed more on meeting CMMI requirements than on delivering value to customers.
6- Difficult to Scale for Smaller Organizations: CMMI can be too complex for smaller organizations with limited resources. The full implementation of CMMI may not be practical for companies with smaller teams or those in early stages of growth.
7- Long-Term Commitment: Achieving higher maturity levels in CMMI is a long-term commitment that requires continuous assessment, improvement, and maintenance. This ongoing process may be unsustainable for organizations that do not have long-term dedication to process improvement.
8- Potential for Over-Standardization: While standardization can improve consistency, over-reliance on CMMI's prescribed processes might lead to a lack of flexibility and creativity in the development process, which can hinder the ability to adapt to changing market or customer needs.
9- Limited Focus on Team Culture: CMMI focuses on process improvement but does not directly address the culture of collaboration, communication, or employee engagement, which are critical for successful software engineering practices.
10- Overemphasis on Compliance: The model's focus on process compliance can sometimes create a "checklist" mentality, where organizations focus on meeting the minimum requirements for certification rather than fostering genuine improvement or innovation in their software development practices.

4.2 Testing Maturity Model – TMM

TMM offers valuable guidance for enhancing testing practices, there are several limitations to its application:

1. Focus on Process, Not Product Quality: TMM emphasizes improving the testing process but does not directly address the final product's quality. It may improve testing efficiency, but it does not guarantee that the software will meet customer requirements or expectations unless paired with other quality-focused practices.
2. Resource-Intensive: Implementing the TMM often requires significant investments in time, effort, and resources, particularly for organizations with limited testing expertise or resources. Achieving higher maturity levels might necessitate hiring specialized personnel, acquiring new tools, and undergoing extensive training.
3. Long Implementation Time: Advancing through the maturity levels of TMM can be a slow and incremental process. Organizations may struggle to realize tangible results quickly, especially if they are starting at a low maturity level and need to make substantial changes to their testing processes.
4. Rigidity and Over-Standardization: TMM can lead to overly standardized testing practices that may not be flexible enough to meet the unique needs of different projects or industries. This can stifle innovation and may not accommodate more dynamic, agile development environments.
5. Limited Alignment with Agile Practices: TMM was originally designed with traditional, waterfall-based development in mind and may not align well with agile methodologies. Agile

teams value adaptability, fast feedback loops, and minimal documentation, which can conflict with TMM's more structured and process-oriented approach.
6. Heavy Emphasis on Documentation: Achieving higher maturity levels in TMM often requires extensive documentation to demonstrate compliance with the model's requirements. This can be burdensome, particularly for teams or organizations that prefer lean, agile approaches with less documentation overhead.
7. Not Universally Applicable: The model may not fit well with all organizations or industries. Organizations with specific, niche needs may find TMM's general framework too rigid or overly focused on areas that are not critical to their success.
8. Measurement Challenges: Tracking progress and measuring maturity levels within TMM can be challenging. The model relies on subjective assessments, and the criteria for evaluating improvements may not always be clear, making it difficult to gauge progress consistently.
9. Risk of Becoming Focused on Certification: Organizations may become overly focused on achieving high maturity levels for the sake of certification, rather than genuinely improving their testing processes. This could lead to a "tick-box" mentality where the focus shifts from meaningful improvements to simply meeting the model's requirements.
10. Limited Focus on Team Collaboration: TMM focuses on improving the testing process but doesn't address the collaborative and cultural aspects of testing. Effective testing often depends on teamwork, communication, and the involvement of various stakeholders, which may not be sufficiently emphasized in the model.
11. Inability to Address Fast-Paced Technological Changes: The rapid evolution of testing tools and methodologies, such as automated testing, continuous integration, and AI-driven testing, may outpace the TMM framework, making it harder for organizations to keep the model up-to-date with current trends in testing.

4.3 ISO Process Assessment Model – ISO 15504

The ISO Process Assessment Model (ISO 15504), provides a framework for assessing and improving software processes. While it is widely used and respected, there are several limitations associated with its implementation:

1. Complexity and Resource Intensity: Implementing ISO 15504 can be resource-intensive, requiring significant time, effort, and financial investment. The process assessment, documentation, and certification can be burdensome, particularly for smaller organizations with limited resources. Achieving high capability levels often requires dedicated teams, specialized tools, and extensive training.
2. Long Time to Achieve Results: Progressing through the levels of ISO 15504 can be a slow process. Moving from one capability level to the next involves continuous improvement and may take months or years, which can be discouraging for organizations looking for quicker results.
1. For organizations starting at a low maturity level, the improvement process can be lengthy and require significant cultural and structural changes.
2. Focus on Process, Not Product Quality: Like other maturity models (e.g., CMMI), ISO 15504 focuses heavily on process improvement rather than directly on product quality. While improved processes can lead to better quality, the model does not directly guarantee that the end product will meet customer expectations or requirements.
3. Limited Flexibility for Agile or Rapid Development: ISO 15504 is based on traditional, structured approaches to software development, which may not align well with agile or iterative methodologies. Agile organizations that value flexibility, rapid iterations, and minimal documentation may find the framework too rigid for their needs. While ISO 15504 can be adapted to agile environments, the model is inherently more suited for structured, waterfall-based projects, which can cause challenges for organizations adopting agile practices.
4. Heavy Emphasis on Documentation and Formality: ISO 15504 requires a significant amount of documentation and formal assessments to demonstrate compliance with the process maturity levels. This can create administrative overhead, especially for organizations that prefer lighter, more agile approaches with less emphasis on documentation. The focus on

documentation can lead to a "checklist" mentality, where organizations focus on meeting formal requirements rather than addressing more practical process improvements.
5. Difficulty in Adapting to Specific Organizational Needs: While ISO 15504 is a general framework for process assessment, it may not fully account for the unique needs, constraints, or objectives of individual organizations. It can be too generic and may require significant customization to align with the specific goals of an organization or the nature of its projects.
6. Overemphasis on Compliance: Organizations may focus too heavily on achieving higher capability levels for the sake of certification, rather than focusing on continuous process improvement. This can lead to a superficial or compliance-driven approach, where the goal becomes ticking boxes to meet the standard rather than genuinely improving processes.
7. Resource Constraints for Small Organizations: For small to medium-sized organizations, the resources required to implement ISO 15504 effectively (such as dedicated personnel, tools, and training) may not be available. The framework is often more suited to larger organizations with established process improvement teams, making it harder for smaller companies to adopt without significant investment.
8. Lack of Focus on Team Collaboration and Culture: While ISO 15504 focuses on process improvement, it places limited emphasis on the cultural and collaborative aspects of process improvement. Successful implementation of software processes often relies on team dynamics, communication, and buy-in from stakeholders, but these factors are not directly addressed in the framework.
9. Limited Guidance on Technological Advancements: ISO 15504 primarily focuses on traditional software engineering processes and may not be fully applicable to organizations adopting cutting-edge technologies, such as AI, machine learning, or advanced automation. The framework may need to be updated to accommodate newer developments in the software engineering field.
10. Difficulty in Measuring Non-Process Related Factors: ISO 15504 emphasizes process capability but does not provide direct methods for assessing non-process-related factors like product innovation, team morale, or customer satisfaction. These factors are crucial to overall project success but are not part of the model's direct scope.
11. Challenges in Scaling for Different Industry Types: While ISO 15504 provides a general model, it may not fully account for the specific needs of different industries, such as embedded systems, safety-critical applications, or high-performance computing. Customization is often required for specialized environments.

5. SOFTWARE PRODUCT MATURITY MODELS LIMITATIONS

5.1 Open-Source Maturity Model (OSMM)

While it is useful for assessing the health and progression of open-source projects, there are several limitations to using an Open-Source Maturity Model:

1. Subjectivity and Ambiguity in Assessment: Open-source projects vary widely in terms of goals, community engagement, and technical focus. The subjective nature of certain maturity criteria, such as "community involvement" or "transparency," can lead to inconsistent evaluations. Different evaluators might interpret the same criteria differently, which may skew the results or create ambiguity in maturity levels.
2. Lack of Industry Standardization: The Open-Source Maturity Model may not be universally adopted or standardized across different industries or organizations. This means that while the model can offer guidance, its application and the interpretation of its results may vary, reducing its consistency and reliability.
3. Focus on Process Over Results: Many open-source maturity models place a strong emphasis on the processes, governance structures, and technical standards of a project. While these are important for long-term sustainability, the focus on process can sometimes overshadow the actual impact or outcomes of the project, such as the quality of the software or its adoption in the community.

4. Overemphasis on Formality: Some maturity models may encourage projects to adopt formal processes, policies, and structures that may not be appropriate for every open-source initiative. Smaller projects or those with fewer resources may struggle with implementing the more formalized structures suggested by maturity models, which could inhibit their growth or innovation.
5. Limited Consideration of External Factors: Open-source projects exist in a dynamic and often volatile environment where factors such as market demand, competition, and changes in licensing or technology can influence success. Maturity models typically focus on internal processes and community health, but they may not fully account for these external factors that can significantly impact a project's sustainability and relevance.
6. Inflexibility for Diverse Open-Source Models: Open-source projects are highly diverse, with some focused on rapid prototyping and others on long-term stability. The linear or hierarchical stages often present in maturity models may not suit all types of open-source projects. The model may assume that every project must follow the same path to maturity, which doesn't work for projects that take different approaches to governance, code contribution, or community building.
7. Difficulty in Quantitative Measurement: Many of the factors assessed in Open-Source Maturity Models - such as community engagement, code quality, and collaboration - are difficult to measure quantitatively. Metrics can be subjective, and there may not be a consistent or reliable way to evaluate progress across different projects. This can lead to challenges when trying to compare projects or determine objective progress over time.
8. Inadequate Reflection of Innovation: Open-source projects are often at the forefront of innovation, with rapidly changing ideas and technology. Maturity models, which typically focus on established practices and processes, might not adequately reflect the innovative nature of a project. Projects in the early stages of development or those experimenting with new technologies may not score well on traditional maturity metrics, even though they may be highly influential or revolutionary in their field.
9. Resource Constraints: Implementing a maturity model, including self-assessment and continuous improvement based on the model, can be resource-intensive. Smaller open-source projects may not have the time or financial resources to devote to regular evaluations, which can limit the model’s usefulness for these projects.
10. Potential for Stagnation: Some projects might become overly focused on achieving higher maturity levels according to the model, rather than prioritizing meaningful improvements or innovations. This can lead to a "checklist mentality," where projects focus on ticking off items to "progress" in the maturity model, rather than addressing real-world needs or staying responsive to their communities.
11. Misalignment with Project Goals: Open-source projects often have diverse goals, such as experimentation, community building, or providing software for a niche need. The OSMM might not account for projects that intentionally choose a less formal or more fluid development model. In these cases, trying to apply the maturity model's criteria could force the project into a framework that doesn’t align with its objectives or values.
12. Varying Project Lifecycles: Open-source projects often have different life cycles: some are long-term, sustained efforts, while others may be short-term, experimental, or only intended for a specific purpose. The maturity model may not accommodate the natural evolution of different types of projects, treating all open-source initiatives as if they should progress through the same stages of maturity.

5.2 Software Product Maturity Model

The second product maturity model – the Nastro software product maturity model – has the following limitations:

1. It is for an executable software product. Therefore, it can only be used with an incremental life-cycle which provides multiple releases (versions) of an executable software product.
2. It is not based on any comprehensive quality model, but only on a small number of product quality characteristics (there are five of them).
3. It is designed for the software product itself, rather than the quality of the software product.

4. For each element (core or sub), there is only one measure.
5. It has been built to track and report the software development effort during an incremental life-cycle.

5.3 SCOPE Maturity Model (SMM)

There are several limitations to the SMM that can affect its effectiveness and the accuracy of its assessments:

1. Over-Simplification of Complex Processes: Maturity models, including SMM, tend to break down complex organizational processes into discrete stages or levels. This simplification may overlook nuanced challenges or unique situations that don't fit neatly into the model's predefined stages.
2. Lack of Contextual Flexibility: SMM might not account for all the contextual factors that influence an organization's practices. For example, different industries or regions may have unique challenges that the model doesn't adequately address, leading to potentially skewed assessments.
3. Dependence on Self-Assessment: Often, maturity models rely on self-assessments, which can be biased. Organizations may rate themselves higher than they deserve, leading to an inaccurate representation of their actual maturity level. This can result in misguided improvement initiatives.
4. Rigid Structure: The linear nature of maturity models may lead to a one-size-fits-all approach. Some organizations may require custom or more flexible frameworks for growth, especially in fast-moving or innovative sectors where adaptability and agility are key.
5. Focus on Quantitative Metrics: SMM and similar models often emphasize measurable metrics and structured processes. While this is useful for tracking progress, it may miss important qualitative factors such as organizational culture, employee engagement, or innovation, which also contribute significantly to overall maturity.
6. Potential Resistance to Change: The structured stages in SMM can sometimes be seen as a rigid roadmap for improvement. This may cause organizations to become resistant to change or overly focused on moving through stages, rather than on continuous improvement.
7. Resource Intensive: Implementing and tracking improvements based on the maturity model may require significant resources, time, and commitment, which can be a barrier for smaller organizations or those with limited capacity for large-scale transformation.
8. Overemphasis on "Best Practices": Maturity models often emphasize "best practices," which can lead organizations to focus too much on established methods rather than exploring innovative approaches. This focus might hinder creativity or delay adoption of newer, more effective practices.
9. Limited Focus on External Factors: External factors like market changes, regulatory shifts, and competitor behavior might not be sufficiently addressed by the SMM, limiting its ability to provide a comprehensive view of an organization's position within its broader environment.
10. Lack of Real-Time Feedback: Many maturity models, including SMM, might not provide real-time feedback or adaptability to ongoing changes. This can make it difficult for organizations to adjust their strategy based on rapidly evolving business needs.

5.4 Software Product Quality Maturity Model (SPQMM)

The PQMM is limited to the ISO 9126 quality model and its associated measures. Another limitation is that the results of these models assume equal weight for all measures, characteristics, and sub-characteristics. To overcome this assumption, organizations can apply relevant statistical techniques or assign their own weights using methods such as the Analytical Hierarchy Process (AHP) (KOSCIANSKI & COSTA, 1999). Subsequently, they can use appropriate techniques to combine these weights into aggregated values at higher levels, similar to approaches used in the QEST multi-dimensional models for quality and performance (BUGLIONE & ABRAN, 1999, 2002).

6. CONCLUSIONS

In conclusion, maturity models play a pivotal role in the field of software engineering by offering a structured framework for assessing and enhancing both development processes and software products. They provide organizations with a clear benchmark for evaluating their current capabilities, identifying areas for improvement, and guiding them toward higher levels of efficiency, quality, and innovation. Whether focusing on the optimization of development processes or the refinement of product quality, maturity models serve as essential tools for continuous improvement and standardization within the industry. By enabling organizations to measure their performance against industry best practices, these models foster a culture of growth, ensuring that software development is not only effective but also adaptive to evolving technological demands. As this paper explores various maturity models, their application, and their impact, it becomes evident that these models are invaluable assets for organizations seeking to achieve long-term success and excellence in software engineering.

In this paper, three software process maturity models and four software product maturity models have been analysed and discussed. However, a number of limitations for each of the discussed maturity models has been identified to be used as a guidance for any future research work to enhance such maturity models.

7. REFERENCES

Al-Qutaish, R. E., & Abran, A. (2011). A Maturity Model of Software Product Quality. *Journal of Research and Practice in Information Technology 43*(4), 307-327.

Bruin, T. d., Rosemann, M., Freeze, R., & Kulkarni, U. (2005). *Understanding the main phases of developing a maturity assessment model* Australasian Conference on Information Systems (ACIS'05), Sydney, Australia.

BUGLIONE, L., & ABRAN, A. (1999). Geometrical and statistical foundations of a three-dimensional model of software performance. *Advances in Engineering Software*, *30*(12), 913–919.

BUGLIONE, L., & ABRAN, A. (2002). QEST nD: n-dimensional extension and generalisation of a software performance measurement model. *Advances in Engineering Software*, *33*(1), 1-7.

Burnstein, I., Suwanassart, T., & Carlson, C. R. (1996a). Developing a Testing Maturity Model: Part I. *CrossTalk: the Journal of Defense Software Engineering*, *9*(8), 21-24. http://www.stsc.hill.af.mil/crosstalk/frames.asp?uri=1996/08/Developi.asp.

Burnstein, I., Suwanassart, T., & Carlson, C. R. (1996b). Developing a Testing Maturity Model: Part II. *CrossTalk: the Journal of Defense Software Engineering*, *9*(9), 19-26. http://www.stsc.hill.af.mil/crosstalk/frames.asp?uri=1996/09/Developi.asp.

Golden, B. (2004). *Succeeding with Open Source*. Addison-Wesley Professional.

ISO. (2003). *Information Technology - Process assessment - Part 2: Performing an Assessment*. International Organization for Standardization.

ISO. (2004a). *Information Technology - Process Assessment - Part 1: Concepts and Vocabulary*. International Organization for Standardization.

ISO. (2004b). *Information Technology - Process Assessment - Part 3: Guidance on Performing an Assessment*. International Organization for Standardization.

ISO. (2004c). *Information Technology - Process Assessment - Part 4: Guidance on Use for Process Improvement and Process Capability Determination*. International Organization for Standardization.

ISO. (2006). *Information Technology - Process Assessment - Part 5: An Exemplar Process Assessment Model, Document Number: N3302 Dated on 14 September 2005*. International Organization for Standardization.

Iversen, J., Nielsen, P. A., & Norbjerg, J. (1999). Situated assessment of problems in software development. *Database for Advances in Information Systems*, *30*(2), 66-81.

Jakobsen, A., & Punter, T. (1999). *Towards a Maturity Model for Software Product Evaluations*. Retrieved Accessed on Nov. 25, 2024 from https://www.researchgate.net/publication/2504368_Towards_a_Maturity_Model_for_Software_Product_Evaluations

KOSCIANSKI, A., & COSTA, J. C. B. (1999). Combining analytical hierarchical analysis with ISO/IEC 9126 for a complete quality evaluation framework. 4th IEEE International Symposium and Forum on Software Engineering Standards, Curitiba, Brazil.

Maturity Model: A framework that assesses the level of maturity of an organization's processes and practices. Retrieved Visited on Nov. 25, 2024 from https://www.brightwork.com/glossary/maturity-model.

McBride, T., Henderson-Sellers, B., & Zowghi, D. (2004). Project Management Capability Levels: An Empirical Study. In *Proceedings of the 11th Asia-Pacific Software Engineering Conference - APSEC'2004* (pp. 56-63). IEEE Computer Society Press.

Nastro, J. (1997). A Software Product Maturity Model. *CrossTalk: the Journal of Defense Software Engineering*, *10*(8). http://www.stsc.hill.af.mil/crosstalk/1997/08/product.asp.

SEI. (1993). *Capability Maturity Model for Software Engineering (Version 1.1)*.

SEI. (2002a). *Capability Maturity Model Integration for Software Engineering (CMMI-SW) - Staged Representation, Version 1.1*. http://www.sei.cmu.edu/pub/documents/02.reports/pdf/02tr029.pdf.

SEI. (2002b). *Capability Maturity Model Integration for Software Engineering (CMMI-SW) - Staged Representation, Version 1.1*. http://www.sei.cmu.edu/pub/documents/02.reports/pdf/02tr029.pdf.

Torriano, K. (2022, 15-18 May). *Maturity Models: Testing Your Way to the Top* STC'22 Summit: Technical Communication Conference & Expo, Chicago (Rosemont), IL, USA.

Weber, M. (2008). The business case for corporate social responsibility: A company-level measurement approach for CSR. *European Management Journal 26*(4), 247-261.

Wendler, R. (2012). The maturity of maturity model research: A systematic mapping study. *Information and Software Technology*, *54*(12), 1317-1339.