Bridging the Gap between Software Platforms: A Template Method for Software Evolution

Gerard Nijboer
Department of Information and Computing Sciences
Utrecht University
Utrecht, The Netherlands
g.nijboer@students.uu.nl

Henk van der Schuur
AFAS Software
Leusden, The Netherlands
h.vdschuur@afas.nl

Jan Martijn E. M. van der Werf
Department of Information and Computing Sciences
Utrecht University
Utrecht, The Netherlands
j.m.e.m.vanderwerf@uu.nl

Sjaak Brinkkemper
Department of Information and Computing Sciences
Utrecht University
Utrecht, The Netherlands
s.brinkkemper@uu.nl

***Abstract*—To prevent issues arising from legacy software platforms, adapting to changing customer needs by software evolution is a growing concern of software organizations. However, current practices are pragmatic and subjective, which restricts benchmarking and reduces efficiency. In order to improve evolutionary practices, this paper proposes the Software Functionality Evolution Method (SFEM). The SFEM provides a software vendor with input for product roadmapping, by mapping functionality between software platforms. Mappings are based on characteristics and constraints of functionality, personas and software platforms. An incremental method engineering approach is put to practice, in which the template method is instantiated and improved over multiple cases. Cases show that the method contributes to efficient reasoning and strategic decision making in software evolution for software product managers.**

I. Introduction

The current pace of technological developments offers opportunities to software developing organizations concerning software platforms and applications. In order to avoid issues caused by legacy software platforms, concerning costs, maintenance, accessibility and extensibility [1], emerging technologies can help organizations to innovate, improve efficiencies, and realize new business opportunities [2]. Software evolution concerns the adaption of capabilities and functionality of a system, in order to meet user needs [3].

Currently, no structured approach exists which assists in the evolution of a software product by mapping functionality on new software platforms. Thus, a gap exists in the evolutionary process, as a mapping of functionality between platforms needs to be created, yet it is uncertain what functionality should be included. A structured approach enables comparison of performance and results, and increases efficiency in method instantiations.

Considering the problem statement above, the main research question of this paper is: *How could a method assist in software product evolution through mapping functionality between software platforms?*

This paper proposes a method, the Software Functionality Evolution Method (SFEM), which assists in the evolution of a software product. It is designed for software product managers, as this organizational role is responsible for strategic decision making [4], including a software product's lifecycle [5]. With this audience in mind, the evolutionary process is considered with a focus on functional, rather than technical properties and constraints. The constraints, raised by characteristics of personas, platforms and functionality, determine the mapping and mapping priority of functionality.

The SPM Competence Model [5] proposes 15 focus areas in the field of Software Product Management (SPM) practices. To position this research and the designed method in the field of SPM, Figure 1 visualizes the relationships with the different focus areas and competencies. Three categories are applied to related focus areas: (1) *triggers* which instantiate the method, (2) *execution* for the mutual support of activities, and (3) *output* for those focus areas that have an interest in the results of an instantiation.

An instantiation of the method can be triggered by activities within the focus areas *Market analysis* and *Product lifecycle management*. An opportunity can be identified, such as a new software platform, which generates a competitive advantage for the software company if implemented correctly.

The SFEM assists in the execution of activities within the focus areas *Requirements gathering*, *Requirements organizing* and *Requirements prioritization*. By mapping existing functionality between platforms, requirements are gathered and prioritized based on their added value in the market and product portfolio.

The results of an instantiation of the method provides the organization with information which can be used in activities in the focus areas *Release definition*, *Roadmap intelligence* and *Product roadmapping*. On the short term, mappings of functionality help to identify which requirements add significant value to a new release. On the long term, mappings assist in the creation of themes for the product roadmap.

This introduction is followed by an explanation of the research approach in Section II. The Software Functionality Evolution Method is presented in Section III. In Section IV, a categorization for method increments is presented, which enables reflection on the process of incremental method engineering. The results of the method instantiations in cases are presented in Section V. Section VI contextualizes the research with related literature, followed by a discussion in Section VII

978-1-4799-6358-4/14/$31.00 © 2014 IEEE

IWSPM 2014, Karlskrona, Sweden

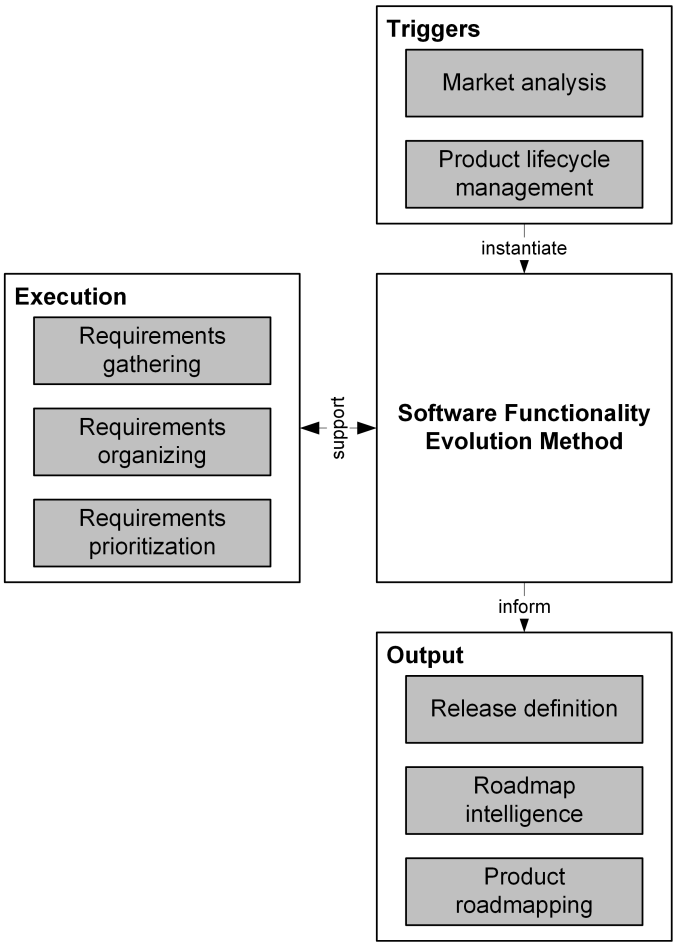

Fig. 1. Positioning of the research to the SPM Competence Model

and the conclusion in Section VIII.

II. Research Approach

The research is based on a combination of a literature review and interviews with domain experts at a Dutch Enterprise Resource Planning (ERP) software vendor. The approach followed in the literature review is inspired by the PRISMA 2009 checklist [6].

By means of method engineering [7]–[9], the initial research results in a conceptual, initial version of the method, which is called the Software Functionality Evolution Method (SFEM). Different from a situational method [7], [8], this template method prescribes what activities and concepts are to be implemented, rather than what an instantiation of the method would look like [10].

In two cases at the ERP software vendor, the template method is instantiated, of which a backlog is recorded for analysis purposes. This backlog contains rationales on the instantiation of activities and concepts, and decisions made accordingly. A structured approach towards the cases is followed [11], [12].

The performance of the template method instantiation is analyzed, in order to identify opportunities to improve the method. The cycle of method engineering, template method instantiation and method improvement is repeated until a stable version of the SFEM is engineered. The method will be contributed to the Software Product Management Body of Knowledge (SPMBOK) [13], classifying the research as design science [14].

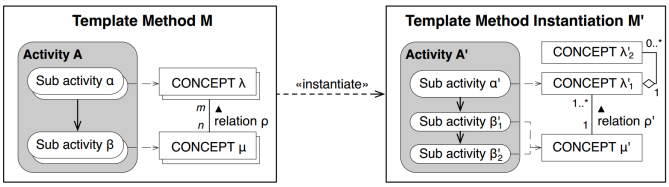

Fig. 2. Template method instantiation [10]

The results of the initial research lay a basis for the design of the first version of the template method. The template method and a case instantiation are analyzed, which results in a set of method increments to improve the template method. The process of instantiation and analysis is repeated in a second case, which results in the final template method.

III. Software Functionality Evolution Method

The Software Functionality Evolution Method (SFEM) is a template method which assists a software developing organization in the evolution of a software product by means of mapping functionality between software platforms. A template method is different from a situational method as it serves as a template for an instantiation, rather than describing the instantiation of the situational method itself. In Figure 2, the concept of template method instantiation is visualized [10]. The figure indicates how the open activities and concepts of a template method may result in extra elements after instantiation.

The method is visualized as a Process-Deliverable Diagram (PDD), which is a technique used for modeling activities and artifacts of a certain process [9]. On the left side of the diagram are the activities of the method's process, of which the notation is based on the UML activity diagram [15]. On the right side of the diagram, deliverables are visualized as concepts to indicate what artifacts are produced by a template method instantiation, of which the notation is based on the UML class diagram [15].

The template method's PDD is shown in Figure 3. The corresponding concepts are explained in Section III-A, followed by supporting theories in Section III-B.

A. Concepts

The concepts of the SFEM are the artifacts of a template method instantiation, produced by the execution of activities in the method. We introduce a definition of each concept in the method, followed by supporting theories in Section III-B.

PROJECT PLAN — The PROJECT PLAN is a document that describes the technical and management approach to be followed for a project. The plan typically describes the work

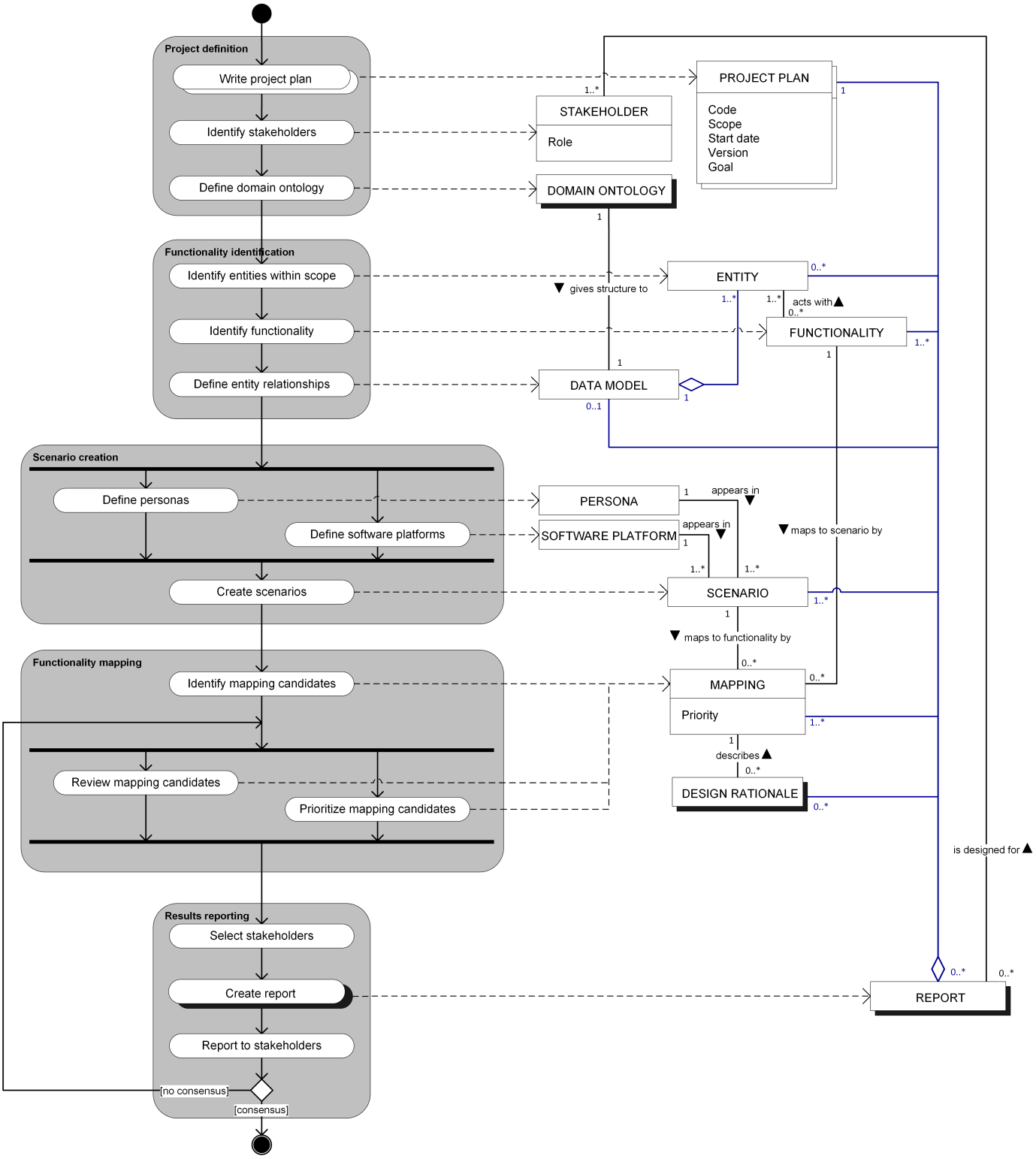

Fig. 3. Software Functionality Evolution Method as a Process-Deliverable Diagram

to be done, the resources required, the methods to be used, the procedures to be followed, the schedules to be met, and the way that the project will be organized [16].

STAKEHOLDER — A STAKEHOLDER is an individual or organization having a right, share, claim or interest in a system or in its possession of characteristics that meet their needs and expectations [17]. A STAKEHOLDER's *Role* in a template method instantiation can be organized as described in Section III-B1.

DOMAIN ONTOLOGY — An ontology describes the ENTITIES within the domain in discourse, and how these ENTITIES are interrelated [18]. The DOMAIN ONTOLOGY represents the domain in which the software product is designed to operate, the domain of discourse.

ENTITY — In computer programming, an ENTITY is any item that can be named or denoted in a program. For example, a data item, program statement, or subprogram. [16].

FUNCTIONALITY — FUNCTIONALITY concerns the capabilities of the various computational, user interface, input, output, data management, and other features provided by a product [19].

DATA MODEL — A DATA MODEL identifies the ENTITIES, domains (attributes), and relationships (associations) with other data and provides the conceptual view of the data and the relationships among data [20].

PERSONA — PERSONAS are defined as representations of the actual users of a system, defined by the goals they aim to accomplish. They are hypothetical archetypes of actual users [21].

SOFTWARE PLATFORM — A platform is the combination of an operating system and hardware that makes up the operating environment in which a program runs [22]. Thus, a SOFTWARE PLATFORM defines the environment in which a software product is designed to operate.

SCENARIO — The combination of possible and relevant appearances of PERSONAS on SOFTWARE PLATFORMS creates an instantiation of the concept SCENARIO. A SCENARIO is used to map FUNCTIONALITY by MAPPINGS and their *Priorities*.

MAPPING — A MAPPING is an assigned correspondence between two things that is represented as a set of ordered pairs [20]. The concepts is instantiated by the combination of a SCENARIO with FUNCTIONALITY. The *Priority* of a MAPPING is determined based on the importance of the FUNCTIONALITY for the given SCENARIO. If a MAPPING has no *Priority* assigned, it is considered to be a candidate.

DESIGN RATIONALE — A DESIGN RATIONALE is defined as information capturing the reasoning of the designer that led to the system as designed, including design options, trade-offs considered, decisions made, and the justifications of those decisions [23]. It presents the arguments behind a MAPPING and its *Priority*.

REPORT — A REPORT is an information item that describes the results of activities such as investigations, observations, assessments, or tests [24]. The results of an instantiation are communicated to selected STAKEHOLDERS by a REPORT, which is designed to suit the STAKEHOLDER's interests.

		Degree of participation	
		Active	Passive
Degree of interaction	Direct	**Participant** - Project manager - Product manager - Domain expert	**Observer** - Project manager - Product manager - Board member
	Indirect	**Informer** - Domain expert - Customer - Partner	**Outsider** - Developer - Designer

Fig. 4. Method Stakeholder Classification Matrix

B. Theoretical Embedding

To support the instantiation of activities and concepts of the SFEM, this research has explored various theories for the embedding of the method. These theories assist an analyst in the instantiation of the template method, by means of extra background information and supporting techniques for the implementation of processes.

1) Method Stakeholder Classification Matrix: In method engineering, it is possible that a process is explicitly carried out by a specific individual or organizational role. In that case, the role is indicated in the activity depicted in the method [9]. On the other hand, stakeholders are involved in a method instantiation to provide or consume information.

To help identify these stakeholders and apply a classification to their role in the method's instantiation, we introduce the Method Stakeholder Classification Matrix (MSCM), presented in Figure 4. The role of the stakeholder is dependent on the *degree of participation* in the instantiation of the template method, and the *degree of interaction* with the deliverables of the instantiation.

The MSCM is applicable in the activity *Identify stakeholders*, and makes the process of describing stakeholders more efficient. The selected value from the matrix can be used as the *Role* of the concept STAKEHOLDER.

2) Software Functionality Identification: To extract entities and their functionality from existent software products and underlying architectures, many techniques have already been discussed in scientific literature. Given the audience of this template method, as discussed in Section I, we have limited the exploration of such techniques by excluding technical approaches such as code-analysis. The techniques are applicable to the activities within the main activity *Functionality identification*, and the concepts resulting from these activities.

The analysis of a user manual allows for an analyst to identify functionality as it was designed and documented for the user. Natural Language Processing [25] can support analysis of such texts by tokenization, and generating tag clouds.

Architecture reconstruction, the process where the "as-built" architecture of an implemented system is obtained from an existing legacy system [26], helps in the identification of relationships among entities, and how this is translated into functionality. Different tools for reconstruction exist, such as ARMIN [26] and the Dali Architecture Reconstruction workbench [27].

A categorization of architecture reconstruction approaches [28] distinguishes *manual architecture reconstruction*, *manual reconstruction with tool support*, *query languages for writing patterns to build aggregations automatically*, and other techniques, such as *clustering*, *data mining* and *architecture description languages*.

To support the identification of functionality, the application of a functionality classifier to entities assists in covering a large portion of the functionality. Examples of such classifiers include CRUD [29], BREAD [30] and read-only/maintain.

3) Scenarios of Personas and Software Platforms: In the template method, the concept SCENARIO plays a central role in the mapping of functionality between software platforms. As described in Section III-A, a scenario is the appearance of a persona on a given software platform. A persona may appear on one or more software platforms, and a software platform may host one or more appearances of personas. It is the combination of personas on platforms that is used to create the mapping of functionality, indicating the priority of the functionality for a given scenario.

As it is not desirable to let an actual user directly influence the designing process [21], the use of pretend users as personas is a good way to represent the actual users during systems design. Different sources [21], [31]–[33] have contributed to the following focus areas in the description of personas:

Characteristics — Make the persona live in the minds of designers by giving him/her characteristics like a name, photo, demographic data and attitudes.
Needs and goals — Explicate what a persona wishes to achieve, which can be achieved by the use of a software product. Goals can be classified as *life goals*, *experience goals* and *purpose goals* [34].
Skills and competencies — Driven by experience and knowledge, skills and competencies define a persona's abilities, and how they are limited in their actions.
Constraints — The separate definition of constraints, which may reside from other focus areas, puts extra emphasis on the inability of personas. These constraints are of particular interest when mapping functionality in the *Functionality mapping* activity phase of the template method.

Platforms, defined as "a foundation technology or set of components used beyond a single firm that brings together multiple parties for a common purpose or recurring problem" [35], represent opportunities for new software applications which a software vendor might adopt in the evolution of a software product. Four focus areas have been defined to help in the description of software platforms:

Platform type — A classification of the platform, either being *desktop*, *web*, *mobile* or *wearable* [36], [37] or *internal platform*, *supply chain platform*, *industry platform* or *two-sided market* [38].
SWOT analysis — The analysis of *strengths*, *weaknesses*, *opportunities* and *threats* of a software platform explicates the potential of the platform in the software product evolution.
Functional context and constraints — Given the functional context of a platform, either virtual or physical, it may enable or restrict the mapping of certain functionality.
Technical context and constraints — Technical opportunities or constraints may allow or disallow for the mapping of functionality to a software platform.

4) Mapping and Prioritization: The mapping of functionality on scenarios is the main goal of the template method. The activities *Review mapping candidates* and *Prioritize mapping candidates* in the PDD of the SFEM are conducted in a group session with relevant stakeholders, in which the mapping and mapping priority of functionality is discussed. A variety of requirements prioritization techniques exist [39], [40], of which the selection of the correct technique is dependent on the complexity of the project at hand.

Techniques which are relevant in the prioritization of requirements include the *Binary Priority List* [41], *cumulative voting* [42], *ranking* [39], the *Wiegers' prioritization model* [43], and *priority groups* [42], [44] such as *MoSCoW* [45] and *requirements triage* [46].

Creating a mapping is dependent on the characteristics and constraints of the persona and software platform of a scenario, and the characteristics and constraints of the regarded functionality. The priority assigned determines the importance of the mapping, compared to other mappings of functionality.

A mapping may have one or more instantiations of the concept DESIGN RATIONALE assigned, which captures the decision making process during the mapping activities. This allows for reasoning about the decisions in later stages of the product evolution.

IV. Reflecting on Method Increments

A multitude of methods has been developed since the emergence of the method engineering research field. All too often, however, the processes of method creation, as well as decisions made within remain underexposed, limiting understanding and repeatability of the respective method engineering research [10].

While elementary method increment types have been distinguished in earlier research [47], these types do not take into account reasoning and motivation for usage, limiting for reflection on method creation. We have attempted to address this issue by categorizing method increments. The following method increment categories[1] have been identified based on creation of the SFEM:

[1]Obviously, this set of method increment categories is not complete and is to certain extent specific to our method.

Constructing (C) — Adding, changing or removing (properties of) activities, concepts or properties in the diagram, including activity and concept types.
Labeling (L) — Adding, changing or removing a label of (properties of) activities, concepts, properties, associations or roles.
Associating (A) — Adding, changing or removing (properties of) associations between existing activities or concepts.

By explicating and motivating each increment in a method increment log, the method construction process as a whole is explicated. Below, an excerpt of the SFEM increment log with motivations per increment is shown.

C	We need to capture documentation to reside with the VISUALIZATION. Instead of adding an explicit concept, a VISUALIZATION will include the concept DESIGN RATIONALE. This implies that a VISUALIZATION does not necessarily need to be a figure, it can just as well be textual.
C	In the *Peer review* activity, a VISUALIZATION of the method's output must be included in order to review the performance. Therefore, we merged the *Peer review* activity with the main activity *Visualization*. This implies that after the *Visualization* activity, we must also be able to return to the *Mapping* main activity.
A	We can't set the project's goal without knowing what the scope actually is we're working in. Therefore, the activity *Set project scope* will be implied by the activity *Define migration project*.
A	The concepts SOFTWARE PLATFORM and PERSONA both appear in at least one SCENARIO, otherwise it would not make sense to define either of the concepts at all.
L	The main activity *Platforms and personas* should be renamed to *Scenarios*, as the aim of the main activity is to develop SCENARIOS.
L	The term for the concept OBJECT is ambiguous. The term ENTITY is more suitable, considering the method domain's jargon.
L	The concept OBJECT TREE should be renamed to DATA MODEL, considering the method domain's jargon.

Figure 5 visualizes how different versions of template method instantiations contribute to the creation of the template method. Since a new version of a template method is constructed based on input from the previous version of the template method as well as its instantiation, template methods particularly benefit from method increment reflection. When a new version of a template method is to be developed, potential method increments (as well as underlying reasoning and motivations) are considered, compared and weighed from both an abstract/conceptual (template) perspective and a concrete/practical (instantiation) perspective. During the construction of the SFEM (Figure 3), we learned that this approach results in thorough yet rapid method development.

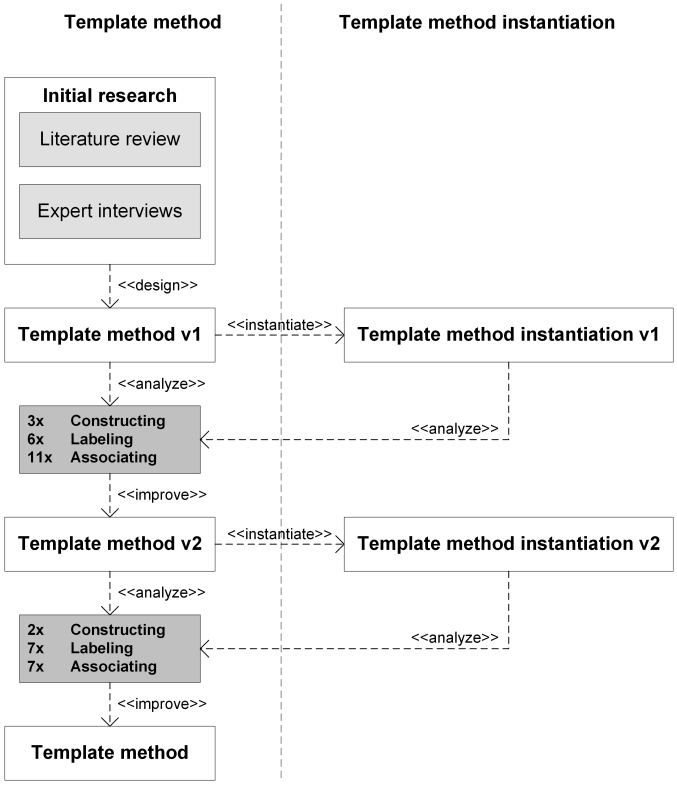

Fig. 5. Template method increments

V. Case Results

The Software Functionality Evolution Method (SFEM) has been instantiated in two cases at an ERP software vendor, having its main office in the Netherlands. The software suite in scope exists of a Windows client application and two web applications, which are an intranet and a portal application. The Windows client is considered being the originating platform, having the base set of functionality.

In an ongoing attempt to evolve the software suite to meet current customer demands, functionality is being extracted from the Windows client and implemented in the web applications. The SFEM has been instantiated to assist the mapping of functionality of two function groups on the intranet platform.

In Figure 6, we present a snippet of the first template method instantiation in a case. The case was scoped towards the function group *Course management* and related functionality, currently implemented in the Windows client platform. Only the software platform INTRANET was relevant in this case, because the existing functionality is designed for employees within an organization. For the example in this paper we have limited ourselves to the persona TEACHER, although other personas have been identified and included in the complete case. We have selected the functionality VIEW PARTICIPANTS PER COURSE EVENT, originating from the entity COURSE EVENT. Given the scenario TEACHER ON INTRANET and the selected functionality, a MAPPING was assigned with a relatively high *Priority* of *0.9*. This is because during the group

session, it became obvious that during a course day, a teacher is particularly interested in the number of participants and their names and organizations, which helps a teacher to prepare for the course. Therefore, a high priority is assigned to the mapping, increasing the possibility that this functionality will eventually be implemented in the intranet web application.

In the first case, a total of 135 sets of functionality was initially identified. 90 sets of functionality were never assigned a mapping, which means they are currently not relevant in the evolution of the software suite for this software platform and identified personas. Of the remaining 45 sets of functionality, an average of the mapping priorities was calculated per functionality, by dividing the sum of priorities by the total number of scenarios. The average was not calculated by dividing the sum of priorities by the number of assigned mappings, as it would neglect the meaning of a missing mapping for a set of functionality. The functionality, the scenarios and their mappings are presented in a matrix, sorted in a descending order by their average mapping priority. A part of the case's report is visualized in Figure 7.

The results of the case for *Course management* have been compared with the actual implementation of the function group in the intranet web application. An analysis of the results has pointed out that even though many similarities exist, some functionality was included through mappings in the case, while the actual web application does not (yet) host the mapped functionality. A discussion with the organization's stakeholders concluded that the mappings had been decided with a pure focus on the added value for the scenarios, and current technical and organizational implications have been left out in the decision making process. This has been a conscious decision. The consideration of required and available resources would have made the mapping process of the method too complicated and less efficient, as it would require more discussion, stakeholders and time.

The second case in which the template method was instantiated was scoped towards functionality in the function group *Fixed assets management*. It initiated with a total of 79 sets of functionality, of which 56 sets were assigned a mapping and priority. This implies that 23 sets of functionality were excluded from the software evolution by discussing them in the group session and deciding not to assign any mapping at all. The second case concerned functionality which has not yet been implemented in the intranet web application. An analysis of the results has shown that an initial consideration of the possibilities of implementing the function group in the intranet web application would be of added value to the identified personas. During the execution of the project, it became clear that the software platform even poses opportunities for new functionality to be developed, due to the functional context of the software platform. However, the results of the case again stressed not to be suitable to be considered a product roadmap, as other important factors have not been discussed during the mapping of functionality, such as required and available resources, strategy, and the software ecosystem.

VI. Related Work

The Actor Dependency (AD) model [48] analyzes the software processes to capture *why* a software process has been implemented by a software developing organization, rather than *how* it was implemented, or *what* the process was designed like. This creates a better understanding of the composition of software development processes and the motivations, intents and rationales behind them. The model creates a basis for the research in terms of understanding software processes and rationales that capture a decision making process.

Strategies to cope with legacy information systems can be subdivided into three categories: *redevelopment*, *wrapping* and *migration* [1]. The difference in impact on the current and new system give a good impression of the wide range of aspects to take into account in software evolution.

The context of legacy system reengineering can be seen from the perspective of *engineering*, *system*, *software*, *managerial*, *evolutionary* and *maintenance* [49]. Each perspective comes with challenges to be considered, to realize an effective approach towards reengineering. Such challenges may also play a role during the mapping and decision making process of the evolutionary method.

The Chicken Little Methodology [50] is considered to be the most mature approach for the migration of a software product [51]. However, the approach makes extensive use of gateways, which increases the complexity of the software. Therefore, the Butterfly Methodology is a gateway-free approach to legacy system migration which reduces the risk of increasing complexity [51].

In Method Engineering [8], [9], situational methods are engineered which are tuned to the situation of the project at hand [7]. Template methods [10] are designed to be more generic, by describing *what*, rather than *how*, the activities and concepts are to be implemented by the instantiating organization. The Software Functionality Evolution Method is designed as a template method, to make it more generic and applicable in different project situations.

VII. Discussion

The research project's goal is to design a template method for software developing organizations. Since the role of a software product manager does not necessarily imply having in-depth knowledge about the product's source code and technical architecture, techniques that concern technical competencies such as the analysis of source code or implemented architecture, are omitted. Thus, the software product is analyzed from a functional perspective. However, this might not be conclusive, and a functional approach may produce more overhead and consume more resources compared to technical, potentially automated approaches that were left out in the research.

During the extraction of entities and functionality from the software product, functionality is not clustered, nor are relationships between functionality recorded. Should we have decided to do so, the process of mapping functionality on scenarios would have become too complex, due to a cascading

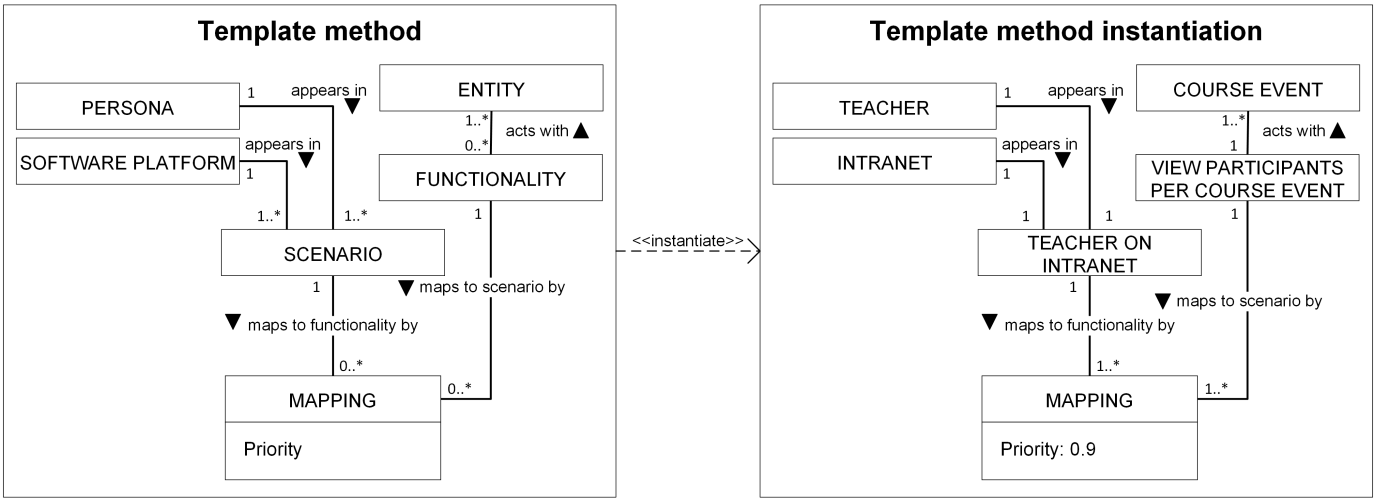

Fig. 6. Template method instantiation for Course management

entity	functionality	manager	teacher	employee	AVG/3
Course	Show course	5	5	5	5,0
Course session	Show course session of course event	5	5	5	5,0
Dossier item	Show dossier item	5	5	5	5,0
Dossier item	Show attachment of dossier item	5	5	5	5,0
Course event	Show course event of course	5	5	5	5,0
Location	Show location	5	5	5	5,0
Organization/Person	Show organization/person	5	5	5	5,0
Dossier item	Add dossier item	5	5	3	4,3
Dossier item	Edit dossier item	5	5	3	4,3
Dossier item	Add attachment to dossier item	5	5	3	4,3
Occupation	Show occupation of course events	5	5	2	4,0
Occupation	Show occupation of locations	5	5	2	4,0
Participant	Show participants per course event	5	5	2	4,0
Participant	Show participant of course event	5	5	2	4,0
Organization/Person	Edit organization/person	5	2	3	3,3
Presence	Show presence of participants	4	4	1	3,0
Presence	Edit presence of participants	4	4	1	3,0
Course session	Add course session to course event	5	2		2,3
Participant	Add participant to course event	5	2		2,3

Fig. 7. Report for Course management

assignment of priorities among functionality. This does not benefit the efficiency of the template method, as the mapping phase is not designed to consider such extensive dependencies.

The template method's deliverable, an instantiation of the concept REPORT, is not suitable to be considered a product roadmap. The method does not take into account the required and available resources for the implementation of each set of functionality during the mapping phase. The required resources are not exclusively dependent on the characteristics of the functionality itself, as the difficulty of implementing functionality can be different per software platform. However, it has been a conscious decision to exclude the consideration of resources, as it would increase the complexity of the mapping phase, making the method less efficient. The prioritized shortlist of requirements, delivered by the method as a report, can in turn be used as input for roadmap intelligence, as described in Section I and visualized in Figure 1.

Also not explicitly considered in the decision-making process of mapping functionality are product strategy aspects, such as platform stability, reliability and pricing, and product ecosystem considerations. However, such considerations can be taken into account during the mapping-phase of the method, and can be captured by means of design rationales.

The template method has been designed, instantiated and validated at a single case company, which produces an integrated ERP software product. It may be possible that validation at another case company, producing different software products, possibly even adhering to another development method-

ology, may result in different performance. Such validation is left open for future research.

VIII. Conclusion

This research proposes an answer to the research question by developing a template method which assists a software developing organization in the evolution of a software product by mapping functionality between software platforms. The method is labeled the Software Functionality Evolution Method (SFEM). Five phases are to be executed in an instantiation of the method, which are *Project definition*, *Functionality identification*, *Scenario creation*, *Functionality mapping* and *Results reporting*. To create an instantiation which is tuned towards the project characteristics, different method fragments are proposed which support the execution of activities in the method.

The template method has been validated by means of instantiations in two different cases. By analyzing the method's design and performance, the method is improved by designing method increments. A categorization for method increments is proposed, which allows for reflection on a research process by a researcher.

The two cases in which the template method is instantiated are performed at one single software developing organization in the Netherlands, which produces an integrated ERP system for the Dutch marketplace. Future work validates and improves the template method by means of cases at other software vendors, with different software products and different project requirements. Thus, a more quantitative approach towards the instantiation and validation of the template is subject to future research.

The research project is designed for software product managers, and thus technical approaches for the analysis of software products and their architecture are omitted. Future work explores the (semi-)automated analysis of software products, to make the process of extracting entities and functionality more efficient and conclusive.

In the mapping phase of the template method, opportunities exist for automated application and cascading of priorities, based on relationships between entities and functionality. However, this is not explored in this research project, as it is still unclear what effect cascading has on the outcome of an instantiation. Such dependencies would require a thoroughly tested algorithm which automatically cascades a mapping's priority based on properties of the relationship between entities or functionality.

Acknowledgments

We would like to thank all interviewees who have participated in the cases and group sessions for their knowledge and cooperation during the research project. Their contribution has been of great significance to the development of the template method.

References

[1] J. Bisbal, D. Lawless, B. Wu, and J. Grimson, "Legacy information systems: Issues and directions," *Software, IEEE*, vol. 16, no. 5, pp. 103–111, 1999.

[2] J. T. Yee and S.-C. Oh, "Focusing on RFID, Interoperability, and Sustainability for Manufacturing, Logistics, and Supply Chain Management," in *Technology Integration to Business*. Springer, 2013, pp. 67–95.

[3] V. T. Rajlich and K. H. Bennett, "A staged model for the software life cycle," *Computer*, vol. 33, no. 7, pp. 66–71, 2000.

[4] C. Ebert, "The impacts of software product management," *Journal of Systems and Software*, vol. 80, no. 6, pp. 850–861, Jun. 2007.

[5] W. Bekkers, I. van de Weerd, M. Spruit, and S. Brinkkemper, "A Framework for Process Improvement in Software Product Management," in *Systems, Software and Services Process Improvement*. Springer Berlin Heidelberg, 2010, vol. 99, pp. 1–12.

[6] D. Moher, A. Liberati, J. Tetzlaff, and D. G. Altman, "Preferred Reporting Items for Systematic Reviews and Meta-Analyses: The PRISMA Statement," *Annals of Internal Medicine*, vol. 151, no. 4, pp. 264–269, 2009.

[7] F. Harmsen, S. Brinkkemper, and H. Oei, *Situational Method Engineering for Information System Project Approaches*. University of Twente, Department of Computer Science, 1994, no. September.

[8] S. Brinkkemper, "Method engineering: engineering of information systems development methods and tools," *Information and software technology*, vol. 38, no. 4, pp. 275–280, 1996.

[9] I. van de Weerd and S. Brinkkemper, "Meta-Modeling for Situational Analysis and Design Methods," in *Handbook of research on modern systems analysis and design technologies and applications*. Information Science Reference, 2008, vol. 35, pp. 35–54.

[10] H. van der Schuur, "Process Improvement through Software Operation Knowledge: If the SOK Fits, Wear It!" *SIKS Dissertation Series*, vol. 2011, no. 43, 2011.

[11] R. K. Yin, *Case study research: Design and methods*. Sage, 2009.

[12] P. Runeson and M. Höst, "Guidelines for conducting and reporting case study research in software engineering," *Empirical Software Engineering*, vol. 14, no. 2, pp. 131–164, Dec. 2009.

[13] International Software Product Management Association. (2014) Software Product Management Body of Knowledge (SPMBOK). [Online]. Available: http://ispma.org/spmbok/

[14] A. R. Hevner, S. T. March, J. Park, and S. Ram, "Design Science in Information Systems Research," *MIS quarterly*, vol. 28, no. 1, pp. 75–105, 2004.

[15] Object Management Group, "UML 2.0 superstructure specification," Technical Report ptc/04-10-02, Tech. Rep., 2004.

[16] "IEEE Standard Glossary of Software Engineering Terminology," *IEEE Standard 610.12*, 1990.

[17] "Systems and software engineering – Software life cycle processes," *IEEE Standard 12207*, 2008.

[18] T. R. Gruber, "A translation approach to portable ontology specifications," *Knowledge Acquisition*, vol. 5, no. 2, pp. 199–220, 1993.

[19] "IEEE Guide for Information Technology - System Definition - Concept of Operations (ConOps) Document," *IEEE Standard 1362*, 1998.

[20] "IEEE Standard for Conceptual Modeling Language - Syntax and Semantics for IDEF1X97 (IDEFobject)," *IEEE Standard 1320.2*, 1998.

[21] A. Cooper, *The inmates are running the asylum: Why hightech products drive us crazy and how to restore the sanity*. Indianapolis, IN: SAMS, Macmillan Computer Publishing, 1999.

[22] "IEEE Standard for Adoption of ISO/IEC 26513:2009 Systems and Software Engineering–Requirements for Testers and Reviewers of Documentation," *IEEE Standard 26513*, 2010.

[23] "IEEE Standard for Information Technology–Systems Design–Software Design Descriptions," *IEEE Standard 1016*, 2009.

[24] "ISO/IEC/IEEE Systems and software engineering – Content of life-cycle information products (documentation)," *IEEE Standard 15289*, 2011.

[25] G. G. Chowdhury, "Natural language processing," *Annual Review of Information Science and Technology*, vol. 37, no. 1, pp. 51–89, 2003.

[26] R. Kazman, L. O'Brien, and C. Verhoef, "Architecture Reconstruction Guidelines, Third Edition," Software Engineering Institute, Carnegie Mellon University, Tech. Rep. November, 2003.

[27] R. Kazman and S. J. Carrière, "Playing Detective: Reconstructing Software Architecture from Available Evidence," *Automated Software Engineering*, vol. 6, no. 2, pp. 107–138, 1999.

[28] L. O'Brien, C. Stoermer, and C. Verhoef, "Software Architecture Reconstruction: Practice Needs and Current Approaches," Software Engineering Institute, Carnegie Mellon University, Tech. Rep. August, 2002.

[29] J. Martin, *Managing the data base environment*. Prentice Hall PTR, 1983.

[30] M. Stolze, P. Riand, M. Wallace, and T. Heath, "Agile Development of Workflow Applications with Interpreted Task Models," in *6th international conference on Task Models and Diagrams for User Interface Design*. Springer, 2007, pp. 2–14.

[31] C. G. Jung and A. E. Storr, *The essential Jung*. Princeton University Press, 1983.

[32] M. Aoyama, "Persona-and-scenario based requirements engineering for software embedded in digital consumer products," in *Proceedings of the 13th IEEE International Conference on Requirements Engineering*, 2005, pp. 85–94.

[33] P. T. A. Junior and L. V. L. Filgueiras, "User modeling with personas," in *Proceedings of the 2005 Latin American conference on Human-computer interaction*. New York, New York, USA: ACM Press, 2005, pp. 277–282.

[34] A. Cooper, R. Reimann, and D. Cronin, *About Face 3: The Essentials of Interaction Design*. John Wiley & Sons, 2012.

[35] A. Gawer and M. A. Cusumano, "Platform leadership: How Intel, Microsoft, and Cisco drive industry innovation," *Innovation: Management, Policy & Practice*, vol. 5, no. 1, pp. 91–94, 2003.

[36] J. Bosch, "From Software Product Lines to Software Ecosystems," in *Proceedings of the 13th International Software Product Line Conference*, no. Splc. Carnegie Mellon University, 2009, pp. 111–119.

[37] S. Mann, "Wearable computing: a first step toward personal imaging," *Computer*, vol. 30, no. 2, pp. 25–32, 1997.

[38] A. Gawer, *Platform dynamics and strategies: from products to services*. Cheltenham: Edward Elgar Publishing Limited, 2009.

[39] P. Berander and A. Andrews, "Requirements Prioritization," in *Engineering and Managing Software Requirements*. Springer Berlin Heidelberg, 2005, pp. 69–94.

[40] Z. Racheva, M. Daneva, and L. Buglione, "Supporting the Dynamic Reprioritization of Requirements in Agile Development of Software Products," in *Second International Workshop on Software Product Management*, Barcelona, Catalunya, 2008, pp. 49–58.

[41] T. Bebensee, I. van de Weerd, and S. Brinkkemper, "Binary Priority List for Prioritizing Software Requirements," in *Requirements Engineering: Foundation for Software Quality*. Springer Berlin Heidelberg, 2010, pp. 67–78.

[42] D. Leffingwell and D. Widrig, *Managing software requirements: a unified approach*. Addison-Wesley Professional, 2000.

[43] K. Wiegers, "First things first: prioritizing requirements," *Software Development*, vol. 7, no. 9, pp. 48–53, 1999.

[44] I. Sommerville and P. Sawyer, *Requirements engineering: a good practice guide*. John Wiley & Sons, Inc., 1997.

[45] DSDM Consortium, *DSDM Atern Handbook*. DSDM Consortium, 2008. [Online]. Available: http://www.dsdm.org/content/10-moscow-prioritisation

[46] A. M. Davis, "The art of requirements triage," *Computer*, vol. 36, no. 3, pp. 42–49, 2003.

[47] I. van de Weerd, S. Brinkkemper, and J. Versendaal, "Concepts for Incremental Method Evolution: Empirical Exploration and Validation in Requirements Management," in *Advanced Information Systems Engineering SE - 33*, ser. Lecture Notes in Computer Science, J. Krogstie, A. Opdahl, and G. Sindre, Eds. Springer Berlin Heidelberg, 2007, vol. 4495, pp. 469–484.

[48] E. S. Yu and J. Mylopoulos, "Understanding "why" in software process modelling, analysis, and design," in *Proceedings of the 16th international conference on Software engineering*. IEEE Computer Society Press, 1994, pp. 159–168.

[49] S. R. Tilley and D. Smith, "Perspectives on Legacy System Reengineering," 1995.

[50] M. L. Brodie and M. Stonebraker, *Migrating legacy systems: gateways, interfaces & the incremental approach*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1995.

[51] B. Wu, D. Lawless, J. Bisbal, R. Richardson, J. Grimson, V. Wade, and D. O'Sullivan, "The butterfly methodology: A gateway-free approach for migrating legacy information systems," in *Third IEEE International Conference on Engineering of Complex Computer Systems*. IEEE, 1997, pp. 200–205.