CSEPM - A Continuous Software Engineering Process Metamodel

Stephan Krusche
Technische Universität München
Munich, Germany
krusche@in.tum.de

Bernd Bruegge
Technische Universität München
Munich, Germany
bruegge@in.tum.de

***Abstract*—Software engineers have to cope with uncertainties and changing requirements. Agile methods provide flexibility towards changes and the emergence of continuous delivery has made regular feedback loops possible. The abilities to maintain high code quality through reviews, to regularly release software, and to collect and prioritize user feedback, are necessary for continuous software engineering (CSE). However, there exists no software process metamodel that handles the continuous character of software engineering.**

In this paper, we describe an empirical process metamodel for continuous software engineering called CSEPM, which treats development activities as parallel running workflows and allows tailoring and customization. CSEPM includes static aspects that describe the relations between specific CSE concepts including reviews, releases, and feedback. It also describes the dynamic aspect of CSE, how development workflows are activated through change events. We show how CSEPM allows to instantiate linear, iterative, agile and continuous process models and how it enables tailoring and customization.

***Keywords*-Process Model, Agile Methods, Feedback, Release, Review, Change, Event, Workflow**

I. Introduction

The term "software engineering" (SE) was first proposed in the 1960s [1]. It "was deliberately chosen as being provocative, in implying the need for software manufacture to be [based] on the types of theoretical foundations and practical disciplines[,] that are traditional in the established branches of engineering" [2]. The vision was to move from unstructured ways to a defined software life cycle model to understand and characterize how software is developed. By adapting successful methods of other engineering domains, the goal was to industrialize the creation of software with a defined process that is well understood, predictable, repeatable and efficient [3].

Detailed process models emerged with a strong emphasis on planning and the goal to describe the process of engineering software, e.g. Royce's linear waterfall process model [4]. He transfered the sequential order of activities in production lines to form a linear development approach. This resulted in a defined software process following strict rules and avoiding deviations, which were seen as errors that need to be corrected.

In the following years, software developers increasingly recognized that the essence of SE is to deal with changes and that defined process models are not capable of addressing this need. SE consists of experimental knowledge work where creativity is important [5]. Such work includes unexpected events, incidents and uncertainty. Lehman realized in the 1980s that software evolution, the continual change to a software system, is inevitably required to keep software up to date with changing environments and to satisfy stakeholders [6].

In the 1990's, agile methods such as Scrum [7] emerged with the philosophy that SE should not follow a defined process model but rather an empirical model that is not entirely planned. The principles in the agile manifesto include customer satisfaction through early and continuous delivery and the welcome of changing requirements, even late in the project [8]. Based on continuous integration, Jez Humble defined a model for continuous delivery in 2010 [9]: the goal is to keep software in a state so that changes to it can be released at any time. Regular releases lead to an improved product quality and high customer satisfaction [10].

However, Humble's model does not describe how to continuously handle changes and user feedback. In fact, Rodríguez and her colleagues identified "a clear research gap [...] for mechanisms to use customer feedback in the most appropriate way so that information can be quickly interpreted" in their systematic mapping study in 2016 [11]. Jan Bosch introduced the term continuous software engineering (CSE) [12]. However, to our best knowledge, there has not yet been an attempt to model SE continuously. This is the focus of this paper.

The objective is to create a software process metamodel that describes the continuous and unexpected nature of SE through the idea of workflows that can be interrupted by change events. The process metamodel improves the understanding of CSE because it generalizes the essential concepts and activities in a standard and simplifies the communication. It allows the instantiation of multiple process models and facilitates tailoring and customization. This is important because processes have to be adapted to concrete project environments.

The paper is organized as follows. Section 2 describes software process models as the foundation of this work. In Section 3, we introduce CSEPM including its static and dynamic aspects. Section 4 shows how CSEPM can be instantiated with linear, iterative and agile process models. In Section 5, we describe Rugby [13] as one exemplary instance of a continuous process model. Section 6 relates CSEPM to existing process metamodels and Section 7 concludes the paper.

To appear at RCoSE 2017

II. Software Process Models

The term process is used in various contexts and can be defined as "a related set of activities conducted to the specific purpose of product definition" [14]. A process model describes how activities must, should or could be performed in contrast to the process itself which is what really happens. A process is an instantiation of a process model and includes workflows describing work practices in the project. A workflow is a thread of cohesive and mostly sequential activities performed by project participants that produce artifacts [15]. Designing a process model is related to process meta modeling [14].

Process control deals with mechanisms for maintaining the output of a process in a specified and desired range. Control does not mean the process can be completely predicted. One goal of process control is the minimization of risks. If potential problems can be detected early, a reaction can be defined to solve the problem. There are different styles of process models distinguished by their process control, e.g. defined vs. empirical process control. A defined process is a collection of tightly coupled steps: the output of one step is the input to the next step [16]. The defined model handles changes and failures as deviations that need to be corrected and tries to prevent the occurrence of changes in the plan or outcome of the process. This type of process control is used in assembly productions as described by Taylor in the 1920s [17] for the manufacturing of industry products such as cars.

However, it cannot be used for building complex systems which require creative problem solving and adaptivity to change [15]. Software development is a complex process with random variables, that cannot be defined completely deterministic. "It is typical to adopt the defined (theoretical) modeling approach when the underlying mechanisms by which a process operates are reasonably well understood. When the process is too complicated for the defined approach, the empirical approach is the appropriate choice" [18]. Complex processes require therefore an empirical control model.

Empirical process control includes visibility, inspection and adaption. It allows to control complex processes, which cannot perfectly be defined and which would generate unrepeatable and unpredictable results. The empirical model handles changes and failures as opportunities. A quick reaction to changes can lead to advantages compared to competitors. Empirical process control allows to expect the unexpected. If something unexpected occurs, the process is inspected and potentially adapted.

There are three ways to adapt a process model: tailoring, customization and extension. Process tailoring is the adaption to operational needs on a more general level through the removal, modification or addition of workflows, without significantly deviating from the process model. According to Ginsberg, tailoring is the act of adjusting the definition of a general process description - the process model - to derive an alternative environment - the process [19].

Process customization is the adaption to operational needs on a more specific level through the removal, modification or addition of activities in an existing workflow, without significantly deviating from the workflow model of the process. Process extension is the adaption to operational needs through the addition of a new workflow or activity that cannot be described with existing elements of the process model.

III. Continuous Software Engineering Process Metamodel (CSEPM)

In this section, we introduce CSEPM, an empirical software process metamodel. Figure 1 shows CSEPM integrated into the meta object facility (MOF) with four layers and examples. The topmost layer M3 consists of the meta metamodel with the most abstract concept of a class.

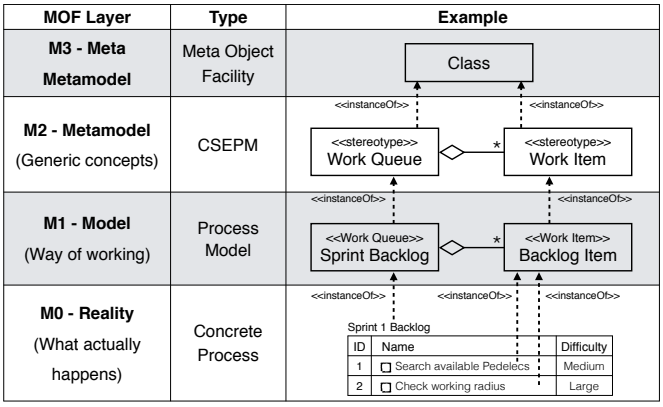

Fig. 1. CSEPM in the meta object facility layers

CSEPM resides in layer M2. An example of a generic concept on M2 is the *Work Queue*, an abstract representation of an ordered list of *Work Item*s with a priority. On layer M1, there can be different process models that describe the way of working in a software development project. An example of a concrete concept on layer M1 is the *Sprint Backlog* in Scrum which is an instantiation of the *Work Queue* on layer M2. The sprint backlog consists of *Backlog Item*s, which are instantiations of *Work Item*s. Instantiations are represented using the stereotype notation, e.g. *«Work Queue»*, which is similar to an inheritance relationship.

Concrete processes used in projects are instances of process models and reside on the M0 layer, that describes what actually happens in the reality. An example would be the real *Sprint 1 Backlog* of a project written on a board or stored in a tool. The example in Figure 1 includes two concrete backlog items "Search available Pedelecs" and "Check working radius" with specific values for attributes such as ID and difficulty.

A. Static Model

Figure 2 shows the static core model of CSEPM[1]. The workflow model includes the *Work Queue*, a prioritized list of *Work Item*s, that can be larger activities or small tasks, modeled with the composite pattern. New elements are sorted into the queue after their priority and urgency. If priority or

[1]For readability reasons, some model elements (e.g. roles) and unimportant attributes and methods are excluded.

urgency of a work item change, the element can be moved in the queue. There might be cases where a team member decides to start with a different work item than the first one in the queue, e.g. due to dependencies or due to capacity reasons. Work items can depend on each other and can have different states, such as ready, running or finished.

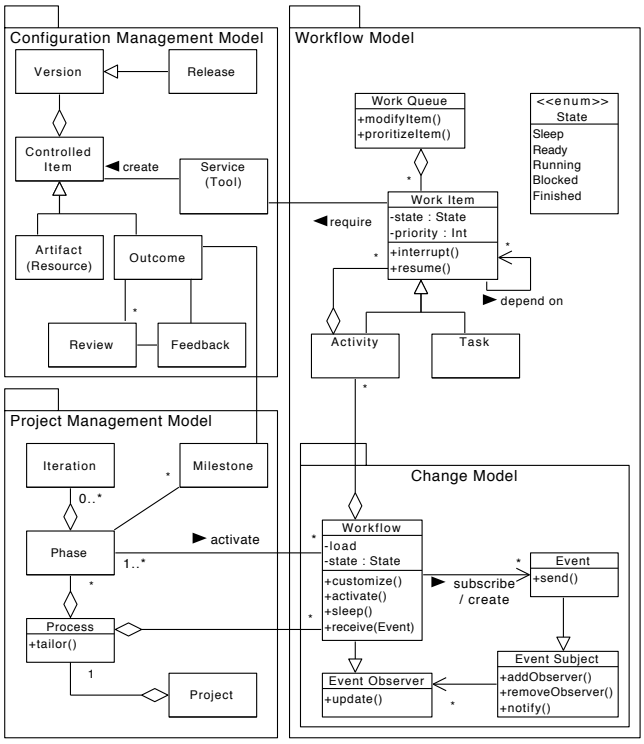

Fig. 2. CSEPM's core concepts and relationships as UML class diagram

The change model includes the workflow as an *Event Observer*, which can subscribe to certain *Event*s and also create events that trigger the activation of other workflows. A workflow belongs to the *Process* of the *Project*. The process consists of multiple *Phase*s. There are process models, in which certain workflows are only activated in specific phases. Each phase can have *Milestone*s, e.g. an important meeting where the continuation of the project is decided or where an *Outcome* of the project is discussed, e.g. the documentation of the requirements analysis phase.

*Controlled Item*s are outcomes of work items created by using *Service*s (*Tool*s) and have a *Version*. The same document might exist in multiple versions after it was changed due to feedback. Certain outcomes (e.g. builds) are combined into a *Release*, which is a specific version. An outcome can be subject of a *Review* that leads to *Feedback* on the outcome.

B. Change Model

CSEPM includes different *Event* types. An event is the generic form of a change, can trigger interruptions of the current workflow and can lead to the activation an another workflow. This is modeled with the observer pattern. Figure 3 shows a simplified version of the event taxonomy with some exemplary events. The event model is extensible: events can be added, changed and removed through process tailoring, workflow customization or the use in different application domains. New events can even be added during a project.

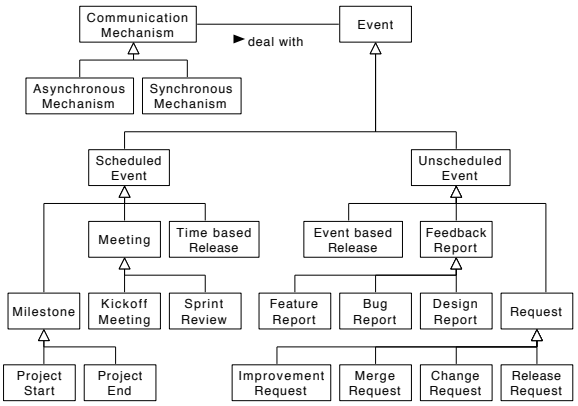

Fig. 3. CSEPM's change model including an event taxonomy for exemplary scheduled and unscheduled events. The model is extensible with new events.

The change model distinguishes between scheduled events (e.g. *Meeting*s) and unscheduled events (e.g. a *Bug Report* or *Change Request*) and provides a unified model for both. An event only notifies a workflow if it is interested, i.e. if it subscribed to the event. Workflows create events and notify other workflows when they are active. For instance, a bug report is interesting for the implementation workflow, while a feature request is interesting for the analysis workflow. An important distinction in CSEPM is made between time based and event based releases. Time based releases are scheduled, e.g. at the end of a Sprint in Scrum. Event based releases are unscheduled, e.g. if developers need feedback, customers request a new release or if a new feature is finished and should be delivered immediately.

C. Dynamic Model

The core of CSEPM's dynamic model is the workflow model. Figure 4 shows the dynamic view of CSEPM describing the control flow of workflows. In CSEPM, all workflows are started in the beginning of the project with the *Workflow Start* event and run continuously (potentially inactive) until the project end. After the project start, workflows subscribe to events they are interested in and then immediately sleep until an *Incoming Event* occurs which they have subscribed to. In the most extreme case, when the events, a certain workflow has subscribed to, never occur, then this workflow is never activated and sleeps until the project end.

If a subscribed event occurs, the workflow is activated and performs the work until the work is finished. Then, the workflow notifies other workflows about the finished work by sending an *Outgoing Event*. A specific event, which each workflow subscribes to, is the *Workflow Customization*. It is created, when the team wants to customize a workflow. Before the customization, the workflow is stopped, then customized and started again. The workflow might now subscribe to different events. At the project end, all workflows are stopped.

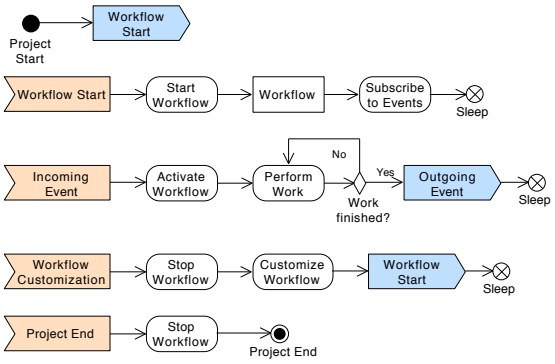

Fig. 4. CSEPM's core workflow model as UML activity diagram

Figure 5 shows the lifecycle model of a workflow. After the workflow has been started and has subscribed to events, it transitions to the *Sleep* state. Incoming events (orange) trigger the transition to the *Active* state. Workflows can be interrupted so that they are in the state *Blocked*. This happens e.g. if important information is missing that is necessary to perform the work, or if other more important work has to be done. If the problem is solved, e.g. the information is available, the workflow can resume and transition to the active state again. If the work is performed, the workflow omits an outgoing event (blue) and transitions to the sleep state again. The workflow transitions to the *Finished* state, when it is stopped, e.g. for customizations, or when the project ends.

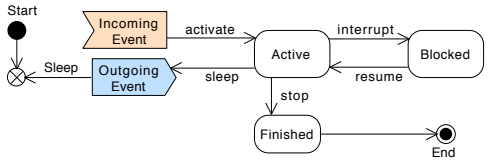

Fig. 5. The lifecycle model of a CSEPM workflow as UML state diagram

IV. CSEPM Instances

We show 3 instantiations of CSEPM, which reside on layer M1 in Figure 1, a linear, iterative and agile process model.

A. Linear Process Model

The first example of an instance of CSEPM is the linear waterfall model by Royce [4] which uses a defined process model control. The model has a one to one mapping between phases and workflows, e.g. the requirements elicitations phase corresponds to the requirements elicitations workflow. All phases produce an outcome and depend on a milestone of the previous phase that represents the end of the phase, e.g. the implementation phase can only start if the design phase is finished and the design document was completely realized. There is only one release produced after the testing phase that is further adapted in the maintenance phase.

Figure 6 shows how to model workflows in the waterfall model dynamically[2]. Requirements Elicitation starts directly after the *Project Start* event and lasts until the *Problem Statement* is realized which triggers the event *Requirements Elicitation finished* that activates the *Analysis* phase. Each workflow in the Waterfall model subscribes to only one event, which is triggered by the end of the previous phase, and ignores all other events. After activation, a workflow runs solely until its outcome was produced. Then the workflow sleeps and the next one starts.

[2]This is a simplified view that leaves out the transitions to previous phases that are also described in the initial publication by Royce [4]. A defined process can have a change control process as e.g. defined in [20].

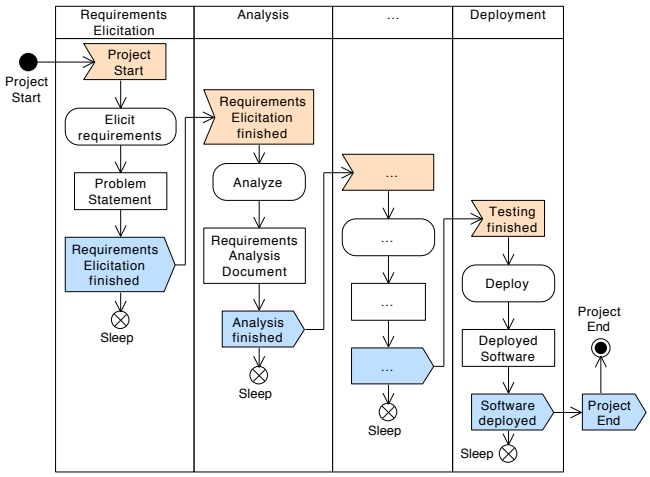

Fig. 6. Dynamic view of the Waterfall process model as UML activity diagram: example of a linear instantiation of CSEPM (adapted from [4])

B. Iterative Process Model

The second example is the iterative Unified Process by Jacobson et al. [21]. The dynamic view of the process model is shown in Figure 7.

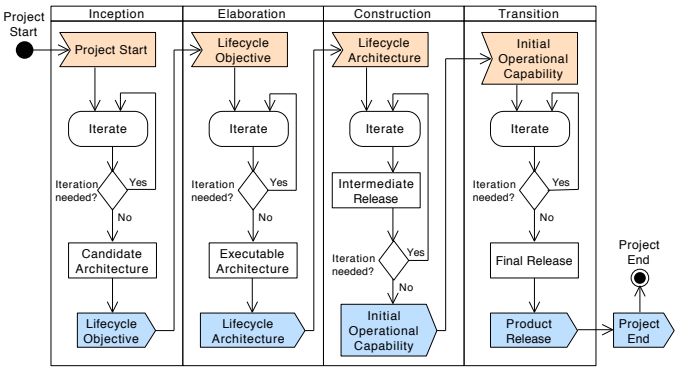

Fig. 7. Dynamic view of the Unified Process model as UML activity diagram: example of an iterative instantiation of CSEPM (adapted from [21])

It includes the four phases inception, elaboration, construction and transition. The *Inception* phase creates the *Candidate Architecture* as outcome and upon the *Lifecycle Objective* milestone, the *Elaboration* phase is activated that creates an *Executable Architecture Baseline* and lasts until the *Lifecycle Architecture* milestone. The *Construction* phase creates multiple *Intermediate Release*s, which are increments that include a specific functionality and build upon the last intermediate

release. This is usually the longest phase in the Unified Process. It finishes when the *Initial Operation Capability* milestone is reached. Then the *Transition* starts with the goal to deploy the release in the target environment and lasts until the *Final Release* is produced.

Milestones are modeled as events, which transition between two phases. Phases can be split into multiple iterations. The inception, elaboration and transition phases run until their respective outcome is finalized. In the construction phase, each iteration creates an intermediate release as outcome. At the beginning of the project, six core workflows and three supporting workflows are activated. Each phase has a specific emphasis on certain workflows, e.g. the inception phase focuses on business modeling and requirements.

C. Agile Process Model

The third example is the agile Scrum process by Schwaber and Beedle [16]. Development is completed in sprints (modeled as instance of phase) that produce *Product Increment*s as outcomes, which are reviewed during the *Sprint Review Meeting*. Each product increment builds upon the previous one and incrementally adds new features. The dynamic view of the process model is shown in Figure 8. After the project starts, the kickoff meeting is conducted, in which the product backlog is created. After the product backlog is finished, the event *Product Backlog created* leads to the start of the first sprint. Sprints are conducted until the project is finished.

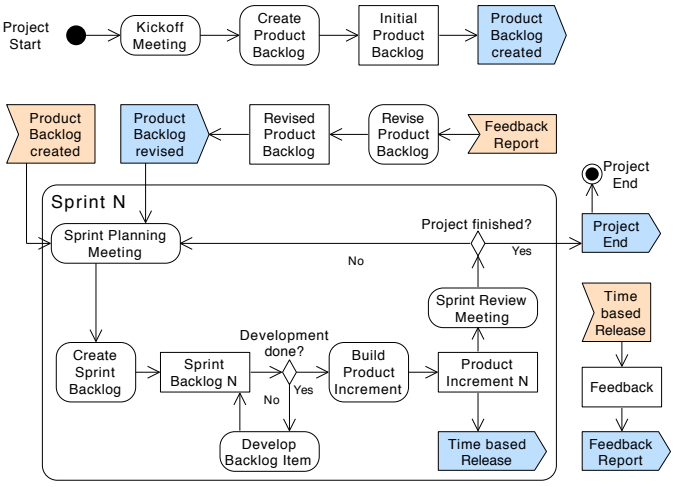

Fig. 8. Dynamic view of the Scrum process model as UML activity diagram: example of an agile instantiation of CSEPM (adapted from [16])

A sprint starts with a *Sprint Planning Meeting*, in which the sprint backlog is created as base for the development. The team meets daily while developing the backlog items. It builds the product increment, a time based release, before the sprint review meeting, which marks the sprint end. During this meeting, the time based release is presented and can lead to feedback which triggers the event *Feedback Report*. A feedback report can lead to a revised product backlog, that is used for the next sprint.

V. Rugby as CSEPM Instance

Based on Takeuchi and Nonaka [22], we use the term Rugby for a process model that uses agile concepts of Scrum [16] and the idea of iterative workflows of the Unified Process [21] to create a CSE process model. We first described Rugby in [13] and subsequently tailored it for industry projects [23]. Rugby's process model is described in detail in [24]. Rugby includes additional workflows for reviews [25], releases [13] and user feedback [26]. In this section, we show Rugby as an example of a continuous process model as instantiation of CSEPM.

A dynamic view of Rugby's process model is shown in Figure 9. Rugby defines the *Sprint 0* as an upfront project phase which customizes its three additional workflows for reviews, releases and feedback. Another goal in Sprint 0 (not shown in detail in Figure 9) is the creation of a shared understanding of the project requirements in the initial product backlog and of the software architecture. Typical deliveries in this phase, which takes 2 - 4 weeks depending on the experience of the team, are an empty review, an empty release, and an empty feedback to ensure the workflows are set up properly and everyone in the team is able to apply them.

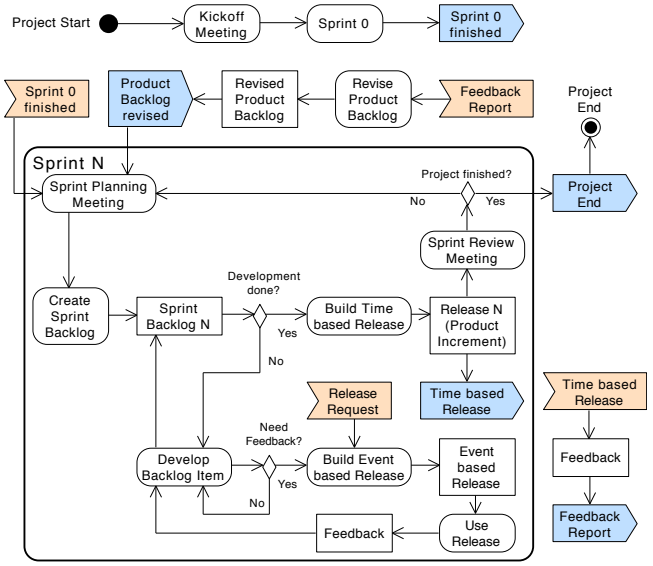

Fig. 9. Dynamic view of the Rugby process model as UML activity diagram: example of a continuous instantiation of CSEPM (adapted from [13])

After Sprint 0, the team works in development sprints as defined in Scrum and has to build a product increment before the sprint review meeting. Development sprints in Rugby have the same goal as in Scrum. The difference to Scrum is that in Rugby, the team can build event based releases and obtain feedback during the development sprint. In addition, Rugby uses a review workflow with merge requests as described in [25] and shown in Figure 10 in the details of the *Develop Backlog Item* activity.

Figure 11 shows the synchronization of parallel workflows in Rugby through change events. *New Backlog Item*s are identified through change requests or feedback reports and need to be prioritized and specified to be *ready for development*.

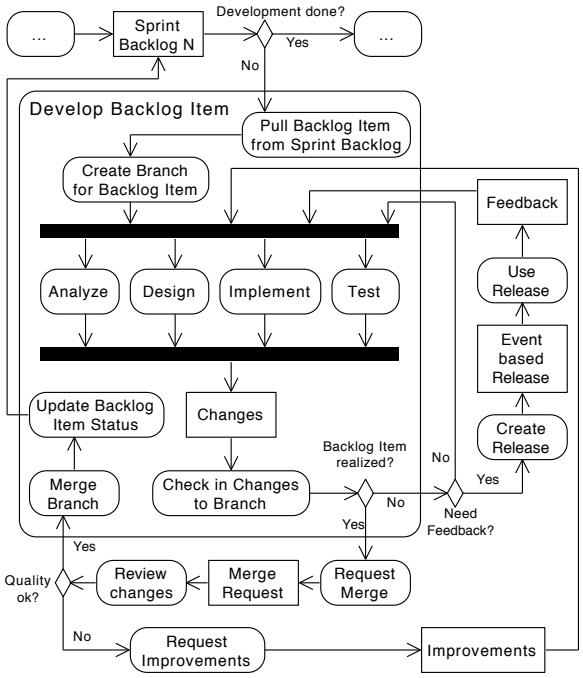

Fig. 10. Details of the activity Develop Backlog Item in Rugby

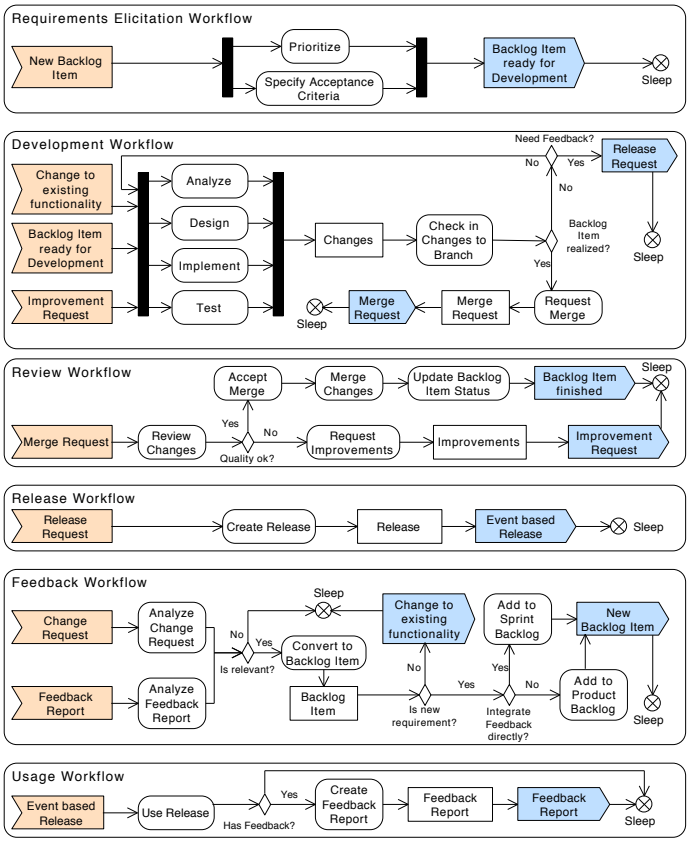

Fig. 11. Dynamic view of the synchronization of Rugby's parallel workflows through change events

This inserts them into the work queue. A developer can assign the backlog item to himself which activates the development workflow. The developer analyzes, designs, implements, and tests in parallel to produce source code changes. If the backlog item is realized, i.e. it fulfills all acceptance criteria, the developer requests a merge. This activates the review workflow where the reviewer either accepts the changes (if the quality of the changes meets the defined criteria) so that they can be merged, or where the reviewer requests improvements (e.g. if merge conflicts occur), so that development is activated again.

Another possibility is that the developer requests a release of the unfinished backlog item if he needs feedback or has a question. Then, the release workflow is triggered and creates an *Event based release* that can be used in the usage workflow. If the user has feedback, he creates a *Feedback Report*, which is analyzed in the feedback workflow and leads to a *Change to an existing functionality* or to a *New Backlog Item*.

The shown workflow synchronization in Figure 11 is an example of how workflows can be defined in a concrete process model. CSEPM also allows to tailor the process, e.g. by adding a new workflow for test driven development. It also enables workflow customization, e.g. by changing the review workflow to review every commit, or by changing the release workflow to continuous deployment.

Rugby is in use in multiple university courses including capstone courses [27] and interactive lecture-based courses [28]. It is also tailored in industry projects at a large German company Capgemini [23].

VI. Related Work

Steenweg and her colleagues found four systematic approaches for software process metamodels in their literature review [29]: (1) V-Model XT metamodel, (2) Software Process Engineering Metamodel (SPEM), (3) Software Engineering Metamodel for Development Methodologies (SEMDM) and (4) MetaME. In addition, (5) the Business Process Definition Metamodel (BPDM) is used to define business process models. In this section, we relate CSEPM to these five metamodels.

The V-Model XT metamodel [30] is based on the ideas of software process lines and modularity. Process variants can tailor, customize and extend the reference model which is similar to CSEPM's capability of tailoring, customization and extension. The V-Model XT metamodel focuses on artifact-oriented software processes and includes change procedures. It does not describe workflows and its lifecycles.

SPEM was released in 2002 by the Object Management Group (OMG) [31]. It is used to define and describe SE processes and their components. As CSEPM, SPEM includes a list of fundamental elements which are sufficient to describe different types of software process models. In contrast to CSEPM, it does not include specific features for project management. The goal of SPEM is to accommodate a large range of development methods and processes of different styles, cultural backgrounds, levels of formalism, lifecycle models, and communities. While SPEM includes milestones as important events, it does not include a workflow model, a change model or an event taxonomy.

SEMDM is an ISO standard based on UML [32] that addresses software process design. It is influenced by the idea of situational method engineering (SME) [33], the construction of methods which are adapted to specific situations of development projects. CSEPM's change model and its customization possibilities also allow to adapt to specific situations and provide more flexibility towards changes and feedback.

MetaME allows to create software engineering methods by combining product and process models [34]. It is based on SME and provides a proposal of how to create software engineering methods. As CSEPM, it includes a workflow model and describes different types of dependencies, but does not include change events or how to handle changes.

BPDM has been specified in 2008 by the OMG [35] as standard definition of concepts used to express business process models. The metamodel defines concepts, relationships, and semantics for the exchange of user models between different modeling tools. It includes interactions between independent business processes called choreographies which can be specified independently of its participants. As CSEPM, BPDM defines an event taxonomy and distinguishes between start and end events. It defines success events, failure events, abort events and error events. Similar to CSEPM, BPDM allows modelers to extend the event taxonomy with newly defined events. BDPM events are similar to CSEPM's change model where events trigger the activation of workflows. However, BDPM does not include SE related workflows.

VII. Conclusion

In this paper, we described CSEPM, an empirical process metamodel for continuous software engineering. CSEPM treats development activities as parallel running workflows. It includes static models that describe the relations between specific CSE concepts including reviews, releases, and feedback. With its workflow model, CSEPM focuses on dynamic aspects: how development workflows are activated and interrupted through change events. It consists of an extensible change model and treats changes as events that activate workflows.

While CSEPM also allows to instantiate linear and iterative process models, it was designed for agile and continuous process models and allows tailoring, customization and extension. It helps to understand the continuous and unexpected nature of software engineering. In the future, we want to provide a reference implementation and tool support.

References

[1] M. Mahoney, "The roots of software engineering," *CWI Quarterly*, vol. 3, no. 4, pp. 325–334, 1990.

[2] P. Naur, B. Randell, and J. Buxton, *Software engineering: concepts and techniques: proceedings of the NATO conferences*. Petrocelli, 1976.

[3] M. Fowler, "The new methodology," *Wuhan University Journal of Natural Sciences*, vol. 6, no. 1-2, pp. 12–24, 2001.

[4] W. Royce, "Managing the development of large software systems," in *Proceedings of IEEE WESCON*, 1970.

[5] V. Basili, "The role of experimentation in software engineering: past, current, and future," in *Proceedings of the 18th international conference on Software engineering*. IEEE, 1996, pp. 442–449.

[6] M. Lehman and L. Belady, *Program evolution: processes of software change*. Academic Press, 1985.

[7] K. Schwaber, "Scrum development process," in *Proceedings of the OOPSLA Workshop on Business Object Design and Information*, 1995.

[8] K. Beck, M. Beedle, A. Van Bennekum, A. Cockburn, W. Cunningham, M. Fowler, J. Grenning, J. Highsmith, A. Hunt, R. Jeffries *et al.*, "Manifesto for agile software development," *The Agile Alliance*, 2001.

[9] J. Humble and D. Farley, *Continuous delivery: reliable software releases through build, test, and deployment automation*. Pearson, 2010.

[10] L. Chen, "Continuous delivery: Huge benefits, but challenges too," *Software, IEEE*, vol. 32, no. 2, pp. 50–54, 2015.

[11] P. Rodríguez, A. Haghighatkhah, L. E. Lwakatare, S. Teppola, T. Suomalainen, J. Eskeli, T. Karvonen, P. Kuvaja, J. M. Verner, and M. Oivo, "Continuous deployment of software intensive products and services: A systematic mapping study," *Journal of Systems and Software*, 2016.

[12] J. Bosch, *Continuous Software Engineering*. Springer, 2014.

[13] S. Krusche, L. Alperowitz, B. Bruegge, and M. Wagner, "Rugby: An agile process model based on continuous delivery," in *Proceedings of the 1st International Workshop on Rapid Continuous Software Engineering*. ACM, 2014, pp. 42–50.

[14] C. Rolland, "Modeling the requirements engineering process," *Information Modelling and Knowledge Bases*, 1993.

[15] B. Bruegge and A. Dutoit, *Object Oriented Software Engineering Using UML, Patterns, and Java*, 3rd ed. Prentice Hall, 2009.

[16] K. Schwaber and M. Beedle, *Agile software development with Scrum*. Prentice Hall, 2002.

[17] F. W. Taylor, *The principles of scientific management*. Harper, 1914.

[18] B. A. Ogunnaike and W. H. Ray, *Process Dynamics, Modeling, and Control*. Oxford University Press, 1994, vol. 1.

[19] M. Ginsberg and L. Quinn, "Process tailoring and the the software capability maturity model," Carnegie Mellon University, Tech. Rep. CMU/SEI-94-TR-024, 1995.

[20] P. Bourque and R. Fairley, *Guide to the Software Engineering Body of Knowledge, Version 3.0*. IEEE Computer Society Press, 2014.

[21] I. Jacobson, G. Booch, and J. Rumbaugh, *The unified software development process*. Addison-Wesley, 1998, vol. 1.

[22] H. Takeuchi and I. Nonaka, "The new new product development game," *Harvard business review*, vol. 64, no. 1, pp. 137–146, 1986.

[23] S. Klepper, S. Krusche, S. Peters, B. Bruegge, and L. Alperowitz, "Introducing continuous delivery of mobile apps in a corporate environment: A case study," in *Proceedings of the 2nd International Workshop on Rapid Continuous Software Engineering*. IEEE/ACM, 2015, pp. 5–11.

[24] S. Krusche, "Rugby-a process model for continuous software engineering," Ph.D. dissertation, Technische Universität München, 2016, https://mediatum.ub.tum.de/download/1290336/1290336.pdf.

[25] S. Krusche, M. Berisha, and B. Bruegge, "Teaching Code Review Management using Branch Based Workflows," in *Proceedings of the 38th International Conference on Software Engineering*. IEEE, 2016.

[26] D. Dzvonyar, S. Krusche, R. Alkadhi, and B. Bruegge, "Context-aware user feedback in continuous software evolution," in *Proceedings of the International Workshop on Continuous Software Evolution and Delivery*. ACM, 2016, pp. 12–18.

[27] B. Bruegge, S. Krusche, and L. Alperowitz, "Software engineering project courses with industrial clients," *ACM Transactions on Computing Education*, vol. 15, no. 4, pp. 17:1–17:31, 2015.

[28] S. Krusche, A. Seitz, J. Börstler, and B. Bruegge, "Interactive learning: Increasing student participation through shorter exercise cycles," in *Proceedings of the 19th ACE Conference*. ACM, 2017.

[29] R. Steenweg, M. Kuhrmann, and D. M. Fernández, "Software engineering process metamodels - a literature review," Technische Universität München, Tech. Rep. TUM-I1220, 2012.

[30] T. Ternité and M. Kuhrmann, "Das V-Modell XT 1.3 Metamodell," Technische Universität München, Tech. Rep. TUM-I0905, 2009.

[31] *Software & Systems Process Engineering Meta Model Specification (Version 2.0)*, Object Management Group, 2008.

[32] *International Standard 24744: Software engineering — Metamodel for development methodologies (2nd edition)*, ISO/IEC, 2014.

[33] J. Ralyté, R. Deneckère, and C. Rolland, "Towards a generic model for situational method engineering," in *Proceedings of the 15th International Conference on Advanced Information Systems Engineering*, 2003, pp. 95–110.

[34] G. Engels and S. Sauer, "A meta-method for defining software engineering methods," in *Graph transformations and model-driven engineering*. Springer, 2010, pp. 411–440.

[35] *Business Process Definition Meta Model Specification (Version 1.0)*, Object Management Group, 2008.