A Conceptual Framework for Guiding the Development of Feedback-Controlled Bulk Data Processing Systems

Martin Swientek
Paul Dowland

School of Computing and Mathematics
Plymouth University
Plymouth, UK
e-mail: {martin.swientek, p.dowland}@plymouth.ac.uk

Bernhard Humm
Udo Bleimann

Department of Computer Science
University of Applied Sciences Darmstadt
Darmstadt, Germany
e-mail: {bernhard.humm, udo.bleimann}@h-da.de

***Abstract*—The design, implementation and operation of an adaptive enterprise software system for bulk data processing differs from common approaches to implement enterprise systems. Different tasks and activities, different roles with different skills and different tools are needed to build and operate such a system.**

This paper introduces a conceptual framework that describes the development process of how to build an adaptive software for bulk data processing. It defines the needed roles and their skills, the necessary tasks and their relationship, artifacts that are created and required by different tasks, the tools that are needed to process the tasks and the processes, which describe the order of tasks.

***Index Terms*—adaptive middleware; software development process**

I. Introduction

The processing type is usually a fixed property of an enterprise system that is decided when the architecture of the system is designed, prior to implementing the system. This choice depends on the non-functional requirements of the system. These requirements are not fixed and can change over time.

Traditionally, enterprise systems for bulk data processing are implemented as batch processing systems [1]. Batch processing delivers high throughput but cannot provide near-time processing of data, that is the end-to-end latency of such a system is high.

A lower end-to-end latency can be achieved by using single-event processing, for example by utilizing a message-oriented middleware for the integration of the services that form the enterprise system. While this approach is able to deliver near-time processing, it is hardly capable for bulk data processing due to the additional communication overhead for each processed message. Therefore, message-based processing is usually not considered for building a system for bulk data processing requiring high throughput [2].

Additionally, enterprise systems often need to handle load peaks that occur infrequently. When the system faces moderate load, a low end-to-end latency of the system is preferable. During the peak load, it is more important that the system can handle the load at all. A low end-to-end latency is not as important as an optimized maximum throughput in this situation.

In [2] we have introduced the concept of a middleware that is able to adapt its processing type fluently between batch processing and single-event processing. By adjusting the data granularity at runtime, the system is able to minimize the end-to-end latency for different load scenarios.

The design, implementation and operation of such a system differs from common approaches to implement enterprise systems:

- There are specific activities or tasks needed to implement the feedback-control subsystem.
- There are roles needed with different skills.
- There are different tools needed to aid the design and development of such as system.

Developing software is a complex process, the quality of a software product depends on the people, the organisation and procedures used to create and deliver it [3].

This paper introduces a conceptual framework to guide the design, implementation and operation of an adaptive system for bulk data processing. It defines views, roles, tasks and their dependencies, and processes to describe the necessary steps for design, implementation and operation of an adaptive system for bulk data processing.

Figure 1 shows an overview of the conceptual framework. It is organized among the phases plan, build and run. Each phase contains tasks, which are relevant for each phase:

- **Plan**
  The plan phase contains tasks relevant for the analysis and design of the system, such as the definition of the service interfaces, definition of the integration architecture and definition of performance tests.
- **Build**
  The build phase contains tasks relevant for the implementation of the system, such as the implementation of services, implementation of the integration layer and the implementation of the feedback-control subsystems.
- **Run**
  The run phase contains tasks relevant to the operation

of the developed system, such as monitoring, setup and tuning.

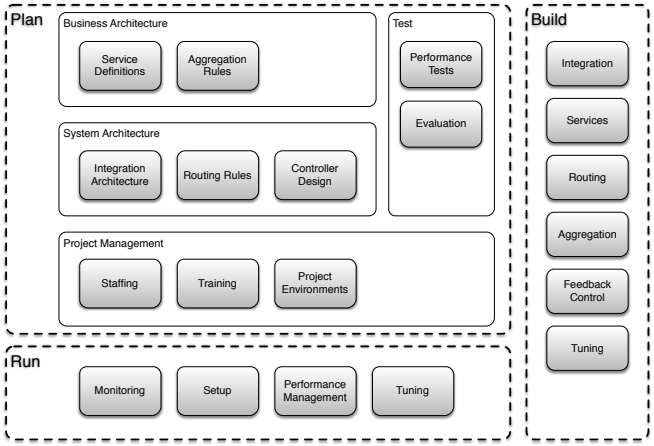


Fig. 1. Overview of Conceptual Framework

The conceptual framework only describes concepts that are specific to the design and implementation of an Adaptive Middleware as described in the previous chapter. It does not describe common concepts for software development.

The remainder of this paper is organized as follows. Section II briefly introduces the concept of an adaptive middleware for bulk data processing. The conceptual framework is presented in Section III. Section IV gives an overview of other work related to this research. Finally, Section V concludes the paper and gives an outlook to the next steps of this research.

II. Background

This section briefly introduces the concept of an adaptive middleware, which is able to adapt its processing type fluently between batch processing and single-event processing.

It continuously monitors the load of the system and controls the message aggregation size. Depending on the current aggregation size, the middleware automatically chooses the appropriate service implementation and transport mechanism to further optimize the processing [2].

Figure 2 shows an overview of the adaptive middleware and its components.

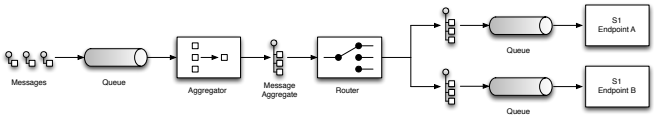


Fig. 2. Overview of the adaptive middleware for bulk data processing [2]

The components of the middleware are based on the Enterprise Integration Patterns described by [4], as shown in Table I.

To control the level of message aggregation at runtime, the middleware uses a closed feedback loop with the following properties (see Figure 3):

TABLE I
Components of the Adaptive Middleware. We are using the notation defined by [4]

Symbol	Component	Description
	Message	A single message representing a business event.
	Message Aggregate	A set of messages aggregated by the Aggregator component.
	Queue	Storage component which stores messages using the FIFO principle.
	Aggregator	Stateful filter which stores correlated messages until a set of messages is complete and sends this set to the next processing stage in the messaging route.
	Router	Routes messages to the appropriate service endpoint.
	Service Endpoint	Represents a business service.

- **Input (u):** Current aggregation size
- **Output (y):** Change of queue size measured between sampling intervals
- **Set point (r):** The change of queue size should be zero.

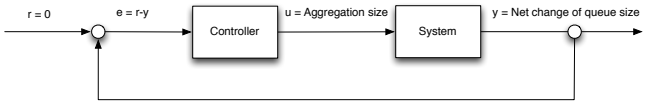


Fig. 3. Feedback loop to control the aggregation size

III. Conceptual Framework

The design, implementation and operation of a system based on the adaptive middleware introduced in Section II differs from common approaches to implement enterprise systems. We have therefore developed a conceptual framework to describe a development process how to build such a system.

A. Metamodel

The conceptual framework consists of the following entities, as shown in Figure 4:

- **Phase**
  Phases correspond to the different phases of a software development lifecycle, such as design, implementation and operations and contain the relevant tasks.
- **Task**
  Tasks represent the activities of the development process. A task
  - is contained in a phase
  - is processed by a role
  - produces and requires artifacts

– uses tools

- **Role**
  Roles represent types of actors with the needed skills to process specific tasks.
- **Artifact**
  An artifact represents the result of a tasks. Additionally, an artifact is a requirement of a tasks.
- **Tool**
  A tool is used by a tasks to produce its artifact.
- **Process**
  A process contains an ordered list of tasks that need to be processed in a certain order.

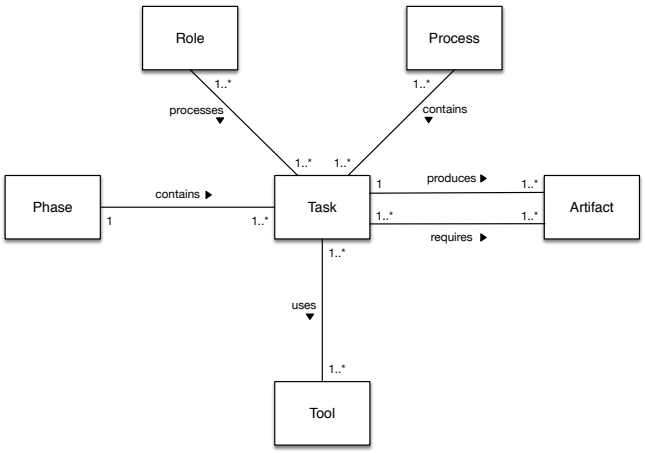


Fig. 4. Metamodel

B. Roles

Roles represent the actors, which process tasks, that is, they describe *who* does something. The description of a role contains its responsibilities and needed skills. A role is not the same as a person, a single person can have multiple roles and change the role according to the context of the current task.

The conceptual framework defines the following roles:

- **Business Architect**
  The business architect is responsible for defining the business architecture of the software system.
- **System Architect**
  The system architect is responsible for defining the technical architecture of the software system.
- **Software Engineer**
  The software engineer is responsible for implementing the software system.
- **Test Engineer**
  The test engineer is responsible for defining and performing the system test.
- **Operations Engineer**
  The operations engineer is responsible for all aspects concerned with running the developed software system.
- **Project Manager**
  The project manager is responsible for managing the software development process.

A role is described by the following attributes:

- **Name**
  The name of the role.
- **Description**
  Description of the responsibilities of the role.
- **Tasks**
  The tasks the role is responsible to process.
- **Needed skills**
  The skills the role has to have in order to successfully process its tasks.

C. Tasks

Tasks are the main entities of the conceptual framework. A Tasks describes *what* should be done, *why* should it be done, and *who* should do it. Additionally, it describes the required and produced artifacts, the tools that should be used to process the task and the expected challenges.

Tasks depend on each other, some tasks must be processed in a certain order. A task can have multiple subtasks.

The Conceptual Framework only describes tasks that are specific to the design and implementation of an Adaptive Middleware for Bulk Data Processing as described in [2]. It does not describe common tasks or activities that are needed for every software system.

Figure 5 shows an overview of the tasks grouped by the different phases of the Conceptual Framework.

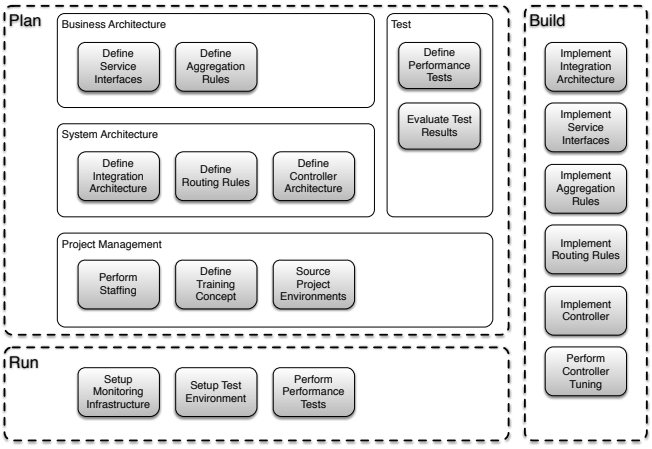


Fig. 5. Overview of tasks

A Task is described by the following attributes:

- **Name**
  The name of the task.
- **What**
  Describes the content of the task.
- **Why**
  Describes the purpose of the task.
- **Who**
  Describes the roles, that are responsible for processing the task.
- **Input**
  The required artificats of the task.

- **Output**
  The artifacts produced by the task.
- **Tools**
  The tools that are needed to process the task.
- **Challenges**
  Describes the expectable challenges when processing the task.

D. Processes

A process contains an ordered list of tasks that are concerned with the implementation of a certain feature of the software system. Processes are modeled using Unified Modeling Language (UML) activity diagrams. The conceptual framework describes the following processes:

- Implement Integration
- Implement Aggregation
- Implement Feedback-Control

1) Implement Integration: This process describes the necessary tasks to implement the integration layer and the integrated service interfaces, as shown in the UML activity diagram in Figure 6.

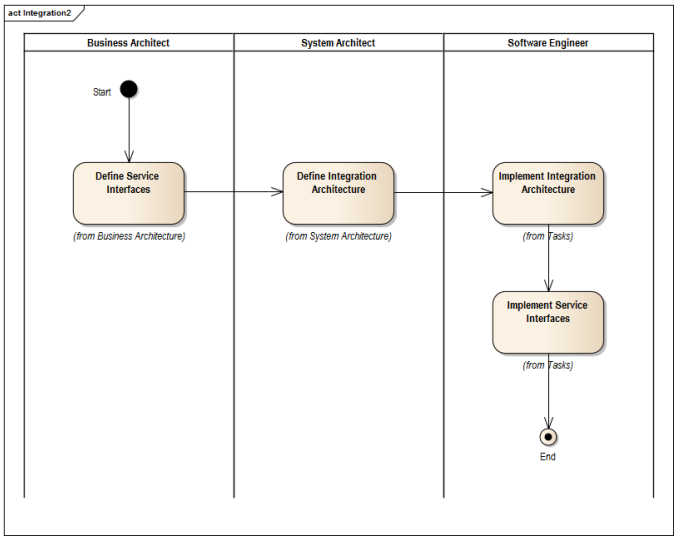


Fig. 6. UML Activity Diagram: Implement Integration

2) Implement Aggregation: This process is concerned with the implementation of the message aggregation, as shown in the UML activity diagram in Figure 7.

3) Implement Feedback-Control: This process contains tasks that are concerned with the design, implementation and tuning of the feedback-control loop, as shown in Figure 8.

There are two options for implementing the feedback-control loop:

- Using a system model for performing the controller tuning, as shown in the UML activity diagram in Figure 9a.
- Without using a model, the control architecture needs to be implemented prior to the controller tuning, as shown in the UML activity diagram in Figure 9b.

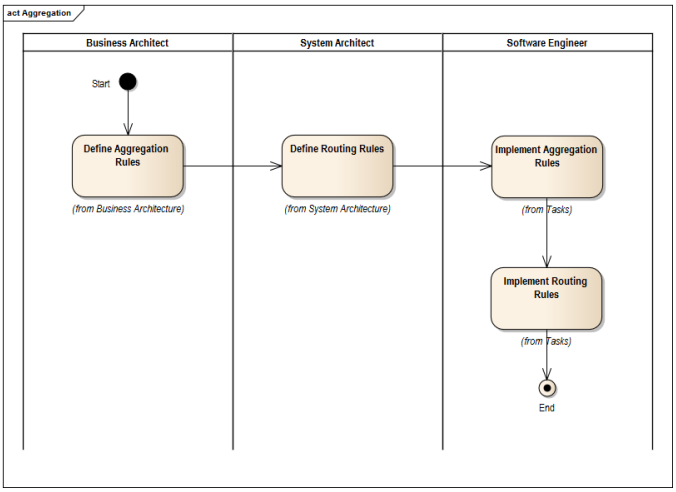


Fig. 7. UML Activity Diagram: Implement Aggregation

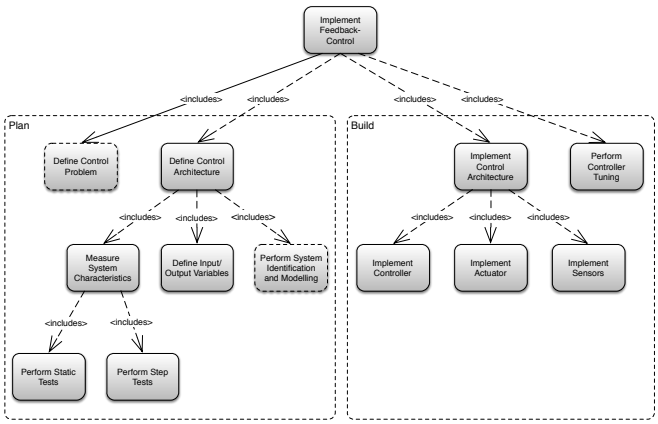


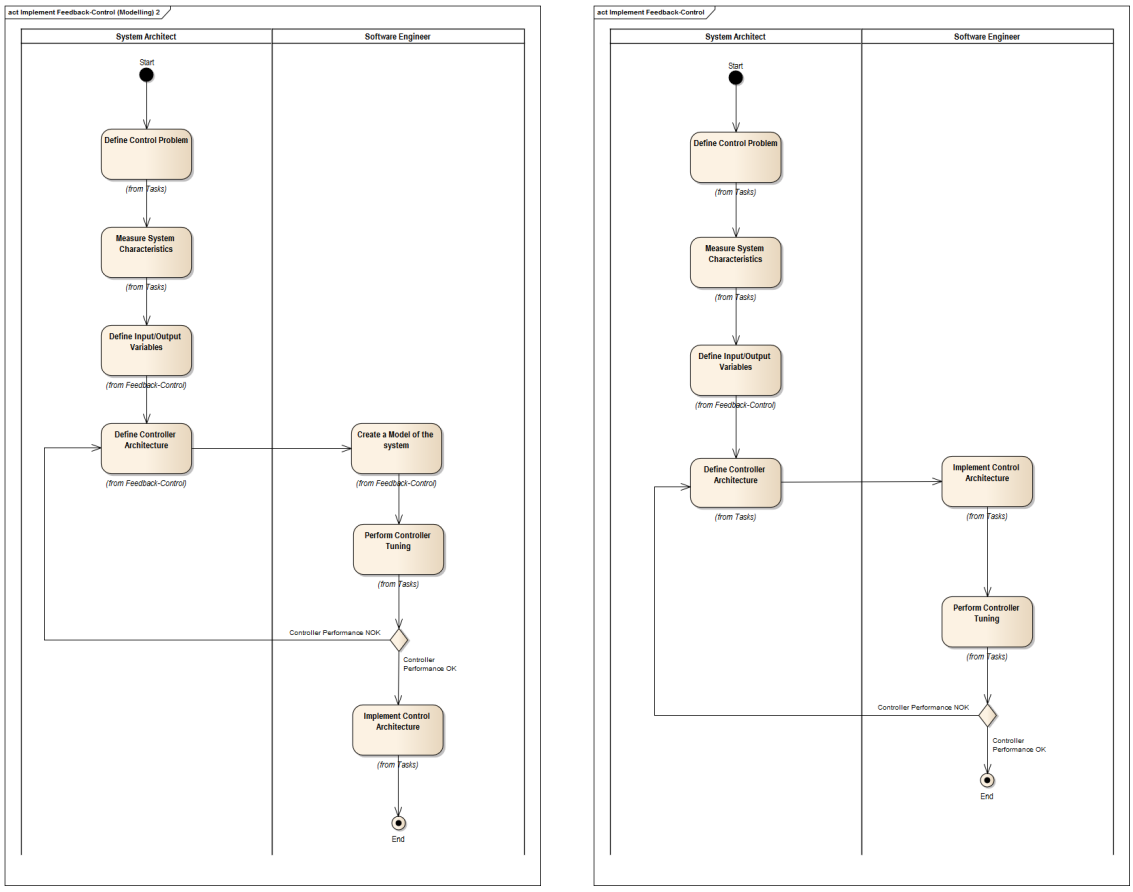
Fig. 8. Tasks for implementing the feedback-control loop

E. Artifacts

An artifact is a result of a task. It is an intermediate result, that is needed for development of the software, but not the software product itself. Additionally, it can also be prerequisite of another task.

The conceptual framework defines the following artifacts:

- **Performance Requirements**
  Defines the requirements regarding the performance of the system, such as required maximum throughput, required maximum latency or desired minimum latency. Defines the workload scenarios of the system.
- **Service Interface Definition**
  Defines the structure of input and output data. Does not include informations about the technical format, such as Extended Markup Language (XML) or JavaScript Object Notation (JSON), and the integration style, such SOAP or Representional State Transfer (REST).
- **Aggregation Rules**
  Defines how events should be correlated with each other by the Aggregator.



(a) Using a model

(b) Without using a model

Fig. 9. UML Activity Diagram: Implement Feedback-Control Loop

- **Integration Architecture**
  Defines the technical integration of the business services, including Middleware technology or product, transports, such as Java Messaging Service (JMS), SOAP or File Transfer Protocol (FTP), Technical format of the input and output data, such as XML or JSON, Comma Separated Values (CSV) or binary formats.
- **Routing Rules**
  Defines which service endpoint should be called by the Router for a given aggregation size.
- **System Model**
  The system model is used to build a simulation of the system which can be used for implementing the controller.
- **Controller Configuration**
  The controller configuration specifies the parameter of the Controller.
- **Training Concept**
  Defines the training concept, including the audience, the content and the type of training. Additionally it contains a timeplan, learning modules and needed facilities to conduct the training.
- **Staffing Plan**
  Defines the required team members and their utilisation over the project time (staffing curve), the required roles and their assignment to team members and a skill matrix that shows the required skills and the knowledge of each team member.

An artifact is described by the following attributes:

- **Name**
  The name of the artifact.
- **Description**
  A description of the artifact.
- **Task**
  The task that produces the artifact.
- **Role**
  The role that is responsible for producing the artifact.

IV. Related Work

This section discusses work related to the conceptual framework presented in this paper. It introduces the terms *Software Process* and *Software Process Modelling* and discusses approaches to model the software process using UML.

A. Software Process

"The software process is a partially ordered set of activities undertaken to manage, develop and maintain software

systems." [5]

McChesney [6] describes the software process as "collection of policies, procedures, and steps undertaken in the transformation of an expressed need for a software product into a software product to meet that need.".

Another similar definition comes from Fugetta [3]. He defines the software process as the "coherent set of policies, organizational structures, technologies, procedures, and artifacts that are needed to conceive, develop, deploy, and maintain a software product."

It is necessary to differentiate between the terms software process and software lifecycle. A software lifecycle describes the states through which the software passes from the start of the development until the operation and finally the retirement. [7] Examples of software lifecycle models are the waterfall model [8] or the spiral model [9].

B. Software Process Modelling

Software process modelling describes the creation of software development models [5]. A software process model is "an abstract representation of a process architecture, process design or process definition, where each of these describe, at various levels of detail, an organization of process elements of either a completed, current or proposed software process" [10].

Process models are described using Process Modelling Languages (PMLs). A PML is defined in terms of a notation, a syntax and semantics, often suitable for computational processing [13].

Typical elements of PMLs are (see for example [11], [5], [3] and [12]):

- Agent or Actor
- Role
- Activity
- Artifact or Product
- Tools

Process Models commonly use different perspectives to describe the software process [12]:

- Functional: what activities are being performed
- Behavioral: In which order (when) are activities performed
- Organizational: where and by whom is an activity performed
- Informational: the entities produced by the process

Examples of software process models include the IEEE and ISO standards IEEE 1974-1991, ISO/IEC 12207 and the Rational Unified Process (RUP).

C. Software Process Modelling using UML

UML is commonly used for modelling software processes.

UML for Software Process Modelling (UML4SPM) is an UML-based metamodel for software process modelling [13], [14]. It takes advantages of the expressiveness of UML 2.0 by extending a subset of its elements suitable for process modelling. UML4SPM contains two packages. The process structure package, which contains the set of primary process elements and the foundation package, which contains the subset of UML 2.0 concepts extended by this process elements to provide concepts and mechanisms for the coordination and execution of activities.

Software & System Process Modelling Metamodel (SPEM) 2.0 is a metamodel for modeling software development processes and a conceptual framework, which provides concepts for for modeling, documenting, presenting, managing, interchanging, and enacting development methods and processes [15]. It provides a clear separation between method content, for example deliverables and key roles, and workflows supporting different software lifecycle models. The SPEM 2.0 metamodel consists of seven main metamodel packages, with each package extending the package it depends on.

Both approaches, UML4SPM and SPEM 2.0 extend the UML 2.0 notation with additional elements, which does not allow the usage of standard UML tools.

[16] use UML 2.0 for modelling software processes at Siemens AG. According to the authors, the usage of standard UML 2.0 notation, which is supported by standard modelling tools, increases readability of processes for software developers since UML is also used for modelling the software itself. They describe four distinct process views, that are described by UML activity diagrams, class diagrams and use-case diagrams: process-oriented, activity-oriented, product-oriented, and role-oriented. The following UML diagram types are used by their approach:

The conceptual framework for feedback-controlled systems for bulk data processing presented in this chapter is based on the properties of the described approaches in this section for modelling the software development process. It uses standard UML use-case and activity diagrams for describing tasks and processes for the following reasons:

- **Understandability**
  Using standard UML 2.0 notation elements and diagrams facilitate the understanding of the conceptual framework since they are commonly used by software engineers for the design of the software system itself.
- **Tool support**
  Standard UML 2.0 notation elements and diagrams are supported by a wide range of modelling tools.

Standard metamodels for software process modelling such as SPEM 2.0 have not been used because they seemed to heavyweight for this purpose.

V. Conclusion

In this paper, we have presented a conceptual framework to guide the design, implementation and operation of an enterprise system that implements the adaptive middleware for bulk data processing as described in [2].

The conceptual framework consists of the entities phases, roles, tasks, artifacts and tools. It describes:

- The needed roles and their skills for the design, implementation and operation.

- The necessary tasks and their relationships for the design, implementation and operation.
- The artifacts that are created and required by the different tasks.
- The tools that are needed to process the different tasks.
- The processes that describe the order of tasks to implement a certain feature of the software system.

It should be noted that software processes are not fixed during their lifetime, they need to be continuously improved. [3] The conceptual model can therefore be tailored to specific projects requirements, it does not have to be followed strictly.

The next step of this research is the evaluation of the conceptual framework by using quantitative research methods, such as expert interviews and its application in real-life projects.

REFERENCES

[1] J. Fleck, "A distributed near real-time billing environment," in *Telecommunications Information Networking Architecture Conference Proceedings, 1999. TINA '99*, 1999, pp. 142–148.

[2] M. Swientek, B. Humm, U. Bleimann, and P. Dowland, "An Adaptive Middleware for Near-Time Processing of Bulk Data," in *ADAPTIVE 2014, The Sixth International Conference on Adaptive and Self-Adaptive Systems and Applications*, Venice, Italy, May 2014, p. 37 to 41.

[3] A. Fuggetta, "Software process: a roadmap." *ICSE - Future of SE Track*, pp. 25–34, 2000.

[4] G. Hohpe and B. Woolf, *Enterprise Integration Patterns: Designing, Building, and Deploying Messaging Solutions*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 2003.

[5] S. T. Acuña and X. Ferré, "Software process modelling." in *ISAS-SCI (1)*, 2001, pp. 237–242.

[6] I. McChesney, "Toward a classification scheme for software process modelling approaches," *Information and Software Technology*, vol. 37, no. 7, pp. 363 – 374, 1995. [Online]. Available: http://www.sciencedirect.com/science/article/pii/0950584995491492I

[7] S. T. Acuña and X. Ferre, "The software process: Modelling, evaluation and improvement," *Handbook of Software Engineering and Knowledge Engineering*, vol. 1, pp. 193–237, 2001.

[8] W. W. Royce, "Managing the Development of Large Software Systems: Concepts and Techniques." *ICSE*, pp. 328–339, 1987.

[9] B. W. Boehm, "A Spiral Model of Software Development and Enhancement." *IEEE Computer ()*, vol. 21, no. 5, pp. 61–72, 1988.

[10] P. Feiler and W. Humphrey, "Software process development and enactment: concepts and definitions," in *Software Process, 1993. Continuous Software Process Improvement, Second International Conference on the*, Feb 1993, pp. 28–40.

[11] K. Benali and J. C. Derniame, "Software processes modeling: What, who, and when," in *Software Process Technology*. Berlin/Heidelberg: Springer Berlin Heidelberg, Jan. 1992, pp. 21–25.

[12] B. Curtis, M. I. Kellner, and J. Over, "Process modeling," *Communications of the ACM*, vol. 35, no. 9, pp. 75–90, Sep. 1992.

[13] R. Bendraou, M.-P. Gervais, and X. Blanc, "UML4SPM: A UML2.0-Based Metamodel for Software Process Modelling," in *Model Driven Engineering Languages and Systems*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2005, pp. 17–38.

[14] ——, "UML4SPM: An Executable Software Process Modeling Language Providing High-Level Abstractions," in *Enterprise Distributed Object Computing Conference, 2006. EDOC '06. 10th IEEE International*, Oct 2006, pp. 297–306.

[15] OMG, "Software Process Engineering Metamodel SPEM 2.0," Object Management Group, Technical Report ptc/08-04-01, 2008.

[16] S. Dietrich, P. Killisperger, T. Stückl, N. Weber, T. Hartmann, and E.-M. Kern, "Using uml 2.0 for modelling software processes at siemens ag," in *Information Systems Development*, R. Pooley, J. Coady, C. Schneider, H. Linger, C. Barry, and M. Lang, Eds. Springer New York, 2013, pp. 561–572. [Online]. Available: http://dx.doi.org/10.1007/978-1-4614-4951-5_45