A Situational Approach for the Definition and Tailoring of a Data-Driven Software Evolution Method

Xavier Franch[1], Jolita Ralyté[2], Anna Perini[3], Alberto Abelló[1], David Ameller[1], Jesús Gorroñogoitia[4], Sergi Nadal[1], Marc Oriol[1], Norbert Seyff[5], Alberto Siena[6], Angelo Susi[4]

[1]Universitat Politècnica de Catalunya (UPC), Barcelona, Spain
{aabello, franch, dameller, snadal, moriol}@essi.upc.edu
[2] University of Geneva, Switzerland
jolita.ralyte@unige.ch
[3] Fondazione Bruno Kessler (FBK), Trento, Italy
{perini, susi}@fbk.eu
[4] ATOS, Madrid, Spain
jesus.gorronogoitia@atos.net
[5] University of Applied Sciences Northwestern Switzerland (FHNW)
norbert.seyff@fhnw.ch
[6] Delta Informatica SpA, Trento, Italy
alberto.siena@gmail.com

Abstract. Evolution is a cornerstone activity in the software life cycle. Successful evolution heavily depends on the selection of the right features to be included in the next release. Such selection is difficult, and companies often report bad experiences about user acceptance. To overcome this challenge, there is an increasing number of approaches that propose intensive use of data to drive evolution. This trend has motivated the SUPERSEDE method, which proposes the collection and analysis of user feedback and monitoring data as the baseline to elicit and prioritize requirements, which are then used to plan the next release. However, every company may be interested in tailoring this method depending on factors like project size, scope, privacy issues, etc. In order to provide a systematic approach, we propose the use of Situational Method Engineering to describe SUPERSEDE and guide its tailoring to a particular context.

Keywords: software evolution, situational method engineering, software process.

1 Introduction

Software evolution aims at keeping software systems of any kind aligned with users' needs, which are influenced by individual, social, economic, and technological changes. That part of software engineering is receiving more and more attention since software has become a pervasive and key element in modern society [1][2].

Basic principles and processes of software evolution [3] have been revisited in the light of the high availability of data, reflecting the variety of modern software, and of efficient techniques to mine such data. Indeed, key knowledge to foster short and fre-

Franch, X. [et al.]. A situational approach for the definition and tailoring of a data-driven software evolution method. A: International Conference on Advanced Information Systems Engineering. "Advanced Information Systems Engineering: 30th International Conference, CAiSE 2018: Tallinn, Estonia, June 11-15, 2018: proceedings". Berlín: Springer, 2018, p. 603-618. DOI 10.1007/978-3-319-91563-0_37

quent evolution cycles can be extracted from these data, which is both produced explicitly by users (i.e. user feedback), and resulting from monitoring the context in which the software is executed, as well as the software itself at runtime [1].

The SUPERSEDE H2020 project (www.supersede.eu) enables a data-driven software evolution process, by offering a set of tools that can be flexibly combined to support an iterative and incremental development process that fits specific needs of companies and projects. The consortium includes three companies, SIEMENS, ATOS and SEnerCon, which are playing a main role in the understanding of the key insights of data-driven software evolution and that allow to evaluate project results in contexts which vary regarding, among others: (i) the number and heterogeneity of end-users, ranging from about 10 app developers in the SIEMENS use case, to millions of Olympic game spectators in ATOS; ii) the variety of deployment technology, ranging from mobile devices to desktop computers; iii) the type of user feedback, for instance user forums or issue tracking systems; iv) the size and type of the organization, e.g. big and distributed company, which tends to adopt a structured, certified software development process, vs. small company, which aims at using a simplified agile process.

One of the outcomes of the SUPERSEDE project is the formulation of a method. The SUPERSEDE method drives the data-driven evolution process in a systematic way. Therefore, it needs to reconcile generality to accommodate such different types of companies (and others that potentially may be interested) and customisability, as to allow the method to be effectively adopted by a company.

The research goal described in this paper is to provide a systematic definition of the SUPERSEDE method for data-driven software evolution. We aim at SUPERSEDE acting as a reference method that can be tailored to different situations. Therefore, we adopt Situational Method Engineering (SME) [4] as engineering approach to design the method as a composition of reusable components called method chunks. The research questions that we address in relation to this type of evolution are:

RQ1. What are the constituent parts of method chunks?

RQ2. What are the most fundamental method chunks for SUPERSEDE evolution?

RQ3. What are the criteria whose combination allows expressing the context in which these method chunks apply?

RQ4. How can the different method chunks be combined in order to create a customization of the SUPERSEDE method tailored to a particular context?

The rest of the paper is organized as follows. Section 2 presents the research method. Section 3 provides the background. Sections 4 to 7 are the core of the paper, addressing the four research questions. Finally, Section 8 presents conclusions and future work.

2 Research Context and Method

As commented in the introduction, the SUPERSEDE method is an engineering artefact developed in the context of a project with the same name. The research method has been somehow coordinated with the project timeline, so we find three milestones:

- *Requirements gathering*. The first milestone finished at the third month of the project (July 2015), and resulted in the documentation of the requirements posed by

the three companies [5]. Requirements comprised as-is and to-be goal models, plus the user stories emerging from this to-be model, all of them coming from workshops held at the companies' site (see protocol in [5]).

- *Elaboration of SUPERSEDE engineering artefacts (models, techniques and tools)*. This second milestone was basically achieved at the end of the second year (April 2017), although the artefacts will be refined until the end of the project (April 2018). The elaboration and evaluation of these engineering artefacts has helped to understand the intricacies of the elements to be integrated in the method.
- *Formulation of the SUPERSEDE method for evolution.* This third phase will be active until the end of the project (April 2018) and the first tangible result is this paper. The research method has consisted in the adoption of SME as scientific approach, and then the joint work of engineering artefacts providers under the coordination of the SUPERSEDE scientific manager, the SME expert and the supervision of the SUPERSEDE project coordinator (first three authors). We used a shared space to collaboratively elaborate the method components that will be outlined in this paper. Regular meetings allowed the incremental synchronization of the method components. Advances were systematically checked against the result of the first milestone (requirements compiled in [5]).

3 Background

3.1 Situational Method Engineering

To formalise the SUPERSEDE method, we apply Situational Method Engineering (SME) [4] principles and techniques. In SME, a method is viewed as a collection of autonomous and interoperable method components that can be selected and assembled in a way to satisfy the particular situation of the project at hand. The definition of method components and their assembly techniques vary form one SME approach to another [4]), but the main objective stays the same: to make the method knowledge modular and reusable for the construction and/or adaptation of situation-specific methods. Most of the assembly techniques support method construction from scratch based on the identified situational context and requirements, other deal with incremental organization's method adaptation or even method family construction.

In our work, we adopt and adapt the *assembly-based and method chunk-based SME* approach [6][7] that supports situation-specific method construction and extension in three steps: method requirements specification, method chunks selection and assembly of the selected chunks (details given in Section 6 as needed). Method chunks are reusable method components. A method chunk includes a process (i.e., the guidelines provided by the method chunk) and its related product knowledge (i.e., the formalisation of concepts and artefacts used and produced by the method chunk), and is specified by the situation in which it can be applied (i.e., the required input artefacts) and the intention (i.e. the engineering goal) to be reached. Finally, the reuse context of the method chunk is specified by a set of criteria that can be defined by using a taxonomy like the reuse frame proposed in [8]. We apply an *ad hoc method chunk definition* approach [9] that creates chunks from experts' knowledge, based on a method chunk metamodel.

3.2 The SUPERSEDE Control Loop

The data-driven software evolution process of SUPERSEDE takes inspiration from the autonomic control loop proposed for adaptive systems [10]. In SUPERSEDE, the control loop drives also software evolution, considering runtime and context data, and also explicit user's feedback, which the user might deliver upon having used the software.

The SUPERSEDE evolution process can be characterized by the following steps:

Collect. Multi-modal feedback gathering techniques allow users to express their feedback as textual comments, emoticons, rating and pictures. Flexible and configurable monitoring components collect data from the context and system usage [11]. These data, of different types, are stored in a big data storage, which maps data to a semantic model at support of analysis [12].

Analyse. Different types of analysis are tool-supported, for instance, sentiment analysis and extraction of feature requests and bug issues from user textual comments [13], tweet mining to understand perceived quality of experience [14], and combined analysis of end-user feedback and contextual data.

Decide. Focusing on software evolution tasks, automated reasoning techniques support collaborative-decision making concerning, for instance, the identification of new requirements, and their prioritization with respect to multiple criteria [15].

Act. Operationalizations of the decisions made are performed at this step. Selected features are included in a release plan that takes into account available resources, deadlines and organization priorities [16]. In addition, this step can be refined during the actual implementation of the next release using a continuous release planning approach [17].

4 Method Chunks Metamodel for SUPERSEDE

In this section, we implement the concept of method chunk using a process-oriented view. We base our work on existing software process modelling approaches, e.g. from SPEM [18]. We build a metamodel articulating the subset of method elements that are relevant for our purposes, namely activities, artefacts, roles and tools, each one yielding a particular class in the metamodel. Then, we link method chunks to these classes.

4.1 SUPERSEDE Method Elements

Activities, artefacts, roles and tools are declared as specializations of an abstract class MethodElement (see Fig. 1), which links every method element to one or more phases of the SUPERSEDE loop, allowing also referencing to external concepts (e.g., a role not needed by the method but referenced for the sake of completeness). We declare a reflexive association class to express relationships among method elements that will be specialized according to the different types of method elements whenever needed. All method elements may present structural relationships (composition and specialization) and each method element may present other particular types, as we detail below. The existence of specialization brings also the concept of abstract method element.

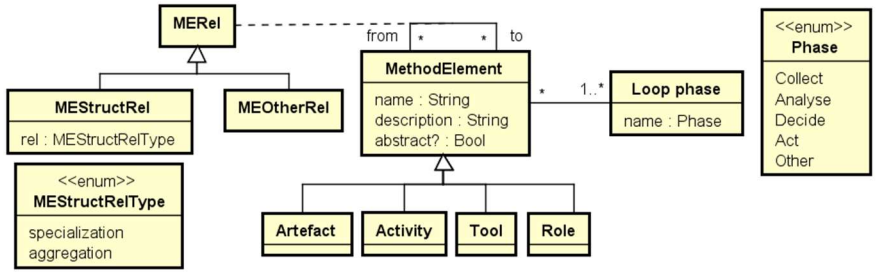

Fig. 1. Elements of the SUPERSEDE method

Activities involve artefacts, tools and roles as described in the paragraphs below. Examples are Feedback Collection (linked to the Collect phase), Domain Ontology Definition (Analyse), Requirements Prioritization (Decide) and Release Planning (Act). Structural relationships will be widely used; for instance, Requirements Prioritization will be specialized into several sub-activities, each adopting a particular prioritization strategy (e.g. Requirements Prioritization with AHP). In addition, activities may present several types of temporal relationships; we adopt the proposal in [19] (richer than SPEM when it comes to temporal relationships) with relationships like start-start, end-start, exclusion, etc. (e.g. Release Planning cannot start before Requirements Prioritization has ended). In all the types of process elements, particular relationships are defined by specializing the MEOtherRel class (see Fig. 2) and including the necessary integrity constraints to enforce application to the right type, e.g. `context` ArtefactRel `inv`: self.from->oclIsTypeOf(Activity) `and` self.to->oclIsTypeOf(Activity).

Artefacts represent informational resources that are produced, consumed or just used as a working asset by an activity. Examples of artefacts are Feedback Document (produced as output by the Feedback Collection activity), Prioritized List of Requirements (consumed as input by Release Planning) and AHP Matrix (used as working asset by Requirements Prioritization with AHP). Artefacts are declared of a particular category. Examples are: Model (e.g., Integration-Oriented Ontology), File (e.g., Monitoring Data), DataSet (e.g., Project Schedule), Technology (e.g., Event Queue Endpoint) and Expression (e.g., Complex Event Pattern). Categories may be decomposed into subcategories at any level, for instance the Integration-Oriented Ontology belongs to the Ontology subcategory in Model. Artefacts may be related to each other by structural relationships: aggregation (a List of Requirements as aggregation of Requirement) and specialization (Prioritized List of Requirements as specialization of List of Requirements). Other relationships are possible, like constraint (e.g., a Release Plan constrains a Project Schedule) or in-sync, meaning that changes in one artifact imply changes in another (e.g., a Release Plan is in-sync with a Prioritized List of Requirements).

Roles are involved in activities either individually (e.g., Project Owner) or as a set of person (e.g. Set of Developers), which is expressed with an association class (SetOf).

Tools are also used in activities, and they produce and consume artefacts and involve roles. All these associations among the four type of process elements are not independent and some integrity constraints need to be declared (not included here for the sake of brevity), e.g. `context` Activity `inv`: self.tool->includesAll(self.input.tool). Relationships among tools include connection (i,e., the result of one tool is used by another).

Again this association is related with others through integrity constraints, e.g. if the RePlan tool (which produces a ReleasePlan) is connected to DMGame (which produces a Prioritized List of Requirements), then the corresponding artefacts need to be in-sync.

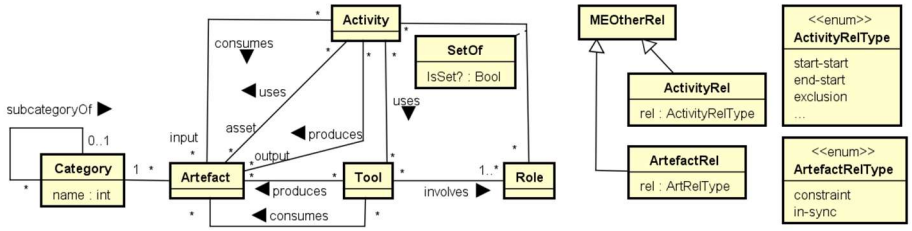

Fig. 2. Detail of the SUPERSEDE method elements

4.2 Definition of Method Chunks

Every activity in SUPERSEDE begets a method chunk for the catalogue. We consider that the description of the activity is the process part of the chunk, while the output of such activity (i.e., the list of produced artefacts) is the product part. The situation of the method chunk is given by the list of consumed artefacts, while the intention needs to be explicitly given in the form of a goal. We also include the definition of the chunk reuse context, which has been defined in [8] as a taxonomy of criteria (details are given in Section 5.2); not all the criteria apply to every chunk. Finally, we extend the definition of method chunks with the Role and Tool attributes representing the corresponding SUPERSEDE method elements and presented in the section above.

Fig. 3 shows the metamodel corresponding to the definition of method chunk. For convenience, several derived roles are explicitly declared which can be computed from roles appearing in Fig. 2. Also an associate association class is introduced as derived from the corresponding activity-binding association class appearing in Fig. 1.

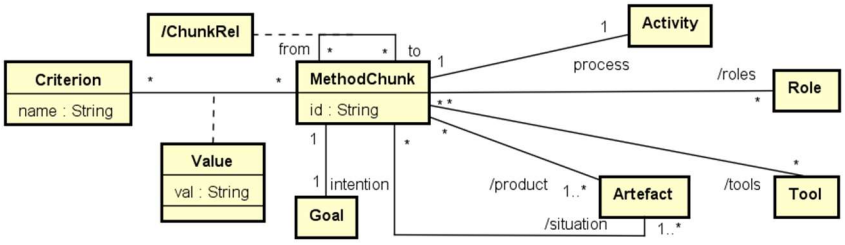

Fig. 3. Metamodel for method chunks

5 A Catalogue of Method Chunks for SUPERSEDE

In this section, we present the method chunks of the SUPERSEDE method that we have developed following an ad-hoc approach and applying the method chunk metamodel shown in Fig. 3. The context criteria play a crucial role in order to understand in which situations the chunks are applicable; therefore, we start by detailing them.

5.1 Context Criteria applicable in the SUPERSEDE Method

As reported in Sections 3.1 and 4.2, context definition using SME is implemented by the definition of context criteria that capture the factors that may influence in the selection of a method chunk for a particular instantiation of a method. In [8], a proposal is given resulting in two relevant outcomes: 1) a 3-tier context structure in which a few dimensions are decomposed into facets, and the facets into criteria; 2) a concrete proposal of this context structure for the information systems field by 2006.

We have evolved the original proposal along two different directions: first, to reflect the evolution on the systems engineering field in these last ten years; second, to fit it to the SUPERSEDE context. Table 1 presents an excerpt of the result, focusing on those criteria that are the most relevant for chunk selection. Criteria with an (*) are new, mostly related to the data-driven approach adopted in the SUPERSEDE method. Range of values adhere to the original proposal as much as possible; when subjective, value assignment relies in expert criteria. Definition of the relevant criteria follow:

- *User involvement*. Related to the participation of the user in the chunk's activity. Relevant for activities related to feedback gathering (end-user involvement) but also for technical ones (e.g., involvement of project managers in release planning).
- *Resources required* (evolution of *Means shortage* in [8]). To measure the complexity of an activity with respect to personnel, infrastructure, etc. It affects especially activities that require code development (e.g., new monitoring components).
- *Size* (of the project). Size may impact activities whose feasibility depends on the volume of work. A typical example would be prioritization techniques (e.g., AHP comparison) that do not scale well with a large number of requirements.
- *Privacy*. Data-driven methods like SUPERSEDE may be highly constrained in the case of serious privacy restrictions due to regulations, national laws, etc. Remarkably, tracing user's behaviour may be particularly problematic.
- *Scope*. The scope of the project is relevant for different reasons. For instance, processing feedback in worldwide application is problematic because this feedback will be expressed in different languages making it difficult to process.
- *Delivery strategy*. The type of delivery impacts the later stages of evolution. Heavyweight prioritization techniques are cumbersome to use with frequent releases, and classical release planning does not apply in continuous delivery contexts.
- *Accuracy*. The level of accuracy sought impacts on the selection and customization of some techniques. For example, as we will see in Section 5.3, different prioritization techniques exhibit different accuracy levels.
- *Type of end-user*. Some users may be more educated than others from an IT consumer perspective. It may be expected that a technician using the feedback gathering mechanisms will be more accurate than a regular citizen.
- *Motivation*. In data-driven approaches, one strategy to get more data is to motivate stakeholders to make them willing to participate actively. Not only end-users, but also technical stakeholders may be involved using gamification techniques.
- *Business domain*. The business domain has several consequences. For instance, a domain as smart cities is likely to produce tons of data from sensors that may help eliciting requirements for next releases.

Table 1. Criteria for SUPERSEDE reuse context

Criterion	Values
User involvement	Low, Medium, High
Resources required	Few, Fair, Much
Size	Small, Medium, Large
Privacy (*)	Very sensible, Sensible, Not sensible
Scope (*)	Narrow, Wide, Multi-national
Delivery strategy	Spare releases, Frequent releases, Continuous delivery
Accuracy (*)	Low, Medium, High
Type of end-user (*)	Citizen, Organization, Technician
Motivation (*)	Low, Medium, High
Business domain	Smart cities, Banking, …
System architecture (*)	Desktop, Web-based, Mobile, …
Other assumptions (*)	Minor, Fair, Major

- *System architecture*. The type of system architecture may impact on the complexity of development. For instance, a monitoring tool for a web client may require less effort to implement than the same instrument for mobile devices.
- *Other assumptions*. This criterion collects very specific conditions not belonging to the previous types that need to be fulfilled in order to apply the chunk. They may refer to the organization, adopted techniques, involved roles, etc.

5.2 A Catalogue of Method Chunks

We group the chunks in four categories. For all the categories, an initial chunk (Chu-XXX-01) represents an initial deployment activity, which is sometimes decomposed into subactivities. We omit these ones for brevity and present below the rest of chunks and their relationships (see Fig. 4, where "e-s" stands for "end-start" relationships).

Chunks for collection. Data enters the SUPERSEDE process in two different ways. First, end users may provide feedback using multi-modal mechanisms configured at design time but also at runtime (although runtime issues, belonging to the self-adaptation world, are not included in this paper) (Chu-Col-02: Feedback Collection). Second, SUPERSEDE may monitor data without users' explicit intervention (Chu-Col-03: Monitoring Data Collection). There are three different types of monitored data: quality of service, e.g. response time or availability (Chu-Col-03a: QoS Monitoring); monitoring of social networks as Twitter (Chu-Col-03b: Social Network Monitoring); users' usage monitoring (Chu-Col-03c: Usage Monitoring). The collected data are sent for analysis following the SUPERSEDE loop.

Chunks for analysis. The SUPERSEDE data-driven approach relies on the conceptualization of relevant ideas using a domain ontology defined by a data steward who mediates with domain experts from the organization (Chu-Ana-02: Domain Ontology Definition). This ontology is used as the basis to define the event recognition rules (Chu-Ana-03: Definition of Event Recognition Rules) that will capture needs for evolution as well as the logical definition of data coming from feedback and monitoring (Chu-Ana-04: Source Ontology Extraction). Collected data are processed in a last chunk

(Chu-Ana-05: Event and Pattern Detection) that applies the recognition rules to the monitored data coming from Chu-Col-02 and Chu-Col-03 and produces the real needs in the form of events and patterns to be decided upon in the next phase.

Chunks for decision. The alerts produced by Chu-Ana-5 are captured by a method chunk that converts them into a list of requirements by means of collaborative editing involving some selected requirements analysts (Chu-Dec-02: Requirements Collaborative Editing). Since they have been produced independently, it may be the case that requirements are overlapping or present other relationships, therefore a consolidation of the list is made right away (Chu-Dec-03: Requirements Similarity Check). This consolidated list is then prioritized involving again the appropriate (or available) set of stakeholders, to generate a prioritized list of requirements (Chu-Dec-04: Requirements Prioritization) to be processed by the enactment method chunks. There are several strategies for prioritizing requirements, and we present here three of them (Chu-Dec-04a, Chu-Dec-04b and Chu-Dec-04c, using AHP, Gamified AHP and Genetic Algorithms, respectively) which differ in some context criteria (see details in next subsection).

Chunks for enactment. The prioritized list of requirements is processed by a release planning activity (Chu-Ena-02: Release Planning) that considers also the list of available resources as input, and then produces a release plan.

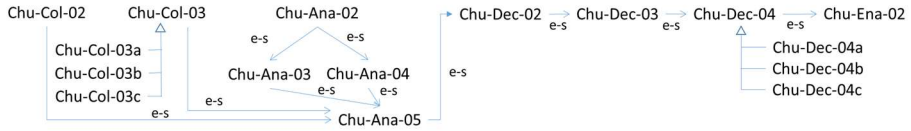

Fig. 4. Relationships among SUPERSEDE evolution method chunks.

5.3 Example: Requirements Prioritization

For the sake of illustration, in this subsection we present a method chunk in detail, namely the chunk for requirements prioritization, together with a summary of its three current specializations. Requirements prioritization elaborates on a set of requirements produced by a previous activity (Chu-Dec-03, which needs to be finished before starting this one), and applies a set of weighted criteria to the information elicited from several stakeholders, as to provide a list of prioritized requirements that serves as input of the release planning activities. The SUPERSEDE method applies a gamification approach to prioritization, and a software tool called DMGame has been developed with this purpose. Three types of stakeholders collaborate around this tool: a Supervisor supervising the process, a Negotiator mediating in conflicts and the Decision Providers that provide the necessary information for prioritizing. It is worth remarking that this chunk has not context criteria applicable, meaning that it will be instantiated in any possible situation. We expect this to be the usual situation in abstract method chunks, leaving the criteria to the specializations as shown below.

We have built three different specializations for this abstract chunk until now. Two of them are based on the AHP method, a third one on genetic algorithms. As a specialization, the information is inherited, thus only new information needs to be added.

Table 2. Method chunk for requirements prioritization (DM: Decision-Making)

Method chunk	Content
Id	Chu-Dec-04 [abstract]
Name	Requirements Prioritization
Description	This activity applies some selected technique in order to prioritize a list of requirements involving several selected stakeholders
Context	--
Situation	• Set of (possibly interrelated) requirements • Set of weighted criteria for the prioritization
Intention	Prioritize requirements in a collaborative way
Process part	Description of the activity – not included for brevity reasons
Product part	LP: List of prioritized requirements
Roles	• DM Supervisor: Organizes the full setting of the prioritization and supervises the execution of the activity • DM Negotiator: Facilitates the resolution of conflicts among decision providers • Set of DM Decision Providers: Provides information useful for prioritization
Tools	DMGame: decision-making web-based tool
Related chunks	Chu-Dec-03 e-s Chu-Dec-04

In the case of these three specializations, only the context is concerned. The context criteria that affect the selection of the specialization are shown in Table 3, with the values that apply to every specialization. As a short summary, AHP-based methods require more user involvement and thus more resources and also suffer from limitations on the number of requirements to handle and from the assumptions needed to ensure the accuracy of the method (which under these assumptions, is higher than the case of genetic algorithms). Although not critical, there are some differences also in the type of end-users and the delivery strategy. Concerning the two AHP methods, the main difference among them is the higher motivation required for AHP.

Table 3. Context criteria for the three specializations of requirements prioritization.

Criterion	AHP	Gamified AHP	Genetic
User involvement	High	High	Medium
Resources required	High	High	Medium
Project size	Small	Small	Medium
Delivery strategy	Frequent	Frequent	Continuous
Type of end-user	Technician	Technician	Mixed
Accuracy	High	High	Fair
Motivation	High	Medium	Medium
Assumptions	High	High	Low

6 Steps for adopting SUPERSEDE: A Situational Approach

It is not sufficient to transform a method into a collection of method chunks to make it situational. We need to provide guidance for creating situation-specific SUPERSEDE methods and allowing companies to tailor SUPERSEDE to their needs. For that, we follow the generic three-step assembly process mentioned in Section 3, that we specialise for SUPERSEDE as follows (see Fig. 5). The formalization of the process is just outlined; we cannot be developed in full due to space reasons.

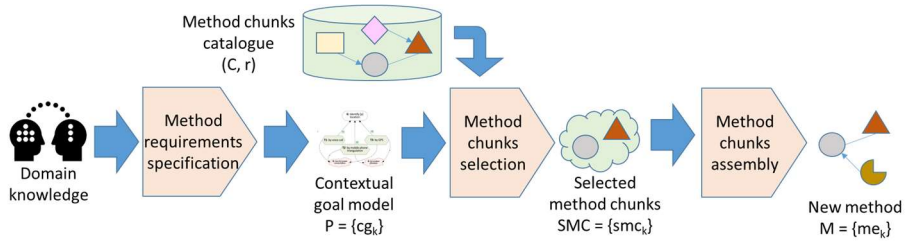

Fig. 5. The three-step assembly process for the SUPERSEDE method tailoring.

Method requirements specification. The first step consists in defining functional and contextual method requirements. Functional requirements capture a set of engineering intentions that shall be fulfilled by the new method, while the contextual ones reflect the situation of the project at hand in terms of assessed criteria from the reuse context (see Table 1).

Identifying method requirements is not an easy task, and the SME literature only provides very generic recommendations. For SUPERSEDE, we propose to use goal models (represented as *i** models [20]) as well as the domain knowledge to extract method intentions and to assess context criteria. Indeed, a goal model interconnects the main actors via goal dependencies and allows to derive method intentions from these goals. We recommend, to construct first the as-is goal model reflecting the current situation of the organization. Then, the envisioned situation is designed as a to-be goal model. For the to-be situation, we propose to link intentional elements to relevant context information that allow identifying the most pertinent context criteria (e.g. user involvement, privacy, delivery strategy, etc.) and, therefore, to derive not only method intentions but also contextual requirements. The result of this step can be stated then as a set of method requirements, namely contextual goals, $P = \{cg_k\}$ where every method requirement is a pair $cg_k = (g_k, C_k)$ of a goal and a set of conditions of the form $C_k = \{(c_i, v_i)\}$, being c_i a contextual criterion and v_i a valid value for such criterion. Note that C_k is a correspondence, allowing thus different values for a criterion.

Method chunks selection. In the second step, the method requirements are used to select method chunks by matching them with method chunk components. The selection query is the model P above which is compared against the catalogue (C, R) of method chunks, $C = \{mc_k\}$, and their relationships, $R = \{(mc1_i, mc2_i, tr_i)\}$, being $mc1_i$ and $mc2_i$ two method chunks and tr_i a type of relationship valid for them. For the selection process, we consider only as relevant information of method chunks the intention and the context, $mc_k = (int_k, cont_k)$, where $cont_k$ has a similar structure than C_k above.

The matching among P and (C, R) can then be defined as a set of selected method chunks $SMC = \{smc_k\}$ where each $smc_k = (int_k, cont_k)$ fulfils several conditions: 1) the intention of the chunk satisfies some functional requirement g_k of the query, $int_k \Rightarrow g_k$; 2) the context criteria of such functional requirements are satisfied by the chunk, $cont_k \Rightarrow C_k$; 3) the intention of the chunk does not violate any other functional requirement of the query, $\neg(int_k \Rightarrow \neg g_i)$. Please note that we do not consider relationships in the step, but in the next one.

Ideally, the selected method chunks should cover all functional method requirements and satisfy the context conditions. In case some functional requirement is not satisfied, other method sources will be explored and formalised as method chunks to fill the gap (see next step). On the contrary, if there are several method chunks satisfying the same requirement (i.e. producing the same outputs in different ways) two possibilities are to be considered: (1) selecting only one of these chunks, so the decision is taken by the method engineer, or (2) postpone the decision to method enactment time (see next step).

Method chunks assembly. The last step consists in assembling the selected method chunks into a coherent method. In the case of SUPERSEDE, that will mainly consist in defining the order of chunks execution based on their input and output artefacts and complete missing elements by ad hoc (eventually non-reusable) chunks.

Let M be the method corresponding to the SUPERSEDE instantiation under construction, $M = \{me_k\}$, being me_k an instance of the MethodElement abstract class introduced in Fig. 1. The rules to be applied are:

1. Connect the selected method chunks according to the relationships in R: $\forall(m1, m2) \in SMC: (m1, m2, t) \in R \Rightarrow (m1, m2, t) \in M$. We assume that this inclusion in M takes care also of including and connecting artefacts, roles and tools as specified in the metamodel.
2. For those contextual goals $\{cg_k\}$ in P not covered by any method chunk in SMC, explore other method sources and formalise them as method chunks to fill the gap, applying the steps above as required.
3. For those contextual goals $\{cg_k\}$ in P covered by more than one method chunk in SMC, providing arguments for the inclusion in M of either only one of them (if they are exclusive) or a subset (if they are complementary).

7 Example

We present an illustrative example of the application of the SUPERSEDE method to the SIEMENS project use case, which concerns the development of a smart-city platform (Eco Sys. Platform). A department in SIEMENS is responsible to evolve the platform, staying in continuous contact with app developers and utility devices producers, which are the main platform users. Moreover, the platform usage is constantly monitored to assess and predict mid- to long-term service level agreement compliance.

Method requirements specification. A SIEMENS department (partner in the SUPERSEDE project) has adopted the SUPERSEDE method as shown in Fig. 6, which depicts an excerpt of the *i* to-be* goal-oriented model for this use case (see [5] for the full version). The Project Manager relies on the SUPERSEDE method for achieving key goals that will allow improving the way the department evolves and maintains the Eco Sys. Platform, in relation to feedback analysis, collaborative decision-making and release planning. The Development Team relies on SUPERSEDE method to improve collaborative decision-making. Eco Sys. Platform requires SUPERSEDE to ensure privacy compliance. The goal diagram of SUPERSEDE actor models the goals that the method has been delegated to achieve, which are refined using decomposition (just hinted in the figure).

The SIEMENS department can perform different, and possibly simultaneous projects. Each project is characterised by specific values of context properties (depicted as a simple list of item in the rectangular shapes in Fig. 6) which need to be mapped to contextual criteria (Table 1). For instance, in *Project1*, which concerns the implementation of a small set of new requirements to improve the platform reliability, privacy compliance is not critical, while in *Project2*, which deal with a larger set of requirements related to the management of usage logs, privacy compliance is highly relevant. The analysis of the resulting contextual goal model will lead to the identification of a set of method requirements, such as $cg_{1.1}$ = (G1.1, $C_{1.1}$), where $C_{1.1}$ = {(user involvement, high) (size, small) (accuracy, high) (motivation, high)}, and $cg_{1.2}$ = (G3, $C_{1.2}$) where $C_{1.2}$ = {(privacy, sensible)} for the SUPERSEDE method configuration in *Project1*. Similarly, for *Project2,* method requirements include $cg_{2.1}$ = (G1.1, $C_{2.1.}$), where $C_{2.1}$ = {(user involvement, medium) (size, medium) (accuracy, fair) (motivation, medium)}, and $cg_{2.2}$ = (G3, $C_{2.2}$) where $C_{2.2}$ = {(privacy, very sensible)}.

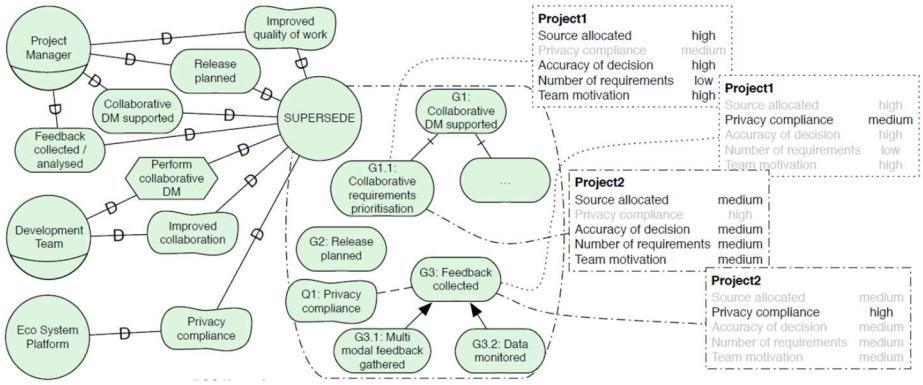

Fig. 6. To-be contextual goal model with information of two projects (simplified example)

Method chunks selection. These method requirements drive the selection of method chunks. For instance requirements $cg_{1.1}$ matches to the method chunk Chu-Dec-04 (for the goal part, G1.1), while for the selection of the method variant we need to match $C_{1.1}$ with context criteria (see Table 3), resulting in the selection of the AHP variant (Chu-Dec-04a). Analogously, requirement $cg_{1.2}$ will lead to the identification of method chunks Chu-Col-2 (because privacy is not very sensible), Chu-Col-03a, and Chu-Col-03c. On its turn, for *Project*$_2$ the genetic algorithm-based prioritization technique Chu-Dec-04c will be selected, due to the increasing number of requirements). Concerning data collection, the analysis of context condition $C_{1.2}$ leads to exclude the use of both Chu-Col-02 and one specialization of Chu-Col-03, Chu-Col-03b, because neither feedback gathering nor usage monitoring are considered appropriate when privacy is a highly relevant issue (which is explicitly stated in the corresponding chunks). In both projects, Chu-Ena-02 (Release Planning) is selected due to the Release Planned goal.

Method chunks assembly. On the one hand, the selected method chunks are now assembled taking into account execution order (see Fig. 4) and input / output artefacts. This will create the assembly (Chu-Dec-04a, Chu-Ena-02) in *Project1* and (Chu-Dec-04c, Chu-Ena-02) in *Project2*. However, collection chunks cannot be assembled because the

analysis chunks have not been selected. The reason is that the goal model does not include information enough as to realize that these chunks are needed. Therefore, they need to be selected in this phase and assembled correspondingly, e.g. (Chu-Col-02, Chu-Ana-05) in *Project1*.

8 Conclusions and Future Work

In this paper, we have provided a systematic definition of the SUPERSEDE method for data-driven software evolution oriented towards its customization in particular contexts. We have defined a metamodel for method chunks built upon activities, artefacts, roles and tools (RQ1), then presented a catalogue of method chunks for SUPERSEDE-based software evolution compliant to such metamodel (RQ2), we have defined a set of context criteria to describe the context in which every chunk can be selected (RQ3) and we have presented an SME-based process to guide the definition of a tailoring of the SUPERSEDE method in a particular context (RQ4), illustrating it with an example.

It is worth mentioning that the answer to RQ4 is a methodological contribution beyond the application to the SUPERSEDE method. It represents an evolution to recent SME works that we have undertaken [21][22] given the inclusion of context criteria as part of the method requirements. The full development of the formalization just outlined in this paper, as for instance the applicability of previously proposed context goal modelling approaches that support contextual annotations, e.g. [23], is future work.

A point of discussion is the possible complexity of the method. As in any SME-based approach, we assume that a method engineer would lead the customization of the method with the help of key stakeholders, e.g. a requirements engineer for the construction of the goal model. We plan also to develop tool support as to assist the method engineer, e.g. by suggesting the missing chunks in the third step of the assembly

One interesting issue to discuss is the fact that companies will rarely adopt an approach as SUPERSEDE independently of their current development methods. For instance, we are currently developing a Jira plug-in that allows connecting the requirement prioritization and releasing planning functionalities to this tool. This brings an interesting aspect to the method that needs to be brought in the form of new chunks.

In connection with the point above, an aspect that has not been discussed is the extent to which a company may adopt the SUPERSEDE method. If it is at the level of the company, or one particular area, type of software/project, or even at the extreme only for one project. As presented here, our approach has focused more on the project level, but in the example we have combined with an organizational level (to have a single to-be model for the organization, with project information as context). Going further in this direction is definitively another open line of research.

As additional future work, we aim at adding the dynamic adaptation side of SUPERSEDE to this evolution approach. In SUPERSEDE, there are activities related to system self-adaptation and configuration that are integrated with the evolution activities. Including them in the SME approach would provide a holistic view to data-driven software engineering using the SUPERSEDE control loop.

Acknowledgments

This work is a result of the SUPERSEDE project, funded by the EU's H2020 Programme under the agreement number 644018.

References

1. Wang, X., Guarino, N., Guizzardi, G., Mylopoulos, J.: Software as a Social Artifact: A Management and Evolution Perspective. ER 2014.
2. Mens, T., Serebrenik, A., Cleve, A. (eds.): *Evolving Software Systems*. Springer, 2014.
3. Lehman, M.M.: Programs, Life Cycles, and Laws of Software Evolution. *Proceedings of the IEEE*, 68(9), 1980.
4. Henderson-Sellers, B., Ralyté, J. Ågerfalk, P., Rossi, M.: *Situational Method Engineering*. Springer, 2014.
5. Stade, M., et al.: D3.1: Requirements for Methods and Tools. SUPERSEDE project deliverable, 2015. Available at www.supersede.eu.
6. Ralyté, J., Rolland, C.: An Approach for Method Reengineering. ER 2001.
7. Ralyté, J., Deneckère, R., Rolland, C.: Towards a Generic Model for Situational Method Engineering. CAiSE 2003.
8. Mirbel, I., Ralyté, J.: Situational Method Engineering: Combining Assembly-based and Roadmap-driven Approaches. *Requirements Engineering Journal*, 11, 2006.
9. Ralyté, J.: Towards Situational Methods for Information Systems Development: Engineering Reusable Method Chunks. ISD 2004.
10. Brun, Y. *et al*.: Engineering Self-Adaptive Systems through Feedback Loops. In *Software Engineering for Self-Adaptive Systems*, Springer, 2009.
11. Stade, M., Fotrousi, F., Seyff, N., Albrecht, O.: Feedback Gathering from an Industrial Point of View. RE 2017.
12. Nadal, S. *et al.*: A Software Reference Architecture for Semantic-aware Big Data systems. *Information & Software Technology*, 90, 2017.
13. Morales-Ramirez, I., Kifetew, F.M., Perini, A.: Analysis of Online Discussions in Support of Requirements Discovery. CAiSE 2017.
14. Guzmán, E., Alkadhi, R., Seyff, N.: An Exploratory Study of Twitter Messages about Software Applications. *Requirements Engineering Journal* 22(3), 2017.
15. Busetta, P. *et al*: Tool-Supported Collaborative Requirements Prioritisation. COMPSAC 2017
16. Ameller, D. *et al.*: Replan: A Release Planning Tool. SANER 2017.
17. Ameller, D. *et al.*: Towards Continuous Software Release Planning. SANER 2017.
18. Object Management Group (OMG): Software & Systems Process Engineering Meta-Model Specification (SPEM), Version 2.0. Technical Report, April 2008.
19. Ribó, J.M., Franch, X.: A Precedence-based Approach for Proactive Control in Software Process Modelling. SEKE 2002.
20. Dalpiaz, F., Franch, X., Horkoff, J.: iStar 2.0 Language Guide. Available at https://arxiv.org/abs/1605.07767
21. López, L. *et al*.: OSSAP - A Situational Method for Defining Open Source Software Adoption Processes. CAiSE 2016.
22. López, L. *et al*.: Agile Quality Requirements Management Best Practices Portfolio: A Situational Method Engineering Approach. PROFES 2017.
23. Ali, R., Dalpiaz, F. and Giorgini, P.: A Goal-based Framework for Contextual Requirement Modeling and Analysis. *Requirements Engineering Journal*, 15(4), 2010.