Comparison of Approaches for developing Self-adaptive Systems

Christian Krupitzer, Martin Pfannemüller, Vincent Voss, and Christian Becker
Information Systems II, University of Mannheim
Mannheim, Germany
{christian.krupitzer,martin.pfannemueller,vincent.voss,christian.becker}@uni-mannheim.de

ABSTRACT

The engineering of software systems enables developers to create very powerful, complex and highly customized software systems by utilizing newest technical capabilities. However, these systems often are error-prone, inflexible, non-reusable and expensive to maintain. Self-adaptation attends to these challenges, offering new ways to automate the adjustment of a system's structure and state. For that reason, many software development approaches specifically consider self-adaptability, leading to a high diversity of methodologies with different characteristics and areas of application. This work addresses this issue by presenting a taxonomy for the analysis and comparison of different approaches for developing self-adaptive systems. In addition, different sample approaches are presented, demonstrating how these dimensions can be applied to compare and classify related work.

CCS CONCEPTS

• **General and reference → Surveys and overviews**; • **Software and its engineering**; • **Computer systems organization** → *Self-organizing autonomic computing*;

KEYWORDS

Survey, Self-adaptive Systems, Software Engineering, Development Approaches

ACM Reference Format:
Christian Krupitzer, Martin Pfannemüller, Vincent Voss, and Christian Becker. 2018. Comparison of Approaches for developing Self-adaptive Systems. In *Proceedings of* . ACM, New York, NY, USA, 14 pages. https://doi.org/10.475/123_4

1 INTRODUCTION

Due to the continuously increasing complexity of contemporary software systems and the high non-functional requirements they have to meet, traditional software engineering approaches do not succeed with supporting developers in the construction of these systems. The reason for that is that such traditional approaches do not emphasize self-adaptability properties and their integration into the system design, architecture, and deployment. Consequently, many new software development methodologies and processes emerge taking into consideration system properties such as flexibility, dependability, customizability and adaptability to spontaneously occurring changes in the system's environment [6, 43]. The concept of self-adaptation allows for the design and implementation of software systems that are able to change and optimize their behavior autonomously by changing their structure or parameters at run-time.

Self-adaptation is the process of re-organizing, re-structuring, and re-configuring a system as a reaction to changes in the resources or environment of the system [29]. In order to be able to execute self-adaptation, the system is equipped with the *self-* properties* [26]. These properties include, among others, self-healing, self-protection, self-optimization, self-configuration [10]. The adaptation logic controls these properties through adaptation of the system resources. Furthermore, self-adaptation can be considered in respect of its type which can be *compositional* or *parametric*. The adaptation process can exchange components (*compositional adaptation*) or change parameters (*parameter adaptation*) [37].

Self-adaptive Systems (SAS) are systems that integrate the concept of self-adaptation. That is, such systems are able to autonomously react to changes or problems at run-time in order to maintain their functionality [29, 42]. Triggers for performing self-adaptive tasks are the system itself, the environment, or the users, such as hardware failures, location changes of mobile systems, or a change in the user preferences.

Traditional software engineering approaches are not specially tailored to the design and construction of complex SAS as they do not incorporate the ability of self-adaptation in their development processes. This results in highly customized, non-reusable, and inefficient software systems that have to be developed entirely from scratch [28]. In order to overcome this problem, many different development methodologies, processes, and frameworks have been created. As powerful and supportive these approaches are, as diverse are their basic concepts, application domains, adaptation mechanisms, and benefits. For the purpose of analyzing, comparing, and better understanding software development approaches for SAS, this paper discusses different aspects and characteristics, joined together in a taxonomy, as well as classifies a selection of different approaches by means of this taxonomy. The goal of this paper is to offer researchers and developers a way to examine, compare and select software development approaches for SAS for further research or development cases.

This paper is structured as follows. In Section 3, we explain the taxonomy for the comparison and its characteristics. Section 4 uses the taxonomy to classify relevant development approaches for SAS. Then, Section 5 presents the discussion and comparison of development approaches. Section 6 presents related surveys. Section 7 concludes this paper with naming open issues and possible future work.

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM.
https://doi.org/10.475/123_4

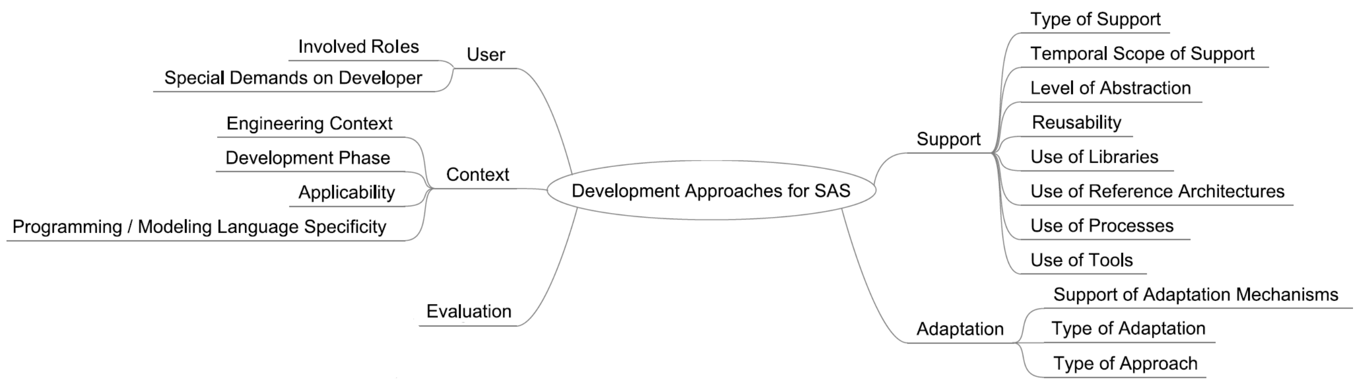

Figure 1: Taxonomy used for the comparison of development approaches for SAS.

2 RESEARCH METHODOLOGY

This paper does not claim to offer an exhaustive survey in the research area of development approaches for SAS. The main purpose is to provide a taxonomy for categorizing such approaches. The application of the taxonomy to classify approaches should highlight the diversity in the area of development approaches for SAS as well as show the degree of support for developers in the research landscape and motivate research gaps. Therefore, this paper focuses on the most important approaches in the research landscape as identified in current surveys (e.g., [29], [33], or [44]) and Dagstuhl seminars in the area of software engineering for SAS in 2008 [6], 2010 [43], and 2013 [16].

As the variety of the relevant approaches is rather large, we decided not to compare all of them together but rather sort the approaches into categories and discuss the differences within the categories as well as cross-categorical. This work looks closely at the categories *Frameworks, Guidelines, Tools, Design concepts,* and *Methodologies.* A framework is an abstraction providing generic functionality that can be extended by user-written code. Guidelines support designers and developers by offering processes for the development. Tools can be used by developers to perform different implementation tasks. Design concepts supports different design activities, such as requirements engineering. Last, software development methodologies are high-level descriptions of procedures. Section 4 presents each category and corresponding approaches.

For the comparison, we worked out a taxonomy with 21 dimensions that are relevant for the development process of SAS. The Research Roadmap by Cheng *et al.* [6], the description of modeling dimensions in [4], the taxonomy from [29] as well as the analysis of the reviewed approaches served as base for the taxonomy's dimensions. The following section presents the taxonomy.

3 TAXONOMY

This section presents the taxonomy used for the analysis, classification, and comparison of development approaches for SAS. Figure 1 provides of the taxonomy. Table 5 in the appendix summarizes the 18 dimensions of the taxonomy as well as their characteristics. In the following, this section presents the dimensions of the taxonomy.

Type of support helps to find a suitable approach for a certain problem. Due to the diversity of development approaches, also the form of support is very diverse. It includes *"framework", "tools", "design principles", "guidelines",* and *"methodologies".*

Temporal scope of support signifies the temporal scope of the different components that provide support. The temporal scope can be *"design-time", "run-time"* or both.

Level of abstraction describes the degree of abstraction of the provided support. Design principles have a high-level abstraction and do not offer concrete implementations, whereas tools and frameworks provide a low-level abstraction, as they directly facilitate the construction of software artifacts.

Reusability refers to the reusability an approach offers and how it is achieved. This includes, e.g., reusable process elements and components, reference architectures, component and design libraries, generic middleware, modeling languages, and design concepts. Some approaches consider reusability at a high abstraction level neglecting lower abstraction levels [28], others do not consider reusability at all.

Use of libraries contains information about the existence and content of libraries, such as component libraries (e.g., based on the MAPE pattern [26]), design pattern libraries (e.g., [41]), and adaptation and coordination mechanisms (cf. [53]).

Use of reference architectures describes whether an approach makes use of a reference architecture and how it is used. Reference architectures serve as architectural templates for the construction of software systems with self-adaptivity properties. However, the structure and functionality can differ.

Use of processes describes the existence and content of processes. Several approaches provide new software development processes tailored to the development of SAS. By contrast, others do neither name nor explain their processes.

Use of tools can support the specification of requirements, the system design, the implementation, or the system validation. Some approaches include proprietary tools that support the software system development, whereas others reference common and open-source software or do not specify tools.

Support of adaptation mechanisms describes how the approach supports adaptation. Dependent on the temporal scope of the support, the approaches consider adaptation at design- or run-time. Furthermore, adaptation can be achieved, e.g., through adaptation and coordination patterns, middleware services, design principles, or the refinement of a model.

Type of adaptation states the granularity of adaptation. It can be *"compositional adaptation"*, *"parameter adaptation"*, or a combination of both.

Type of approach describes the underlying key concept the approach is based on. In accordance with [29], possible manifestations are *"model-based"*, *"architecture-based"*, *"control-based"*, *"service-oriented"*, *"agent-based"*, *"nature-inspired"*, and *"design concepts"*.

Involved roles describes which parties are involved in an approach. Some approaches make a clear and precise statement regarding involved parties, some only distinguish between designers and developers

Special demands on developer covers requirements a developer or designer must possess, such as specific modeling languages, programming languages, or other techniques.

Engineering context describes the engineering context and, thus, the integrability of an approach with the chosen software development process. Several approaches limit their application possibilities to traditional forward engineering, some broaden the applicability to modern engineering contexts, and others do not limit them at all.

Development phase states in which phase of the software engineering process the approach should be applied. This covers phases of traditional and modern engineering processes.

Applicability of an approach can be general or specific. SAS are deployed in many different system domains. Hence, SAS can differ in their structure and functionality, based on specific system domain requirements. To meet these requirements, several approaches support the design and construction of specific system types.

Language specificity states whether an approach is bound to any specific programming or modeling language. Often, frameworks and run-time oriented approaches integrate adaptation logic components and a middleware that are programming language specific. In addition, some approaches require the use of a specific modeling language for designing the SAS.

Evaluation captures the type and extent of evaluation. It is inevitable to use the approaches for developing real-world systems for examining their benefits and challenges. Possible proofs of concept are case studies, prototypes, or expert interview.

4 APPLICATION OF THE TAXONOMY TO CLASSIFY DEVELOPMENT APPROACHES FOR SELF-ADAPTIVE SYSTEMS

This section compares 26 approaches for developing SAS using the taxonomy elaborated in Section 3. This illustrates how the taxonomy can be applied to different approaches and makes it possible to compare them with regard to specific properties. However, this section shall not provide an exhaustive survey in the field. As specified in Section 2, the approaches are grouped into the categories *Frameworks, Guidelines, Tools, Design concepts,* and *Methodologies.* Table 1 provides an overview of the reviewed development approaches.

4.1 Frameworks

In general, a framework is an abstraction providing generic functionality that can be extended by user-written code. Frameworks assist designers and developers with the efficient development of software, as they can concentrate on meeting the software requirements, rather than low-level details of a working system. This enables shorter development times. In the context of SAS, approaches based on frameworks often provide a combination of a reference architecture, tools, middleware, development process workflows, and component libraries.

Rainbow [14, 21] is an architecture-based framework that uses software architectures and a reusable infrastructure to support the development of SAS. It offers an abstract architecture model to monitor and execute the system's run-time properties, evaluates the model, and performs adaptations. Thus, it affects the implementation phase, involved developers, and its level of abstraction is rather low. It considers reusability with the aid of a reusable adaptation infrastructure consisting of system, architecture, and translation layers. Furthermore, it offers a tool suite covering a script editor for the custom developed script language, and the RAINBOW development kit. It is also programming language specific, as it makes use of Java and XML implementations. Case studies were conducted by the authors in the course of their work in order to evaluate the approach.

A Model-Driven Approach for Developing Self-Adaptive Pervasive Systems [13] provides model-based support for adding a resource to or removing resources from a system. The approach offers a framework which specifies the condition for an adaptation, adaptation actions performing the adaptation, and adaptation rules that define which trigger caused which action. The temporal scope of this approach is run-time. Involved people are developers who construct the system based on the framework and it is applied in the implementation phase of a development project. Reusability is considered by using an adaptation architecture and adaptation processes. The approach can be applied to the development of *Pervasive Systems*. Furthermore, this approach's level of abstraction is low, as it offers specific implementations of adaptation mechanisms. Those adaptation mechanisms are supported at run-time through models and the adaptation granularity is compositional. Finally, the framework includes a reference architecture based on communication channels.

Meta-Self [17] is a service-oriented framework supporting the development of SAS in engineering requirements. It covers design-time and run-time and involves designers and developers. They identify system properties, select architectural patterns and adaptation mechanisms, instantiate these patterns, the architecture and policies, and the description of meta-data. The granularity of adaptation is compositional and adaptation is performed at run-time through coordination and adaptation services and the enforcement of policies. The approach has been evaluated by conducting case studies.

Table 1: Overview of different development approaches

Name of approach	Year	Type of approach	Type of support	Applicability
Rainbow [14, 21]	2004	Architecture-based	Framework	SAS
Model-Driven Approach [13]	2008	Model-based	Framework	Pervasive Systems
Meta-Self [17]	2008	Service-oriented	Framework	SAS
SodekoVS [47]	2009	Agent-based	Framework	SAS
MUSIC [24]	2012	Model-based, Service-oriented	Framework	SAS
FESAS Framework [28, 30]	2013	Model-based	Framework	SAS
Architectural Framework for Self-Configuration & Self-Improvement at Runtime [48]	2011	Architecture-based	Framework	SAS
FUSION [19]	2010	Model-based	Framework	SAS
SASSY [38]	2011	Service-oriented	Framework	SAS
Zanshin [46]	2012	Control-based	Framework	SAS
StarMX [5]	2009	Architecture-based	Framework	SAS
MOSES [12]	2012	Service-oriented	Framework	SAS
Software Mobility Framework [35]	2010	Architecture-based	Framework	SAS
GRAF [2]	2012	Model-based	Framework	SAS
Software Engineering Guideline [45]	2010	Agent-based	Guideline	Self-organizing Systems
Development Approach and Automatic Process [1]	2015	Architecture-based	Guideline	SAS
SE Processes for SAS [3]	2013	not defined	Guideline	SAS
Genie [9]	2008	Model-based	Tool	Reflective, component-based Adaptive Systems
FESAS IDE [27]	2016	Model-based	Tool	SAS
Modeling Dimension [4]	2009	Design concept	Design concept	SAS
Design Space [11]	2013	Design-driven	Design concept	SAS
High Quality Specification [31]	2013	Model-based	Methodology	SAS
Behavioral corridors [40]	2010	Verification-based	Methodology	SAS
General Methodology for Designing SOSs [22]	2007	Design concept	Methodology	Self-organizing Systems
FORMS [52]	2010	Model-based	Methodology	SAS
DYNAMICO [51]	2010/2013	Control-based	Methodology	SAS

SodekoVS [47] provides a generic reference architecture and methodical development support for the development phases. As the framework captures the entire development process, designers, developers, as well as testers are involved in it. The engineering context of the approach is traditional forward engineering. The provided reference architecture is responsible for the configuration and integration of self-organizing processes which are considered to be reusable elements. Furthermore, the approach offers a library containing coordination patterns which are applied at design-time. The adaptation granularity is compositional.

MUSIC [24] is a model-driven, service-oriented framework consisting of a development methodology, a tool suite, and a modeling language. Reusability is provided by using generic, reusable middleware components for automatizing context monitoring and system adaptation. Designers and developers have to create an initial list of resource and context dependencies, perform use-case and design modeling, model transformation and deployment, as well as testing and validation. The application is driven by a development process with tasks for every phase. The tool suite contains tools for creating the application adaptation model, generating source code, as well as testing. It is a combination of open source and tailored tools. The MUSIC middleware takes care of supporting adaptation mechanisms at run-time. MUSIC includes the *MAPE-K model* as well as adaptation mechanisms of other approaches.

FESAS Framework [28, 30] is a model-driven framework offering reusable components and design patterns. The framework is equipped with a tool set and includes a middleware that controls system deployment. Designers use the FESAS IDE to create a design model that, then, is transformed into a system model by FESAS. Developers use the FESAS IDE to create code for MAPE components. Because of the inclusion of reusable components, a reference architecture including middleware as well as distribution and design patterns, this approach strongly emphasizes reusability. The reference architecture and middleware are used for the transformation of the design model into a system model. Furthermore, the approach uses a component library containing control loop elements, design/ distribution patterns, and support for meta-adaptation of the adaptation logic at runtime. FESAS makes use of other approaches, as it incorporates the *MAPE-K principle*, the *BASE middleware* [8], and patterns for decentralized control (cf. [53]).

Architectural Framework for Self-Configuration & Self-Improvement at Runtime [48] complements the observer/controller adaptation logic known from Organic Computing [39] with an additional layer for learning new adaptation rules. This framework provides a reusable reference architecture and a simulation-based evaluation of system configurations. The adaptation mechanisms find feasible configuration parameters to adjust the system configuration. If none is found, it relies on learning and simulation. The approach makes use of other tools, e.g., the MASON simulation tool [32] for testing configurations and a Fuzzy Classifier System [49] for rule-based learning. Hence, the support of adaptation mechanisms is at run-time and granularity of adaptation is parameter adaptation. The framework was evaluated in scenarios of traffic and network control.

FUSION [19] is a model-driven framework based on feature models. They represent an abstraction of the system functionality. FUSION provides a reusable architecture that incorporates a learning and adaptation cycle. The adaptation cycle comprises detect, plan, and effect. Adaptation is pursued if the system finds itself not reaching its objectives. Additionally, the learning cycle induces new relationships between features and their impact on the objectives. The used model reflects goals in utility functions. The framework extends the tool XTEAM [18] for modeling goals and features. For learning, the WEKA framework [25] is integrated. The PRISM-MW [36] middleware supports monitoring and dynamic adaptation. The support of the adaptation mechanism is run-time and granularity of adaptation is on system features, i.e., compositional adaptation.

SASSY [38] aims at self-optimization of service-oriented systems at run-time. User priorities are incorporated and traded against each other in a utility function representing the system goals. Goals consist of functional and QoS requirements. Integrating a visual requirements specification for QoS specification, SASSY's self-adaptation approach is capable of generating a system architecture at run-time. SASSY relies on reference architectures for self-adaptation. The approach supports the modeling tool XTEAM and extends xADL [15] for deriving base architectures from its system service architectures at run-time. Therefore, SASSY integrates the Generic Modeling Environment (GME) and the Graph Rewriting and Transformation engine (GReAT). SASSY supports compositional adaptation as it exchanges and initiates varying system architectures at run-time.

Zanshin [46] is a requirements-based framework that adds adaptation to an existing system. The framework monitors failures in the fulfilling of requirements based on log files. The monitor triggers the adaptation process according to the type of failure. Based on goal-oriented requirements engineering, the approach acknowledges functional and non-functional requirements. In [46], Silva Souza extends the Zanshin framework to a control-driven approach that implements compensation for faulty self-adaptation performance. The framework relies on prioritization of requirements during the design phase. Thus, the temporal scope of support is both design-time and run-time. The granularity of adaption is parameter adaption as the adaptation mechanism takes user-defined policies as input. Silva Souza evaluated the approach by conducting a case study on a meeting scheduler system.

StarMX [5] is an architecture-based framework consisting of a development process and a reference architecture which adds self-adaptation to legacy JavaEE-based systems. StarMX's execution engine represents the adaptation logic in the form of processes in an execution chain. Those execution chains represent autonomic managers and get triggered either in intervals or by a defined event sent by the system or other processes. In order to operate, processes depend on anchor objects which provide sensors, effectors and helper functions to interact with the managed system. The approach comprises the development phase and the deployment of the self-adaptation mechanism on an operating system. Designers have to specify self-managing requirements and to provide manageability endpoints. Developers implement a management logic and configure the framework. StarMX supports them with service lookup, proxy generation, activation mechanism, caching, memory scope, data gathering, and logging. The framework incorporates policies and rules as inputs for the adaption cycle. Adapters for the policy engine Imperius and the IBM ABLE rule engine are included but through the exploitation of the adapter design pattern [20], developers may provide their own adapter for any arbitrary policy engine. The authors provide a sample implementation with the web-based application TPC-W in [5].

MOSES [12] focuses on requirement-based QoS aspects within service-based self-adaptive systems. Therefore, Cardellini *et al.* [12] provides an implementation of the adaptation logic following the MAPE cycle. This implementation is reusable in different settings. Developers have to describe the composite service in a workflow orchestration language, such as *Business Process Execution Language*. Additionally, the candidate services have to be described. At run-time, MOSES uses this information to adapt the service-oriented architecture at runtime. As reaction, MOSES might change the composition of the services, hence, performs a structural adaptation. The applicability of MOSES was proven in a JavaEE system.

Software Mobility Framework [35] is an architecture-driven software mobility framework for developing distributed, mobile systems. The framework supports modeling, analysis, implementation, deployment, and run-time migration. Comparable to FUSION [19], it integrates XTEAM for modeling and PRISM-MW for monitoring and architecture-based, structural (re-)configuration. Additionally, DeSi provides algorithms for exploring the configuration space and determining system configurations at design time and runtime. Further, a feedback loop is integrated to deliver information from DeSi to XTEAM which can be used by designers and developers to further improve the system. Hence, it support designers and developers during the whole life cycle. In [35], the framework is evaluated in a robotic scenario.

GRAF [2] integrates TGraphs and accompanying technologies for modeling and manipulating runtime models. GRAF supports automatic updates of runtime models based on observing the managed resources at runtime. Based on the runtime model, GRAF determines adaptation rules for parameter adaptation within the managed resources. The implementation of GRAF relies on Java-based aspect-oriented programming within the managed resources. Amoui *et al.* evaluate the GRAF approach in two case studies: a telephony server and the *Jake2* game engine.

4.2 Guidelines

Guidelines support designers and developers by offering a detailed sequence of working steps that have to be performed in order to achieve a desired result. However, some guidelines also offer the possibility to skip certain steps, extend, or replace them. Typically, they do not offer tools, libraries or reference architectures, but they can be accompanied accordingly.

A Software Engineering Guideline for Self-organizing Resource-Flow Systems [45] is a software engineering guideline combined with a pattern that describes the elements of the system under construction and how they collaborate. This guideline covers detailed design and implementation activities for designers and developers. The guideline can be integrated into a traditional forward engineering approach and is applicable for *Self-organizing Resource-Flow Systems*, such as logistics applications and adaptive production systems. The approach does not offer custom-made tools, but references common development tools that are used in the guideline. The adaptation is based on the construction and execution of the design pattern and is supported at design- and run-time.

Development Approach and Automatic Process for Adaptation at Runtime [1] is a combination of a reference architecture and development guidelines based on automated support. The reference architecture is composed of an adaptation core and four complementary modules, the development, action plan, adaptation rule, and infrastructure module. Involved parties are software engineers as well as domain specialists. The developer involvement covers the identification of system adaptation requirements within the design phase, and the insertion of meta-data in the implementation phase. The adaptation is performed at run-time through the modules of the reference architecture. Included approaches are the *MAPE-K principle* and the *DROOLS framework.*

SE Processes for SAS [3] addresses the issue that traditional software engineering processes cannot cope with the requirements for SAS identified by Andersson *et al.* in [3]. The main difference evolves from the coexistence of design activities performed by designers and developers at design time as well as automatic design activities performed by the SAS at runtime. Consequently, Andersson *et al.* proposed a new process for developing SAS based on the Software and Systems Process Engineering Meta-Model (SPEM) specification. This process targets design time and runtime. The process provides a high-level guideline describing the development process of SAS. However, it does not define any frameworks, tools, or further development support. Contrary, it does not limit the applicability to any specific language, development knowledge, or system domain. As it is a high-level view, it might be easily customized or extended for various settings.

Additionally to the presented guidelines, some of the presented frameworks – e.g., StarMX [5] or FESAS [27, 28] – integrate development processes. However, they are customized to be used in combination with the corresponding framework.

4.3 Tools

As the complexity, functional scope, and requirements of software systems continuously increase, the development of such systems can become inefficient. The provision of design and development tools can make the software engineering process more efficient as they automatize parts of development processes and lower error-proneness. Some approaches offer only one tool for a specific task. This can be the construction and visualization of design concepts, the use of a modeling language, the insertion of meta-data needed for the automated creation of software artifacts, or the validation of such artifacts. By contrast, others offer whole tool suites supporting several development activities.

Genie [9] is a development tool that supports the modeling, generation, and operation of reconfigurable, component-based systems. It allows developers to use three levels of abstraction populated by different artifacts such as models, configurations, policies, and components. As the modeling tool provides design and modeling support, its temporal scope is design-time. However, the artifacts the tool constructs are specific implementations, so that the scope is also run-time. The created artifacts are inserted into a middleware that is able to process the adaptation at run-time. The approach has been evaluated by case studies in the course of the authors' work and includes *MetaEdit+*, an environment for creating and using domain-specific modeling languages.

FESAS IDE [27] complements the Eclipse IDE with two plug-ins. The FESAS Development Tool supports developers in writing of code for MAPE-K component. This plug-in is specific for systems implemented for the FESAS framework. Due to its specificity, this plug-in is excluded for the comparison. The FESAS Design Tool offers a model-based approach for designing SAS. It is based on the

Eclipse Modeling Framework (EMF), the Graphical Modeling Framework (GMF), and the *Acceleo* code generator. It relies on the generic MAPE-K model [26] and offers support for modeling decentralized interaction patterns from [53]. Hence, the information captured by the models is reusable for many different SAS. The graphical editor allows to model the SAS using drag&drop of components. This high abstraction eliminates the need for the system designer to learn the modeling syntax. This model is transformed to a system configuration that is in accordance with the FESAS modeling syntax. However, the modular approach enables to easily define another syntax. In [27], the FESAS IDE is evaluated in five case studies.

Additionally, some of the frameworks integrate different tools for the development, such as XTEAM which is used for modeling in [35], [19], and [38] or the Eclipse plug-ins of the MUSIC framework [24]. However, these tools do not cover the development of SASs independently but are integrated in the development process of the corresponding frameworks.

4.4 Design concepts

Design concepts are focused on design-time. The development of SAS starts with the requirements engineering, analysis, and design. The concept of self-adaptation can be examined very early in order to optimize and emphasize self-adaptation. Design-based approaches include design principles and patterns, but do not offer libraries or concrete implementation proposals. However, they can integrate tool support.

Modeling Dimensions of Self-Adaptive Software Systems [4] provide software engineers with a terminology for specifying self-adaptation. Therefore, the dimensions are categorized, considering goals, change, mechanisms, and effects. This approach offers designers the possibility to explore the system's modeling dimensions. Because they do not offer concrete implementation proposals, but very generic design principles, the level of abstraction is rather high. However, this high abstraction level facilitates high reusability and extensibility. The authors state that this approach can be applied to traditional forward as well as reverse engineering contexts. Furthermore, the approach has been evaluated by different case studies.

Design Space for Self-Adaptive Systems [11] is a design concept which emphasizes systematic design and identifies the design space dimension of SAS. Therefore, it discusses key design decisions, design questions, and answers to these question, organizing them into five different clusters. As this design principles are very generic and abstract, they are also very reusable and suit most SAS. The designers are involved by answering all relevant questions provided by the approach in order to explore the design space of the system development. The level of abstraction is very high because they do not offer concrete implementation blueprints. Furthermore, the approach supports adaptation mechanisms at design-time through the exploration of the system's design space resulting in adaptation requirements, specifications and designs. It also offers high extensibility, as it is a very general design concept that can be adopted, integrated into other approaches and expanded by new dimensions.

4.5 Methodologies

A software development methodology, in general, is a strategy or procedure to deal with a certain problem. It can be limited to a specific task, e.g., specification, validation, or deployment, or cover the whole development process. Besides, it can include processes for sub-tasks or the entire software engineering process, modeling languages, or analysis techniques. However, it is not as extensive as a framework and is not bound to a step-by-step guideline, although it may contain a guideline for special tasks embedded into the methodology.

High-Quality Specification of Self-Adaptive Software Systems [31] is a methodology for the specification of SAS. This methodology includes the UML-based *Adapt Case Modeling Language (ACML)*, which allows for the explicit specification of self-adaptation. Furthermore, it is based on formal semantics which help applying quality assurance techniques to the modeled system. As this approach does not consider implementation aspects in particular, its level of abstraction is high. It supports adaptation mechanisms at design-time by separating self-adaptivity concerns using the *ACML*. It is evaluated by conducting case studies.

Behavioral corridors [40] provides a formal method for the specification of organic computing systems using description of behavioral corridors. The approach uses temporal logic (ITL+) to formally specify the adaptive system and supports the Simple Programming Language (SPL) syntax. Therefore, this approach is specific to the semantics of temporal logic and SPL. Based on the Restore-Invariant approach [23], it supports formal verification of the system. The method distinguishes between two system states: (i) functional state and (ii) erroneous state. In functional state, the system reaches its desired goals. In an erroneous state it does not achieve its goals and, therefore, tries to reconfigure itself to return to a functional state. The approach helps designers to verify a system's formal specification during design phase. The formal method makes use of the KIV theorem prover [7]. The methodology provides a guideline for verification, but not a concrete implementation. Thus, the level of abstraction is rather high. The approach was evaluated by conducting a case study on self-organizing resource flow systems.

General Methodology for Designing Self-Organizing Systems [22] is an iterative and incremental development approach that integrate feedback on its development stages to rework previous steps or influencing the implementation of following steps. As the agent-based methodology comprises the design process, its temporal scope of support is design-time. Involved roles are designers because it focuses on the requirements of a system. The approach does not consider reusability as it does not offer specific implementations, instead it yields a concept to route the exploration of implementations. Hence, its assessment reflects a high level of abstraction. The methodology was evaluated by its application on a case study on self-organizing traffic lights.

FORMS [52] incorporates the concepts of computational reflection [34] and architecture-based adaptation. It supports designers in defining a self-adaptive system's formal specification. FOMRS provides a reusable reference model to specify architectural patterns for SAS with the Z notation to precisely describe the system's elements, their properties, and relationships. It enables formal verification by Community Z Tools. The temporal scope of support is the design-time. The level of abstraction is medium as the reference model contributes concrete guidance on how to specify a system's functional and self-managing elements but does not offer a specific implementation. The authors have evaluated the approach by conducting two case studies.

DYNAMICO [51] is a reference model for context-based self-adaptation. It addresses the shortcoming of previous research that often does not clearly examine visibility of feedback loops and missing control-based modeling. According to [51], the visibility of decoupled feedback loops supports analyzability, assessability, and comparability of the adaptation logic. The DYNAMICO reference architecture adds an additional layer on top of the adaptation logic for monitoring and reasoning on adaptation objectives [51], based on the taxonomy of adaptation metrics from [50]. Further, the three-tier feedback-circuit consisting of an overall control objective loop with two loops for adaptation and context-awareness makes the tasks for self-adaptation and context-awareness more visible for developers. This control-driven and architecture-based methodology provides specific architectural patterns but lacks a concrete reference implementation.

5 DISCUSSION

This section discusses the sample approaches presented in Section 4 in terms of their characteristics that have been elaborated according to the taxonomy proposed in Section 3. Due to the immense amount of available development approaches for SAS and their enormous variety regarding underlying concepts, scope and area of application, the selection of sample development approaches is very constrained and does not reflect the collectivity of available approaches. Thus, the comparison of the sample approaches cannot draw conclusions about development methodologies in general. However, it allows for an overview of similarities and differences of such approaches and reveals strengths, weaknesses, oversupply, and lack of them and their provided support.

Because the various categories of development approaches are fundamentally different, we only compare approaches of the same category with each other. This makes sure that the development approaches can be compared comprehensively, providing meaningful findings. As tools are rather specific, we omit them for this discussion.

5.1 Frameworks

In general, it is noticeable that the type of the approaches varies. We presented architecture-based, model-based, service-oriented, agent-based, component-based frameworks. Most approaches offer support at run-time, whereas the others offer support at design- as well as run-time and, thus, involve not only developers, but also designers in the development process. This approves the extensive nature of frameworks. All frameworks take reusability into consideration, providing generic, reusable adaptation architectures/infrastructures. But they differ regarding the provision of reusable process elements and components on the one hand, and middleware on the other hand. Consequently, reusability as an important success factor for such approaches is taken care of similarly. Further, none of the approaches except *FESAS* focuses on reusing existing code for the algorithms of MAPE components, so reusability on a low level. The authors of *SodekoVS* and *MUSIC* explicitly state that these approaches can be integrated into a traditional forward engineering context, indicating that frameworks are preferably applied in a straight forward development process, whereas the other frameworks do not offer information about this dimension. *Rainbow*, *SodekoVS*, and *FESAS* are applicable to SAS in general, while the others are more specialized. Except for *Rainbow*, there is no information about the demands on the developer. As these approaches are meant to facilitate the development of SAS, it would be helpful to provide such information. Furthermore, the support of adaptation mechanisms differs a lot according to the type of approach. Adaptation is achieved through models, adaptation services, coordination mechanisms, or middleware. All frameworks offer compositional adaptation, confirming their component-based structure. Some frameworks provide comprehensive tool support, e.g., *Rainbow*, *MUSIC*, and *FESAS* . Many frameworks includes other works. All in all, the comparison of the approaches' type of support and their year of publication indicates that present frameworks do not only focus on run-time support, but also include design-time activities into their support.

5.2 Guidelines

Comparing the guidelines presented in Section 4, one can see that *Software Engineering Guideline* [45] is agent-based, whereas *Development Approach and Automated Process* [1] is architecture-based. However, they both provide support at design- and run-time, but differ in the type of support, as the one includes a design pattern and the other provides a reference architecture. The SE processes for SAS [3] describe a rather generic applicable process. Furthermore, all approaches can be applied in a traditional forward engineering context. As different the underlying concepts of the guidelines are, as different is their support of adaptation mechanisms. The first guideline achieves adaptation through the construction and execution of the *Organic Design Pattern*, while the second achieves it through different modules. The SE processes for SAS do not specify any restrictions. Finally, they all do not make use of libraries, but include other works and are evaluated by the conduction of case studies.

5.3 Design Concepts

The *Modeling Dimensions* and *Design Space* approaches are both design concepts that provide reusable design principles for the design of SAS, hence, their temporal scope is design-time and involved parties are designers. *Modeling Dimensions* can be applied in a traditional forward or reverse engineering context, whereas there is no information on this aspect given in the *Design Space* approach. The user involvement is very similar, as the designer has to explore and apply design dimensions or answer design questions. As both approaches are design concepts, they do neither use libraries, processes, nor reference architectures, but are extendable, as they can be integrated into the design phases of other development processes and approaches. Both concepts do not include other works and are evaluated with the aid of case studies. In general, both approaches are very similar regarding their characteristics. Thus, it may be interesting to compare the results of their application to actual software systems in respect of strengths, weaknesses, and contextual overlapping.

5.4 Methodologies

In general, methodologies share a higher level of abstraction than frameworks. While frameworks compose a (sub-) system of reusable components for implementation, methodologies tend to be focused around a procedure dealing with a specific problem. For instance, Gershenson's General Methodology tries to give a standardized view on the development process of SAS, from where to start designing a system [22]. The two approaches providing concrete reference models, FORMS [52] and DYNAMICO [51], both hold a medium level of abstraction, as both provide reference models, but do not provide implementation but a starting point from where to develop a system by deploying the models. Opposite, the other methodologies share a high level of abstraction as they focus more on giving generic guidance on the development process.

6 RELATED WORK

In [29], we present a taxonomy of self-adaptation and a survey on engineering SAS motivating a new perspective on these systems with respect to context adaptation [29]. The classification of different approaches through the "type of approach"-dimension described in Section IV is based on the categorization elaborated in the survey on engineering SAS. Furthermore, ideas for the structure of the taxonomy are gained from the taxonomy of self-adaptation. Finally, the approaches considered in their work facilitated the selection of approaches for the comparison in Section V.

Macías-Escrivá *et al.* review state-of-the-art approaches reported in literature [33]. Therefore, different methods and techniques that are are currently applied in the design of SAS are analyzed. Additionally, the authors survey research challenges and applications for SAS.

Salehie and Tahvildari presented an overview over the landscape of self-adaptive software and related research challenges, including their own taxonomy for self-adaptation [44]. Additionally, they present and cluster different approaches for developing SASs.

7 CONCLUSION

In this paper, a profound taxonomy with detailed dimensions for the analysis and comparison of different development frameworks, guidelines, tools, design concepts and methodologies is proposed. By the use of this taxonomy, different approaches can be classified with respect to their underlying key concepts, provision of support, the user involvement, basic applicability conditions, the use of processes, libraries, and tools, the abstraction level, the handling of adaptation, and the consideration of reusability, extensibility, completeness and validity. Furthermore, the taxonomy is applied to 26 different approaches that are analyzed by means of the dimensions, structured, and presented, allowing for a comparison of their characteristics and contents.

Due to the high diversity, differences regarding key concepts, and various extents of development approaches for SAS, it is very challenging to find one taxonomy that is generic enough to be applicable to every type of approach. Moreover, the taxonomy should be detailed, widespread, and in-depth at the same time. Finding appropriate dimensions that break all information of an approach down into key attributes and pieces of information that make a comparison possible and profound, is the greatest challenge of this work. However, the taxonomy has been successfully used for classification of approaches from different categories. But despite the experience that this taxonomy suits a wider range of approaches, many works do not elaborate the approach's concepts and properties thoroughly enough to examine each dimension extensively. In such cases, additional information is needed to accomplish a successful analysis.

The taxonomy proposed in this paper may be refined by adding, adopting, and specifying dimensions in the future. Therefore, it may be reasonable to perform a more detailed survey for every category to find characteristics that suit best. This could increase the quality and usability of the gathered information. Furthermore, the evaluation of the taxonomy in terms of applying it to more approaches would be helpful to find new and identify problematic characteristics.

ACKNOWLEDGMENTS

The authors would like to thank their former student Tanawat Mark Klaisoongnoen for his contribution.

REFERENCES

[1] Frank José Affonso and Elisa Yumi Nakagawa. 2015. Self-adaptive Software : Development Approach and Automatic Process for Adaptation at Runtime. In *Revista Brasileira de ComputaÃğÃčo Aplicada.* 68–84.

[2] Mehdi Amoui, Mahdi Derakhshanmanesh, JüRgen Ebert, and Ladan Tahvildari. 2012. Achieving Dynamic Adaptation via Management and Interpretation of Runtime Models. *Journal of Systems and Software* 85, 12 (2012), 2720–2737.

[3] Jesper Andersson, Luciano Baresi, Nelly Bencomo, Rogério de Lemos, Alessandra Gorla, Paola Inverardi, and Thomas Vogel. 2013. Software Engineering Processes for Self-Adaptive Systems. In *Software Engineering for Self-Adaptive Systems II.* LNCS, Vol. 7475. Springer, 51–75.

[4] Jesper Andersson, Rogério de Lemos, Sam Malek, and Danny Weyns. 2009. Modeling Dimensions of Self-Adaptive Software Systems. In *Software Engineering for Self-Adaptive Systems.* LNCS, Vol. 5525. Springer, 27–47.

[5] Reza Asadollahi, Mazeiar Salehie, and Ladan Tahvildari. 2009. StarMX: A framework for developing self-managing Java-based systems. In *2009 ICSE Workshop on Software Engineering for Adaptive and Self-Managing Systems.* 58–67.

[6] B. H. C. Cheng, R. de Lemos, H. Giese, P. Inverardi, J. Magee et al. 2009. Software Engineering for Self-Adaptive Systems: A Research Roadmap. In *Software Engineering for Self-Adaptive Systems.* LNCS, Vol. 5525. Springer, 1–26.

[7] Michael Balser, Wolfgang Reif, Gerhard Schellhorn, and Kurt Stenzel. 1999. KIV 3.0 for Provably Correct Systems. In *Proceedings of the International Workshop on Current Trends in Applied Formal Method: Applied Formal Methods.* 330–337.

[8] Christian Becker, Gregor Schiele, Holger Gubbels, and Kurt Rothermel. 2003. BASE – a Micro-broker-based Middleware for Pervasive Computing. In *Proc. PerCom.* 443–451.

[9] Nelly Bencomo, Paul Grace, Carlos Flores, Danny Hughes, and Gordon Blair. 2008. Genie: Supporting the Model Driven Development of Reflective, Component-based Adaptive Systems. In *Proc. ICSE.* ACM, 811–814.

[10] A. Berns and S. Ghosh. 2009. Dissecting Self-* Properties. *International Conference on Self-Adaptive and Self-Organizing Systems* (2009), 10–19.

[11] Yuriy Brun, Ron Desmarais, Kurt Geihs, Marin Litoiu, Antonia Lopes, Mary Shaw, and Michael Smit. 2013. A Design Space for Self-Adaptive Systems. In *Software Engineering for Self-Adaptive Systems II.* LNCS, Vol. 7475. Springer, 33–50.

[12] Valeria Cardellini, Emiliano Casalicchio, Vincenzo Grassi, Stefano Iannucci, Francesco Lo Presti, and Raffaela Mirandola. 2012. MOSES: A Framework for QoS Driven Runtime Adaptation of Service-Oriented Systems. *IEEE Transactions on Software Engineering* 38, 5 (2012), 1138–1159.

[13] Vicente Pelechano Carlos Cetina, Pau Giner, Joan Fons. 2008. A Model-Driven Approach for Developing Self-Adaptive Pervasive Systems. In *Proc. Models@RT.*

[14] Shang-Wen Cheng. 2008. *Rainbow: Cost-Effective Software Architecture-Based Self-Adaptation.* Ph.D. Dissertation. Carnegie Mellon University.

[15] Eric M. Dashofy, André Van der Hoek, and Richard N. Taylor. 2001. A Highly-Extensible, XML-Based Architecture Description Language. In *Proceedings of the Working IEEE/IFIP Conference on Software Architecture.* 103–112.

[16] Rogério de Lemos, David Garlan, Carlo Ghezzi, Holger Giese, Jesper Andersson, Marin Litoiu, Bradley Schmerl, Danny Weyns, Luciano Baresi, Nelly Bencomo, Yuriy Brun, Javier Camara, Radu Calinescu, Myra B. Cohen, Alessandra Gorla, Vincenzo Grassi, Lars Grunske, Paola Inverardi, Jean-Marc Jezequel, Sam Malek, Raffaela Mirandola, Marco Mori, Hausi A. Müller, Romain Rouvoy, Cecilia M. F. Rubira, Eric Rutten, Mary Shaw, Giordano Tamburrelli, Gabriel Tamura, Norha M. Villegas, Thomas Vogel, and Franco Zambonelli. 2018. Software Engineering for Self-adaptive Systems: Research Challenges in the Provision of Assurances. In *Software Engineering for Self-Adaptive Systems III*, Rogério de Lemos, David Garlan, Carlo Ghezzi, and Holger Giese (Eds.). Lecture Notes in Computer Science (LNCS), Vol. 9640. Springer. (to appear).

[17] Giovanna Di Marzo Serugendo, John Fitzgerald, and Alexander Romanovsky. 2010. MetaSelf âĂŞ- An Architecture and a Development Method for Dependable Self-* Systems. In *Proc SAC.* ACM, 457–461.

[18] George Edwards, Sam Malek, and Nenad Medvidovic. 2007. Scenario-driven Dynamic Analysis of Distributed Architectures. In *Proceedings of the 10th International Conference on Fundamental Approaches to Software Engineering.* 125–139.

[19] Ahmed Elkhodary, Naeem Esfahani, and Sam Malek. 2010. FUSION: A Framework for Engineering Self-tuning Self-adaptive Software Systems. In *Proc. FSE.* ACM, 7–16.

[20] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. 1994. *Design Patterns: Elements of Reusable Object-Oriented Software.* Addison-Wesley Professional.

[21] David Garlan, Shang-Wen Cheng, An-Cheng Huang, Bradley R. Schmerl, and Peter Steenkiste. 2004. Rainbow: Architecture-Based Self-Adaptation with Reusable Infrastructure. *IEEE Computer* 37, 10 (2004), 46–54.

[22] Carlos Gershenson. 2007. *Design and Control of Self-organizing Systems (PhD Thesis).* PhD Thesis. Vrije Universiteit Brussel.

[23] M. Güdemann, F. Nafz, F. Ortmeier, H. Seebach, and W. Reif. 2008. A Specification and Construction Paradigm for Organic Computing Systems. In *Proc. SASO.* IEEE, 233–242.

[24] S. Hallsteinsen, K. Geihs, N. Paspallis, F. Eliassen, G. Horn, J. Lorenzo, A. Mamelli, and G.A. Papadopoulos. 2012. A development framework and methodology for self-adapting applications in ubiquitous computing environments. *Journal of Systems and Software* 85, 12 (2012), 2840–2859.

[25] G. Holmes, A. Donkin, and I. H. Witten. 1994. WEKA: a machine learning workbench. In *Proceedings of the 2nd Australian and New Zealand Conference on Intelligent Information Systems (ANZIIS).* IEEE, 357–361.

[26] Jeffrey O. Kephart and David M. Chess. 2003. The Vision of Autonomic Computing. *IEEE Computer* 36, 1 (2003), 41–50.

[27] Christian Krupitzer, Felix Maximilian Roth, Christian Becker, M. Weckesser, Malte Lochau, and Andy Schürr. 2016. FESAS IDE: An Integrated Development Environment for Autonomic Computing. In *Proceedings of the 13th International Conference on Autonomic Computing (ICAC).* 15–24.

[28] Christian Krupitzer, Felix Maximilian Roth, Sebastian VanSyckel, and Christian Becker. 2015. Towards Reusability in Autonomic Computing. (2015), 115–120 pages.

[29] Christian Krupitzer, Felix Maximilian Roth, Sebastian VanSyckel, Gregor Schiele, and Christian Becker. 2015. A survey on engineering approaches for self-adaptive systems. *Pervasive and Mobile Computing Journal* 17, Part B (2015), 184–206.

[30] Christian Krupitzer, Sebastian Vansyckel, and Christian Becker. 2013. FESAS: Towards a Framework for Engineering Self-Adaptive Systems. In *Proceedings of the 7th International Conference on Self-Adaptive and Self-Organizing Systems (SASO).* IEEE, 263–264.

[31] Markus Luckey. 2013. *High-Quality Specification of Self-Adaptive Software Systems.* Ph.D. Dissertation. Paderborn University.

[32] Sean Luke, Claudio Cioffi-Revilla, Liviu Panait, and Keith Sullivan. 2004. MASON: A New Multi-Agent Simulation Toolkit. In *Proceedings of the 2004 Swarmfest Workshop.*

[33] Frank D. Macías-Escrivá, Rodolfo Haber, Raul del Toro, and Vicente Hernandez. 2013. Self-adaptive systems: A survey of current approaches, research challenges and applications. *Expert Systems with Applications* 40 (2013), 7267–7279. Issue 18.

[34] Pattie Maes. 1987. Concepts and Experiments in Computational Reflection. In *Proc. OOPSLA.* ACM, 147–155.

[35] Sam Malek, George Edwards, Yuriy Brun, Hossein Tajalli, Joshua Garcia, Ivo Krka, Nenad Medvidovic, Marija Mikic-Rakic, and Gaurav S. Sukhatme. 2010. An architecture-driven software mobility framework. *Journal of Systems and Software* 83, 6 (2010), 972–989.

[36] Sam Malek, Marija Mikic-Rakic, and Nenad Medvidovic. 2005. A Style-Aware Architectural Middleware for Resource-Constrained, Distributed Systems. *IEEE Trans. Softw. Eng.* 31, 3 (2005), 256–272.

[37] P.K. McKinley, S.M. Sadjadi, E.P. Kasten, and B. H. C. Cheng. 2004. Composing Adaptive Software. *IEEE Computer* 37, 7 (2004), 56–64.

[38] D. Menasce, H. Gomaa, S. Malek, and J. P. Sousa. 2011. SASSY: A Framework for Self-Architecting Service-Oriented Systems. *IEEE Software* 28, 6 (2011), 78–85.

[39] Christian Müller-Schloer, Hartmut Schmeck, and Theo Ungerer (Eds.). 2011. *Organic Computing âĂŤ A Paradigm Shift for Complex Systems.* Springer Basel, Basel, 111–125. https://doi.org/10.1007/978-3-0348-0130-0_7

[40] Florian Nafz, Hella Seebach, Jan-Philipp Steghöfer, Simon Bäumler, and Wolfgang Reif. 2010. A Formal Framework for Compositional Verification of Organic Computing Systems. In *Autonomic and Trusted Computing.* LNCS, Vol. 6407. Springer, 17–31.

[41] O. Babaoglu, G. Canright, A. Deutsch, G. Caro, F. Ducatelle et al. 2006. Design patterns from biology for distributed computing. *ACM Transactions on Autonomous and Adaptive Systems* 1, 1 (2006), 26–66.

[42] P. Oreizy, M. M. Gorlick, R. N. Taylor, D. Heimhigner, G. Johnson, N. Medvidovic, A. Quilici, D. S. Rosenblum, and A. L. Wolf. 1999. An Architecture-Based Approach to Self-Adaptive Software. *IEEE Intelligent Systems* 14, 3 (1999), 54–62.

[43] R. de Lemos, H. Giese, H. A. Müller, M. Shaw, J. Andersson et al. 2013. Software Engineering for Self-Adaptive Systems: A Second Research Roadmap. In *Software Engineering for Self-Adaptive Systems II.* LNCS, Vol. 7475. Springer, 1–32.

[44] Mazeiar Salehie and Ladan Tahvildari. 2009. Self-Adaptive Software: Landscape & Research Challenges. *ACM Transactions on Autonomous and Adaptive Systems* 4, 2 (2009), Art. 14.

[45] Hella Seebach, Florian Nafz, Jan-Philipp Steghofer, and Wolfgang Reif. 2010. A Software Engineering Guideline for Self-Organizing Resource-Flow Systems. In *Proc. SASO.* IEEE, 194–203.

[46] V. A. Silva Souza. 2012. *Requirements-based Software System Adaptation.* PhD Thesis. University of Trento.

[47] Jan Sudeikat, Lars Braubach, Alexander Pokahr, Wolfgang Renz, and Winfried Lamersdorf. 2009. Systematically engineering self-organizing systems: The SodekoVS approach. *Electronic Communications of the EASST* 17 (2009).

[48] Sven Tomforde. 2011. *An Architectural Framework for Self-configuration and Self-improvement at Runtime.* PhD Thesis. Universität Hannover.

[49] Manuel Valenzuela-Rendón. 1991. The Fuzzy Classifier System: A Classifier System for Continuously Varying Variables. In *International Conference on Genetic Algorithms.* 346–353.

[50] Norha M. Villegas, Hausi A. Müller, Gabriel Tamura, Laurence Duchien, and Rubby Casallas. 2011. A Framework for Evaluating Quality-Driven Self-Adaptive Software Systems. In *International Symposium on Software Engineering for Adaptive and Self-managing Systems.* 80–89.

[51] Norha M Villegas, Gabriel Tamura, Hausi A Müller, Laurence Duchien, and Rubby Casallas. 2013. *DYNAMICO: A Reference Model for Governing Control Objectives and Context Relevance in Self-Adaptive Software Systems.* Springer, Berlin, Heidelberg, 265–293.

[52] Danny Weyns, Sam Malek, and Jesper Andersson. 2010. FORMS: A Formal Reference Model for Self-adaptation. In *Proc. ICAC.* ACM, 205–214.

[53] Danny Weyns, Bradley R. Schmerl, Vincenzo Grassi, Sam Malek, Raffaela Mirandola, Christian Prehofer, Jochen Wuttke, Jesper Andersson, Holger Giese, and Karl M. Göschka. 2013. On Patterns for Decentralized Control in Self-Adaptive Systems. In *Software Engineering for Self-Adaptive Systems II.* LNCS, Vol. 7475. Springer, 76–107.

A OVERVIEW ON THE APPROACHES

In the following, the appendix presents the detailed results of the comparison of the approaches presented in Section 4. The tables present for each approach its characteristics for the dimensions of the taxonomy. Due to space limitations, the overview is split into several tables.

Table 2: Taxonomy of Development Approaches (1)

Title of Approach	Year	Type of Approach	Type of Support	Temporal Scope of Support	Involved Parties	Reusability	Development Phase	Engineering Context
Rainbow [14, 21]	2004	Architecture-based	Framework, Tools	Run-time	Developer	Reusable adaptation infrastructure consisting of system, Architecture and translation layers	Implementation	not specified
Model-Driven Approach [13]	2008	Model-based	Framework	Run-time	Developer	Reusable adaptation architecture, Adaptation processes for evolution and involution	Implementation	not specified
Meta-Self [17]	2008	Service-oriented	Framework	Design-time, Run-time	Designer, Developer	Generic infrastructure	Design, Implementation	not specified
SodekoVS [47]	2009	Agent-based	Framework	Design-time, Run-time	Developer, Designer, Tester	reusable elements, Reference Architecture, Library	Entire development process	Forward Eng.
MUSIC [24]	2012	Model-based, Service-oriented	Framework, Tools, Modeling language	Design-time, Run-time	Developer, Designer	Generic, reusable middleware	Entire development process	Forward Eng.
FESAS [28, 30]	2013	Model-based	Framework, Tools, Middleware	Design-time, Run-time	Developer, Designer	Reusable processes and component library	Design, Implementation	Forward Eng.
Arch. Framework for Self-Conf. & Self-Impr. [48]	2011	Architecture-based	Framework	Run-time	Developers	Reference Architecture	Implementation	Forward Eng.
FUSION [19]	2010	Model-based	Framework	Run-time	Developers	Reference Architecture	Implementation	Forward Eng.
SASSY [38]	2011	Service-oriented	Framework, Tools	Design-time, Run-time	Developer, Designer	Reusable processes and components, Reference Architecture	Design, Implementation	Forward Eng.
Zanshin [46]	2012	Control-based	Framework	Design-time, Run-time	Developer, Designer	Reusable processes and components, Reference Architecture	Design, Implementation	Forward Eng.
StarMX [5]	2009	Architecture-based	Framework	Design-time, Run-time	Developer, Designer	Reusable components	Design, Implementation	Forward Eng.
MOSES [12]	2012	Service-oriented	Framework	Design-time, Run-time	Developer	Design methods, Reusable components	Design, Implementation	Forward Eng.
Software Mobility Framework [35]	2010	Architecture-based	Framework	Run-time	Developers	Reference Architecture	Implementation	Forward Eng.
GRAF [2]	2012	Model-based	Framework	Design-time, Run-time	Developer, Designer	Reference Architecture	Design, Implementation	Forward Eng.
Software Engineering Guideline [45]	2010	Agent-based	Guideline, Pattern	Design-time, Run-time	Designer, Developer	Organic Design Pattern	Design, Implementation	Forward Eng.
Development Approach and Automatic Process [1]	2015	Architecture-based	Guidelines, Reference Architecture	Design-time, Run-time	Software Engineer, Domain Specialist	Reference Architecture, Guidelines	development process	Forward Eng.
SE Processes for SAS [3]	2013	Model-based	Process	Design-time	Designer	Reusable process elements	Design	Forward Eng.
Genie [9]	2008	Model-based	Tool	Design-time, Run-time	Designer, Developer	not specified	Design, Implementation	not specified
FESAS IDE [27]	2016	Model-based	Tool	Design-time	Designer, Developer	Generic Tools and processes	Design	Forward Eng.
Modeling Dimensions [4]	2009	Design Concept	Design principles and concepts	Design-time	Designer	Generic, reusable design concepts	Design	Forward Eng., Reverse engineering
Design Space [11]	2013	Design Concept	Design principles and concepts	Design-time	Designer	Generic, reusable design concepts	Design	not specified
High Quality Specification [31]	2013	Model-based	Specification methodology	Design-time	Designer	Modeling Language (ACML)	Design	Forward Eng.
Behavioral corridors [40]	2010	Verification-based	Methodology	Design-time	Designer	Reusable concepts	Design	Forward Eng.
General Methodology for Designing Self-Organizing Systems [22]	2007	Design concept	Methodology	Designers	Design-time	not specified	Design	Forward Eng., Reverse engineering
FORMS [52]	2010	Model-based	Methodology	Designers	Design-time	not specified	Design	Forward Eng.
DYNAMICO [51]	2010/2013	Control-based	Methodology	Designers	Design-time	not specified	Design	Forward Eng.

Table 3: Taxonomy of Development Approaches (2)

Title of Approach	Applicability	Special Demands on Developer	Level of Abstraction	Use of Processes	Use of Reference Architecture	Use of Libraries
Rainbow [14, 21]	Self-Adaptive Systems	Mathematical knowledge	Low	not specified	not specified	none
Model-Driven Approach [13]	Evolution and Involution scenarios of Pervasive Systems	not specified	Low	not specified	Architecture based on communication channels	not specified
Meta-Self [17]	Self-Adaptive and Self-Organizing systems	not specified	not specified	Design- and run-time activities	not specified	none
SodekoVS [47]	Self-Adaptive Systems	not specified	not specified	Self-Organized Coordination Engineering	Configuration and integration of self-organizing processes	Catalog of coordination patterns
MUSIC [24]	Self-Adaptive Systems in mobile and ubiquitous computing scenarios	not specified	Low	Model-driven development methodology with tasks for every development phase	none	none
FESAS [28, 30]	Self-Adaptive Systems	not specified	Low	Workflow for entire development process	Transformation of design model into system model	Component library, design and distribution patterns
Arch. Framework for Self-Conf. & Self-Impr. [48]	Self-Adaptive and Self-Organizing systems	not specified	Low	not specified	Observer/ Controller architectures	None
FUSION [19]	Self-Adaptive Systems	Machine Learning	High	not specified	MAPE-K	None
SASSY [38]	Self-Adaptive Systems	not specified	High	Model-driven development methodology	Service-oriented architecture	none
Zanshin [46]	Self-Adaptive Systems	Requirements Modeling	High	not specified	Automated monitoring	None
StarMX [5]	Self-Adaptive Systems	not specified	High	Workflow for the entire development process	Execution Chain Architecture	none
MOSES [12]	Self-Adaptive Systems	not specified	High	not specified	Reference architecture based on MAPE-K	-
Software Mobility Framework [35]	Self-Adaptive Systems	not specified	High	not specified	MAPE-K	None
GRAF [2]	Self-Adaptive Systems	TGraphs	Medium	For modeling	Transformation of runtime models in rules	None
Software Engineering Guideline [45]	Self-Organizing Resource Flow Systems	none	Medium	Development guideline with different steps	none	none
Development Approach and Automatic Process [1]	Self-Adaptive Systems	not specified	Low	Workflow for the entire development process	Reference architecture consisting of modules	none
SE Processes for SAS [3]	Self-Adaptive Systems	none	High	Workflow for the entire development process	none	none
Genie [9]	not specified	Design, Implementation	Low	not specified	none	not specified
FESAS IDE [27]	Self-Adaptive Systems	-	Medium	Workflow for entire development process	Reference architecture consisting of modules	Component library, design and distribution patterns
Modeling Dimensions [4]	Self-Adaptive Systems	none	High	Application of the design dimensions	none	none
Design Space [11]	Self-Adaptive Systems	none	High	Application of the design space principles	none	none
High Quality Specification [31]	Self-Adaptive Systems	Mastering modeling language (ACML)	High	Application of the ACML within the requirements specification, analysis and design phases	none	none
Behavioral corridors [40]	Self-Adaptive and Self-Organizing systems	ITL+, Simple Programming Language	High	Workflow for entire development process	none	none
General Methodology for Designing Self-Organizing Systems [22]	Self-Organizing Systems	none	High	Workflow for entire development process	none	none
FORMS [52]	Self-Adaptive Systems	none	High	Workflow for entire development process	MAPE-K	none
DYNAMICO [51]	Self-Adaptive Systems	none	High	Workflow for entire development process	MAPE-K	none

Table 4: Taxonomy of Development Approaches (3)

Title of Approach	Use of Tools	Programming Language Specifity	Support of Adaptation Mechanisms	Granularity of Adaptation	Evaluation
Rainbow [14, 21]	Stitch script editor, Rainbow development toolkit	Java, XML	not specified	Compositional	Case studies
Model-Driven Approach [13]	not specified	none	At run-time, Adaptation through models	Compositional	not specified
Meta-Self [17]	none	none	At run-time, Adaptation through application of coordination/adaptation services	Compositional	Case studies
SodekoVS [47]	not specified	not specified	At design-time, Adaptation through coordination mechanisms	Compositional	not specified
MUSIC [24]	Tools for creating the application adaptation model, generating source code, testing and validating	Java, OSGi component framework	At run-time, Adaptation through MUSIC middleware	Compositional, Parameter	Trial development, Testing of a collection of applications
FESAS [28, 30]	Design tool capturing design model, Development tools for creation of adaptation logic	Java	At run-time, Adaptation through refinement of system model	Compositional, Parameter	Case studies
Arch. Framework for Self-Conf. & Self-Impr. [48]	MASON simulation tool	not specified	At run-time, Adaptation through modules	Compositional, Parameter	Case study
FUSION [19]	XTEAM, WEKA, PRISM-MW	Java	At run-time, Adaptation through modules	Compositional	Case studies
SASSY [38]	XTEAM, xADL, GME, GReAT	not specified	At run-time, Adaptation through modules	Compositional	Case studies
Zanshin [46]	not specified	not specified	At run-time, Adaptation through modules	Parameter	Case study
StarMX [5]	IBM ABLE, Imperius	J2EE	At run-time, Adaptation through modules	Parameter	Case study
MOSES [12]	BPEL	J2EE	At design- and run-time, Composition of SOAs	Compositional, Parameter	Case study
Software Mobility Framework [35]	XTEAM, DeSI, PRISM-MW	not specified	At run-time, Adaptation through modules	Compositional, Parameter	Case study
GRAF [2]	TGraph	Java, XML	At design- modeling; at run-time, Adaptation through modules	Parameters	Case studies
Software Engineering Guideline [45]	Common development tools	not specified	At design- and run-time, Adaptation through construction and execution of Organic Design Pattern	not specified	Case study
Development Approach and Automatic Process [1]	not specified	not specified	At run-time, Adaptation through modules	not specified	Case study
SE Processes for SAS [3]	none	none	not specified	not specified	Case study
Genie [9]	Tool for the design and construction of software artifacts	not specified	At design- and run-time, Adaptation through use of domain specific modeling languages and artifacts	Compositional	Case studies
FESAS IDE [27]	Customized Eclipse Plug-ins	Java	At run-time, Adaptation through refinement of system model	Compositional, Parameter	Case studies
Modeling Dimensions [4]	none	none	At design-time, Adaptation through design dimension exploration	Compositional, Parameter	Case studies
Design Space [11]	none	none	At design-time, Adaptation through design space principles	none	Case study
High Quality Specification [31]	none	UML	At design-time, Adaptation through separation of self-adaptivity concerns	none	Evaluation based on case studies, Application studies and projects involving students
Behavioral corridors [40]	none	none	At design-time, definition of adaptive behavior	Parameter	Case study
General Methodology for Designing Self-Organizing Systems [22]	none	none	At design-time, definition of adaptive behavior	not specified	Case study
FORMS [52]	none	none	At design-time, definition of adaptive behavior	not specified	Case study
DYNAMICO [51]	none	none	At design-time, definition of adaptive behavior	not specified	Case study

B OVERVIEW OF THE TAXONOMY

The following table summarizes the 18 dimensions of the taxonomy as well as their characteristics.

Table 5: Taxonomy of Development Approaches: Dimensions and Characteristics

Dimension	Captured Information	Characteristics
Type of support	What kind of support does it provide? What elements does the approach include?	framework, tools, design concept, guidelines, methodology
Temporal scope of support	Which temporal scope does the support by the approach affect?	design-time, run-time, both
Level of abstraction	What is the level of abstraction of the approach? Does it solve certain development issues explicitly?	High, medium, low, not specified
Reusability	Is reusability considered? How is it achieved?	reusable process elements, reusable components, reference architectures, component libraries, design patterns, generic middleware, modeling languages, design concepts
Use of libraries	What do they consist of? How are they used?	provided, not provided, not specified
Use of reference architecture	Does the approach provide a reference architecture? How is it integrated and what is its purpose?	Provided, not provided
Use of processes	Is(are) there any process(es) determined?	Provided, not provided
Use of tools	How do the tools support the development? When are they applied?	proprietary tools, open-source tools, no tools
Support of adaptation mechanisms	How does the approach handle the system‘s adaptation? What mechanisms does it utilize?	At design-time (requirements), at run-time (adaptation logic)
Type of adaptation	What is the granularity of the adaptation?	Compositional adaptation, parameter adaptation, both
Type of approach	What is the key concept? What aspects does it focus on?	model-based, architecture-based, control-based, service-oriented, agent-based, nature-inspired, design concept, verification
Involved roles	Which kind of parties are involved in the development process? What people does the approach aim at?	Designer, developer, tester, not specified
Special demands on developer	What requirements does the developer have to fulfill? What type of and how much knowledge is demanded in order to use the approach?	none, modeling languages, programming languages, not specified
Engineering context	Which software engineering context does it suit?	Forward Eng., reverse engineering, not specified
Development phase	In which step(s) of the software development process can it be applied?	design, implementation, both
Applicability	Which systems can the approach be applied on?	SAS, CPS, adaptive systems
Language specificity	Does the approach require a specific programming or modeling language?	programming language, modeling language, independent
Evaluation	Has the approach already been evaluated? How is it tested?	Case studies, Industry cooperation, prototyping, surveys, no evaluation