ßMACH — A Software Management Guidance

Marcus Hilbrich[1], Fabian Lehmann[1]

[1]Humboldt-Universität zu Berlin, Unter den Linden 6, 10099 Berlin, Germany

Abstract

Creating, maintaining, and operating software artifacts is a long ongoing challenge. Various management strategies have been developed and are frequently used. Nevertheless, a unification of describing the management strategies to compare them is an open question. We present ßMACH as an answer. ßMACH allows systematic descriptions and checks independently from the management strategy. In this paper, we test parts of ßMACH on the example of performance requirements. So we applied ßMACH to V-Model and Scrum.

Keywords

ßMACH, software management, software process model, software artifact, software engineering

1. Introduction

From a software engineer's perspective, software artifacts like source code, executable, and documentation need an active management. How to achieve this is an open, but frequently discussed question [1, 2].

Software process models are commonly used to manage software artifacts, as well as the software life cycle [3, 4]. Well-known examples for software process models are the V-model [5, 6] and Scrum [7]. Thus, the need to manage software artifacts is known for a long time, and different strategies have evolved [8, 9]. Never the less much is unclear:

1) The software life cycle demands that software artifacts are, e.g., changed, and maintained. The V-model aims to deliver the software to the buyer, so it does not include maintenance or later changes. Scrum describes the management of a software project. A project is limited in time, so ongoing maintenance or changing the software is not directly covered. Some DevOps [10, 11] strategies use Scrum like proceedings to overcome these limitations, so it seems to be an open demand. As a result, it is unclear which phases of the software life cycle need to be covered or when the management of software artifacts should be started or ended.

2) The demanded performance of a system or a component should be adequate to use the system [9]. To use the V-model, we have to specify the performance before starting the software development. Therefore, we use, e.g., requirement engineering to find out the needed performance and a usable description. We can test the demands later on. In

SSP'21: Symposium on Software Performance, November 09–10, 2021, Leipzig, Germany
marcus.hilbrich@informatik.hu-berlin.de (M. Hilbrich); fabian.lehmann@informatik.hu-berlin.de (F. Lehmann)
0000-0003-3717-9449 (M. Hilbrich); 0000-0003-0520-0792 (F. Lehmann)

Scrum, we define the system by user stories (in the backlog). However, an office user will not specify the time he can wait for a request to finish, the acceptable probability of requests taking longer, and the hardware infrastructure a cloud operator has to provide. Accordingly, it is uncommon to have a performance definition in the backlog. So, the performance needs to be estimated during the Scrum process, discussed with the buyer, or other roles. As a result, it is not clear how to describe the product or its performance.

Luckily, various ideas on how to manage software artifacts exist. While V-model, Scrum, and Software life cycle are just basic examples, change management [12, 13], microservices [14], usage of best practices as given by Martin [15], or the use of results from Human Resource Management [16] are additional and well-known strategies. However, it is unclear how to describe the management in a uniform language, compare management proceedings based on standards, or survey a management method.

We propose the Systematic Software Management Approaches Characterization Helper (ßMACH), it is at least a partial answer to the ongoing challenges of software artifacts' management. ßMACH groups key aspects for software management based on an ontology to describe a software management proceeding systematically. Thereby, ßMACH defines a uniform description/language and starting point for future comparisons and analysis of management strategies. This paper uses (parts of) ßMACH to look into performance management exemplarily for the V-model and Scrum. We describe the filling of the ßMACH protocol to introduce it and better understand the software process models. This paper serve as an early discussion point on ßMACH and its capabilities.

2. The ßMACH Protocol

ßMACH is a guidance protocol throughout the software development process or other software artifact management strategies. It offers the key aspects of management in a protocol and connects them. Therefore, ßMACH uses specific wording-terms mostly based on knowledge management to bring different management approaches to a common denominator. This enables the ßMACH method to work with various software development frameworks, software process models, agile management strategies, artifact based proceedings, etc. In particular, the ßMACH protocol consists of three parts:

1) A section of metadata to describe and identify the management proceeding based on the team involved, the product, the company, date of filling the protocol, etc.

2) A short description of the management framework used and how it works. Probably by the tailoring of well-known strategies.

3) A table of aspects that have to be handled by any proceeding filled with answers. The ßMACH protocol guides the filling person through asking different predefined questions concerning all possible software artifacts. Thereby, the ßMACH protocol ensures that all key aspects of the software artifact management process are considered.

In this paper, we take a deeper look at the abilities of the ßMACH protocol to describe and explain performance characteristics, requirements, and solutions in a software development process. To underline the generality of ßMACH, we look at Scrum as a representative for agile and the V-model for non-agile development processes. Accordingly,

we only discuss a small subset of the ßMACH protocol focusing on performance.

An excerpt of the ßMACH protocol is depicted in Fig. 1. The protocol itself does not prescribe the table's filling order. In the following, we discuss four of the five columns of ßMACH, to distinguish different origins of knowledge.

In general, the ßMACH method gives five columns to describe all key aspects within a software development process, where it is mandatory to fill all cells. We start with a discussion of Scrum, in concrete we start with a description of artifacts, this knowledge is given by ßMACH in the row Product Properties (see Fig. 1). The first column is Product Knowledge. It contains all knowledge that is needed to describe software artifacts. E.g., for Scrum, this is the Sprint Backlog, as knowledge of the Sprint Backlog is enough to complete a Sprint and get the software artifact in a more complete state.

The next column in ßMACH is Demanded Knowledge. As Sprint Backlogs are not initially available, they are Demanded Knowledge by ßMACH's definition, which indicates that this knowledge must be defined over time.

Next, ßMACH offers a column for the Roles. Here, all roles are named that provide knowledge to a process of the software artifact lifecycle. Within the development process, these are the developers, as they know how to extend the product.

Furthermore, ßMACH names the Process Knowledge. This knowledge comprises all information to perform the process that realizes the product properties.

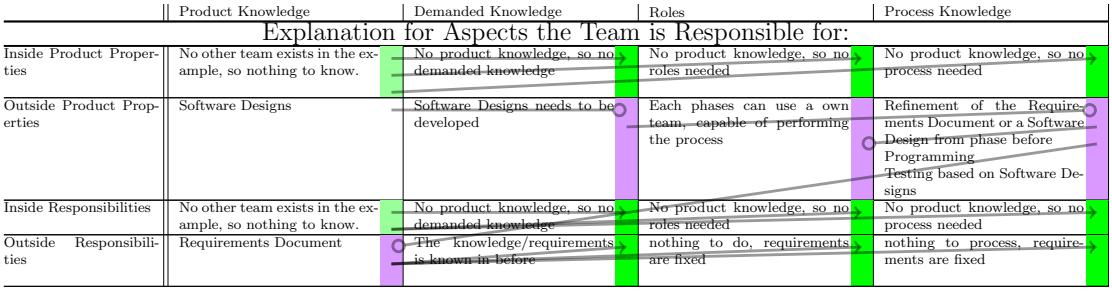
For example, in Scrum, we have to define where the Sprint Backlog comes from. It is developed based on the Product Backlog, thus it is based on the Product Knowledge, which we have to describe, too. The product backlog is not directly a description of the product (even if used to create the Sprint Backlog). Instead, it is a list of items we have confirmed to realize. Thus, it is a responsibility in terms of ßMACH. The Product Backlog is not a stable artifact. It is developed by the complete Scrum team based on discussions – this gives the Demanded Knowledge, the Roles, and the Process.

The missing link is how to get from the Product Backlog to the Sprint Backlog. The Scrum team creates the Sprint Backlog in a discussion, where it defines Processes, Roles, Product Knowledge, and Demanded Knowledge.

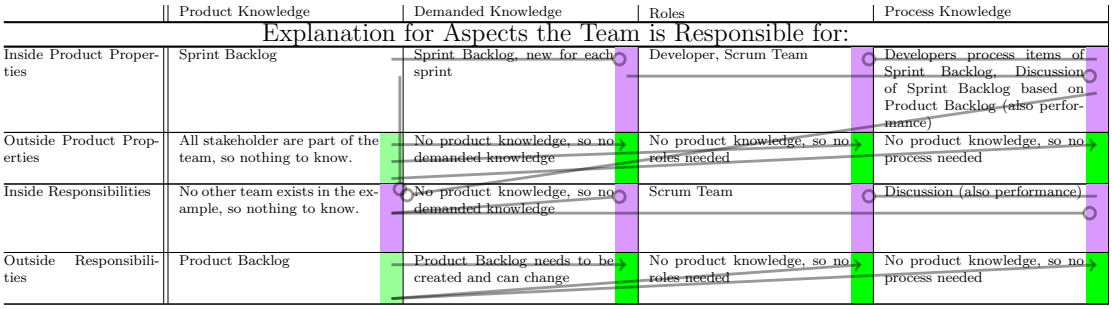
In the following, we describe performance in the ßMACH protocol: The column where performance is defined varies with the paradigm used. Since performance definition is part of the discussion in Scrum, it is defined as an important aspect of an item in the Sprint Backlog, or it can be a more general demand as an item of the Product Backlog. Accordingly, performance is Product Knowledge but can be Demanded Knowledge, too.

Contrary to Scrum, the V-model immediately starts with a Requirements Document. Based on the document, Software Designs are created (by different phases of the V-model), and the last Software Design is fine enough to start programming. Tests are defined parallelly based on Software Designs and Requirements. Moreover, a team is responsible for fulfilling the requirements in the Requirements Document.

The V-model's requirements, and thus the performance, are initially defined and cannot be changed over time. So, this is the definition of the product to deliver. Accordingly, it is Product Knowledge of the Outside Responsibilities. As the document is present initially, no Demanded Knowledge is required for Product Properties. The respected knowledge is addressed in the rows for Product Properties in the table of ßMACH. The



(a) Description of the V-model.



(b) Description of Scrum.

Figure 1: Given are descriptions of software process models based on a subset of the ßMACH protocol. ßMACH defines a set of key aspects to describe. Each cell of the table represents an aspect. ßMACH defines a coloring. Based on the management proceeding, light green is used for aspects that do not need active management (the aspect is realized without a need for actions). Darker green indicates an aspect that is provided by another aspect without a need for active management. Violet is used for aspects that are used or require by additional aspects. Such an aspect is likely of special interest. Arrows with a peak-end describe based on which other aspect an aspect is provided. An arrow with a round-end gives a demand relation. The other aspect needs the aspect at the round-end. The aspects in the tables are described in the paper, the filling of the aspects, too.

protocol contains these rows for Inside and Outside Product Properties, where Inside means a joined work of teams on the same project, while outside describes a product dealing with an external party. The requirements are then tested by the buyer, so we do not need to process the responsibilities and do not need a special role to do so.

The design phases of the V-model produce Software Designs, and these are Product Knowledge of the Product Properties. It needs to be created by performing the V-model, so it is also Demanded Knowledge based on a refinement of the Requirements Document or a Software Design (see Fig. 1).

3. Observations

Due to the focus on performance, we only had a minimal look at ßMACH in the last section. Thus, many aspects of ßMACH are ignored, like interfaces, dependencies, information recording, maintenance, and product improvement. Nevertheless, we got a comparison of Scrum and the V-model, and we can obtain the following observations:

First, the V-model defines performance based on the initial requirements document that is not changed later on. In Fig. 1a the requirements document is a direct or indirect knowledge source for many aspects/cells in ßMACH. As a result, refining the design process, programming, and testing has to consider the performance-based requirements. Contrary, Scrum discusses the performance aspects within the Scrum team, including, e.g., buyers, and users. The product backlog is demanded by a process in Fig. 1b.

Second, different rows are filled within the ßMACH protocol to describe the management (see Fig. 1a and 1b). While the V-model uses the rows marked as outside, Scrum uses the rows marked as inside. This is a direct representation of different management philosophies. Scrum includes buyers, users, etc., while the V-model does not.

Third, while the Scrum process (Process Knowledge) is mainly based on discussions, the V-model relies on refinement and testing. Despite these differences, both software process models provide a set of artifacts to represent the product knowledge. While this was expected for an artifact-based software process model like the V-model, it also applies to Scrum using the Backlogs. However, Scrum has less of such artifacts, but the artifacts seem to be an essential part.

Forth, we can identify shortcomings of the management strategies or the representation of the strategies in this paper. The principle description of the V-model does not give a detailed description of the roles and the proceeding in the phases. In Scrum, processes are based on discussions, or the team knows how to do them. This describes a lack of information about the concrete process and problems of knowledge persistence, e.g., to compensate changes in the team.

The presented observations are neither new findings nor unexpected. The important aspect is, the findings are directly based on the ßMACH protocol. Thus, we can exemplify that the ßMACH protocol helps to represent and understand management strategies. Also, a comparison is enabled by the aspects/cells of the ßMACH protocol and the knowledge transformations described in Fig. 1a and 1b.

4. Conclusion

This paper gives only a coarse introduction to ßMACH. It does not yet provide findings for a customized software management process, tool-based management of performance, or a model-based development. Therefore, a detailed introduction to ßMACH and the according discussion of software management principles are in progress. A first evaluation of ßMACH is prepared, too. As an immediate step, we collect additional input.

Based on the observations we have provided, ßMACH can explain how knowledge is transformed and represented by software management strategies. Furthermore, ßMACH can clearly distinguish between different management strategies. Even if knowledge sources or representation artifacts are not described as a major part of a management process, by the idea of process-centric management, the importance of such artifacts and shortcomings in management are identified by the ßMACH protocol.

Even if the used examples are well understood and the observations are nothing new, we can state the representation of the findings by ßMACH. Furthermore, we expect

similar results for other management strategies. A supervised student thesis already indicated similar findings and lessons learned. Thus, we can state that the systematic presentation of management strategies – based on the well-defined key aspects of ßMACH that we introduced in this paper – helps to understand and compare management strategies for software artifacts. Also, ßMACH can analyze the performance management and enable future community discussions regarding the management proceeding.

References

[1] The Standish Group International, Inc., The CHAOS Report (1994), Technical Report, 1994.

[2] The Standish Group International, Inc., Chaos Report 2015, Technical Report, 2015.

[3] U.S. Department of Justice, Information Resources Management, The Department of Justice Systems Development Life Cycle Guidance Document, 2003. [Online; accessed 22-Juni-2020].

[4] G. D. Everett, J. Raymond McLeod, Software Testing; Testing Across the Entire Software Development Life Cycle, John Wiley & Sons, Ltd, 2006.

[5] F. Cechini, R. Ice, D. Binkley, Systems Engineering Guidebook for Intelligent Transportation Systems, Technical Report Version 3.0, U.S. Department of Transportation, Federal Highway Administration, California Division, 2009.

[6] 4Soft GmbH in Zusammenarbeit mit dem Informationstechnikzentrum Bund und dem Beschaffungsamt des Bundesministeriums des Innern, V-Modell XT Bund, Das Referenzmodell für Systementwicklungsprojekte in der Bundesverwaltung, version: 2.0 ed., Informationstechnikzentrum Bund im Auftrag des Beauftragten der Bundesregierung für die Informationstechnik, 2009.

[7] K. Schwaber, J. Sutherland, The Scrum Guide^{TM}, The Definitive Guide to Scrum: The Rules of the Game, https://www.scrumguides.org/, 2017.

[8] H. M. Sneed, Software Management, Rudolf Müller online DV-Praxis, Köln, 1987.

[9] I. Sommerville, Software Engineering, tenth edition ed., Pearson Education Limited, Edinburgh Gate, Harlow, Essex CM20 2JE, England, 2016.

[10] I. Buchanan, Agile and DevOps: Friends or Foes?, https://www.atlassian.com/agile/devops, 2020. [Online; accessed 11-August-2020].

[11] J. K. Waters, Scrum + DevOps = ScrumOps, https://adtmag.com/articles/2017/05/11/scrumops.aspx?, 2017. [Online; accessed 11-August-2020].

[12] D. W. Edwards, Out of the Crisis, 1986.

[13] P. Bernard, Foundations of ITIL® 2011 Edition, Van Haren, 2011.

[14] J. Lewis, M. Fowler, Microservices: a Definition of this new Architectural Term, 2014. URL: http://martinfowler.com/articles/microservices.html.

[15] R. C. Martin, Clean Code: A Handbook of Agile Software Craftsmanship, Robert C. Martin Series, Prentice Hall, Upper Saddle River, NJ, 2008.

[16] D. Keirsey, Please Understand Me II: Temperament, Character, Intelligence, Prometheus Nemesis Book Company, 1998.