Towards Application and Evolution of Model-Based Heuristics for Improving SOA Service Design

Kai Niklas
Talanx Systeme AG
Enterprise Architecture Management
Hannover, Germany
kai.niklas@talanx.com

Joel Greenyer
Leibniz Universität Hannover
Software Engineering Group
Hannover, Germany
greenyer@inf.uni-hannover.de

Kurt Schneider
Leibniz Universität Hannover
Software Engineering Group
Hannover, Germany
schneider@inf.uni-hannover.de

***Abstract*—Good service design is key to acceptance and success for a service-oriented architecture (SOA) in an enterprise. Enterprises try to achieve good service design by using guidelines which combine experts' experience, company policies and best practices. Applying, evolving and maintaining guidelines overburdens service designers and reviewers due to the amount and volume. This results in inefficient, costly and frustrating processes. Without an automated support, guidelines provide only limited value to the design process. We describe how our design environment prototype addresses these problems and introduce automatic guideline checks using heuristics on service models. Our evaluation confirms applicability and advantages of our tool. We present a selection of heuristics which are used in our tool. As the second contribution we describe our plan of how to support evolution and maintenance of guidelines and heuristics.**

***Index Terms*—Service Oriented Architecture (SOA); Modeling; Heuristics; Experience; Static Analysis**

I. Introduction

To achieve the goals and benefits of service oriented architectures (SOA), e.g., reusability and maintainability, several aspects have to be considered. In particular, the service design is important which is addressed by quality attributes, e.g., loose coupling and discoverability [1][2][3]. Additionally to well-known quality aspects, each company has its own policies, such as conventions and restrictions, which have to be considered for service design in practice.

The design phase is critical since the design of services has an impact on goals and benefits for business and IT in a SOA. We focus on the design phase as it is well known that finding and fixing problems after deploying a software in production is 100 times more expensive than in an early phase [4].

A common approach for designing services is to use modeling languages such as UML, SoaML [5], or domain specific (modeling) languages (DSL). The service model is then used to generated documentation, technical artifacts for implementation, e.g., WSDL and code.

To ensure the quality of service design, guidelines are used to check the quality of service models. To date, the definition and check is done manually by experts since most quality aspects are only described informally in written text. In practice we observe that especially novices and inexperienced designers have problems following design guidelines. Supporting novices and checking guidelines manually cost a lot of time. Even experts struggle with the amount and complexity.

Works from Gebhart et al. formalize quality aspects into metrics [6]. Unfortunately, these formalizations focus on some general aspects only and still need manual intervention by experts before checking them automatically. In practice, modelers' knowledge and motivation are not high enough to perform such required manual tasks.

Our study is embedded and evaluated at Talanx Systeme AG, an IT service provider for insurance.

Our two main **research questions** are:

(**RQ1**) Is it possible to relieve designers and experts in the early stage of service design by checking guidelines automatically using heuristics?

(**RQ2**) Is it possible to use quality assured service models to improve our heuristics?

Our main **contributions** in this article are:

First (**C1**) we present a prototype to demonstrate and evaluate the applicability of applying heuristics on service models. In particular we show that our design environment is able to detect design guideline violations which are mostly first discovered in reviews. Hence, the time and effort for manual reviews can be reduced.

Second (**C2**) we formulate the most important requirements towards automated application, evolution and maintenance of guidelines and heuristics. These requirements are derived from a service design process which is also presented in this article.

Third (**C3**) we give concrete examples of guidelines and heuristics for the domain of service design. Additionally, we correlate them with general goals & benefits of SOA.

Fourth (**C4**) we introduce a general process for engineering guidelines and heuristics based on industrial experience.

In the next chapter we present the foundations and challenges of quality assurance in service design. Next we motivate the automatic application of heuristics based on a service design process. We then present our heuristic engineering process and a selection of design guidelines and heuristics. Next, necessary improvements towards the evolution and maintenance are given. Afterward, we conduct an industrial study to demonstrate the applicability checking service models automatically using our prototype. We then compare and classify our approach with related work.

II. Foundations and Challenges of Quality Assurance in Service Design

The process to establish services in a SOA is complex and several frameworks and life-cycle models exists, e.g., Erl [1], SoaML [5], SOMA [7]. In brief, services are firstly identified, then specified and finally implemented. The concrete service interface definition is part of the specification phase and central to our research. We use the term service design and service specification synonymously in this article. We assume that the definition of services is done using a modeling language such as UML, SoaML or a DSL resulting in a service model.

A. Service Design Guidelines

Enterprises try to achieve good service design by using design guidelines. Guidelines are mainly based on theoretical knowledge from books, e.g., Erl [1][3], Josuttis [2], practical experiences and company policies. The guidelines are created and managed by a central SOA team (*SOA architects*).

The major aims of design guidelines are to 1) avoid known design mistakes and pitfalls, 2) achieve SOA goals, e.g., increased intrinsic interoperability, 3) achieve SOA benefits, e.g., increased organizational agility, 4) establish a common knowledge base and principles for modeling and governance.

Unfortunately, design guidelines have some downsides: 1) Complexity and unawareness having a high number of design guidelines. 2) Variety of interpretation having guidelines in free text form. 3) Effort during modeling to comply with the guidelines. 4) Effort during quality assessment to check all guidelines. 5) High barrier for inexperienced designers and architects.

B. Service Design Heuristics

The downsides of design guidelines (as described above) can be mitigated by applying them on service models automatically. Therefore, guidelines have to be engineered into executable code first. Guidelines can get very complex and they are written for humans. Hence, complex methods and algorithms are necessary to codify such guidelines. As this is often not possible we decided to use heuristics. Due to simpler decision procedures they allow to make decisions fast in cases of uncertainty and complex situations.

Definition 1: In the domain of service design a *heuristic* is an analytical procedure to make statements about a service model having limited time, knowledge and information. Hypothesis are used which proved to be valid, but may lead to wrong statements.

Example 1: Guideline: “The functionality of a service has to be documented well." – Heuristic: “IF the service documentation contains less than five words THEN the documentation should be revised."

In the example above, we do not exactly know in which case a service is well documented. But based on our experience, we can formulate a hypothesis respectively a heuristic. The heuristic can be used to check the model and hence our guideline. Of course, a more complex linguistic method would be desirable to be more precise. But the given heuristic is a good trade-off between effort and benefit.

In general, humans often rely on simple heuristics. Heuristics mimic an expert who detects design flaws in a similar way. Additionally, simple heuristics may perform better than complex models [8]. Especially, in the early design phase only a limited amount of information is available.

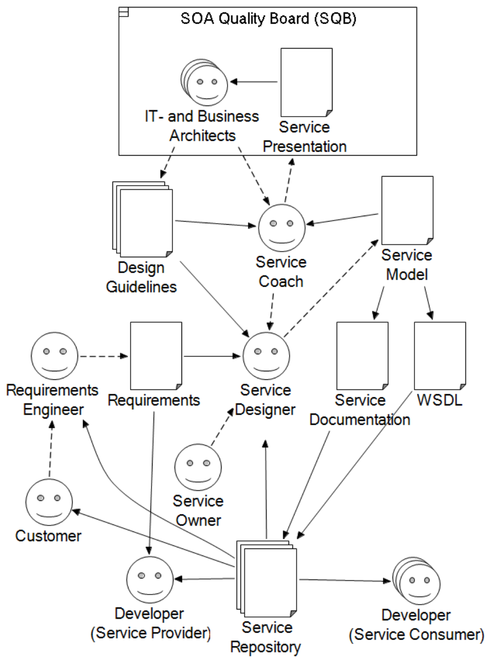


Fig. 1. Service design process with focus on quality assurance.

III. Improving the Service Design Process

Figure 1 illustrates a service design process with focus on quality assurance.[1] The service designer creates service models based on the requirements analyzed by the business analyst. The designer is mentored by a coach (SOA architect). Service designs are checked against design guidelines by the coach and submitted to the SOA quality board (*SQB*). The SQB consists of IT and business architects who perform a review. The SQB has two results: 1) Remarks on the model which are transported to the modeler via the coach and 2) new or modified design guidelines for the knowledge base. Based on the service model necessary artifacts like documentation and technical description, e.g., WSDL, are generated and published to the service repository. Now, service developers can develop or integrate the technical service.

The main problem of the process denoted in Figure 1 is the effort and time exposure to 1) check design guidelines manually, 2) give design advices and 3) to correct the service model in order to comply to the guidelines. Unfortunately, these three steps are often repeated due to humans mistakes, new requirements, or revised models which introduce new design flaws by trying to correct the old ones.

[1]In Fig. 1 we use the FLOW notation [9] to illustrate information and experience flows between people and documents.

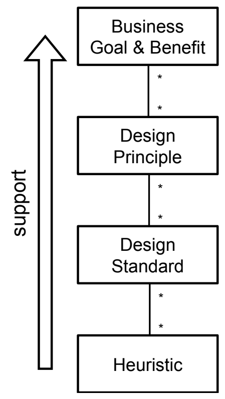


Fig. 3. Correlation of heuristics and business goals & benefits in context of SOA

A. Improvements Checking Design Guidelines Automatically

During modeling our prototype substitutes basic tasks of the coach. The design environment detects design flaws automatically using heuristics similar to those the coach would also apply by hand. That means the coach as central element can be relieved by the tool and is able to focus on more complex and relevant topics which heuristics do not cope. Furthermore, the coach is not any longer a bottleneck in the design process.

Later, in the SQB, known design flaws cannot be missed as they are highlighted in the design environment. Additionally, quality reports can be created automatically based on selected heuristics, e.g., the quality of service documentation. This is useful for an effective and efficient SOA governance.

IV. Service Design Guidelines and Heuristics

Figure 2 gives an overview of the engineering process which we have applied. We divided the process into three steps. 1) discovery of knowledge, 2) engineering of guidelines and 3) engineering of heuristics. Important are the review steps while engineering to increase precision and recall of our heuristics.

For this article we started with 85 service design guidelines from the knowledge base of an IT service organization for insurance. These guidelines are mainly free text descriptions. After an expert review we identified 62 relevant modeling guidelines. We were able to identify and derive 50 computable heuristics. There are six main reasons that we cannot formulate heuristics for all guidelines at the moment.

Language: Specific language aspects are difficult or impossible to check. **Inaccessible information**, e.g., statelessness of application. **Imprecise formulations**, e.g., "use common naming conventions". **Usage of standards**, e.g., ISO. **Runtime behavior**, e.g., the maximum response time of a service. **Complex logic** which have to be specified first.

A. Correlation between Heuristics and SOA Goals & Benefits

By applying design principles successfully, corresponding goals and benefits will be attained [3]. Figure 3 illustrates the correlation between heuristics and business benefits in the context of service design. Design principles are supported by

TABLE I
Examples for heuristics which support design guidelines, principles and business goals & benefits.

Reusability	
Goal & Benefit	Increased Return On Investment (ROI)
Guideline	*Service returning arguments:* Returning arguments should be encapsulated in own (complex) type to stay extensible.
Heuristic	$\forall i$: service.out.attribute[i] = complexType
Loose Coupling	
Goal & Benefit	Reduced IT Burden
Guideline	*Asynchronous Communication:* The exchange of large files (binary) should be handled asynchronously.
Heuristic	$\exists i$: attribute[i].type = binary $\wedge$ service.communication = asynchronous
Standardized Service Contract	
Goal & Benefit	Increased Return On Investment (ROI)
Guideline	*Usage of business objects:* Services should use common business objects (BOs).
Heuristic	$\exists i$: attribute[i].type = BO
Discoverability	
Goal & Benefit	Increased Business & IT alignment
Guideline	*Naming of Service:* Services should be named based on its used business objects.
Heuristic	$\exists i$: attribute[i].type = BO $\wedge$ service.name `contains` attribute[i].name
Statelessness	
Goal & Benefit	Increased Intrinsic Interoperability
Guideline	*Service returning arguments:* Returning arguments should be business objects, not IDs.
Heuristic	!$\exists i$: service.out.attribute[i].name `endsWith` 'id'

design standards which give detailed advices on how to satisfy the principles. Design standards can be patterns, best practices or in our case guidelines. Heuristics help applying and checking design standards which the following examples demonstrate.

B. Examples of Heuristics in the Domain of Service Design

In Table I five concrete heuristics are given. They are categorized into design principles and enriched with business goals & benefits and guidelines. For the notation of heuristics we use predicate logic to stay model independent. If the heuristic is not satisfied (false) the guideline is not satisfied as well. Depending on the used modeling language it is then possible to derive and implement the heuristics, e.g., using OCL for UML models. To date, this is a manual step.

V. Towards Automated Application, Evolution and Maintenance of Guidelines and Heuristics

We have implemented a first prototype of a service design environment to demonstrate and evaluate our approach. It is based on a company-specific DSL created with Xtext running in Eclipse [10]. To date, we have implemented the following main features:

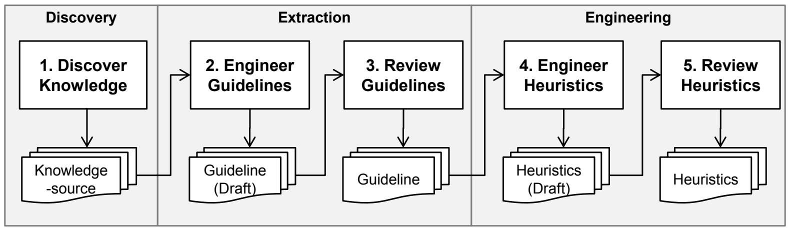
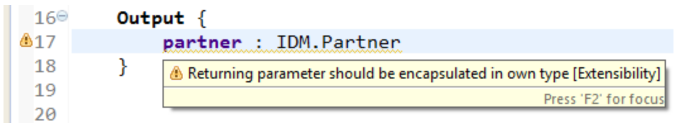


Fig. 2. Process for engineering heuristics in context of service design



```
    Output {
        partner : IDM.Partner
    }
```

Returning parameter should be encapsulated in own type [Extensibility]

Press 'F2' for focus

Fig. 4. Excerpt of an automatically checked service model using a design guideline for extensibility

Immediate heuristic checks. We want to give the user feedback as early as possible. In other design environments checks have to be actively triggered from different perspectives [11] or are executed during compilation [12]. This slows down modeling and annoys users as the feedback loop is often slow.

In-place feedback. Figure 4 depicts such an in-place feedback. Basic information about the detected design flaw is given. Other design environments are only providing lists or worse cryptic log files [12].

Detailed information about the finding are displayed in a separate view next to the model editor. The user can get background information and rational for the finding to understand and learn.

Code templates: Skeletons, e.g., for a new service, can be inserted based on best practices. This speeds up design and avoids common pitfalls.

Alternative views: For more insights alternative views are available, i.e., a UML like class diagram and an outline.

Quickfixes: Detected design flaws can be auto-corrected.

A. *Necessary Improvements*

We plan to investigate and implement the following points to support the evolution and maintenance of guidelines and heuristics.

1) **Capturing of new design guidelines and heuristics**: A lot of knowledge is tacit and has to be externalized before it can be applied automatically. Additionally, model differences after error corrections can be exploited to extract knowledge and correction proposals, e.g., Könemann et al. [13].

2) **Formalization of guidelines and heuristics**: A model independent and formalized description of guidelines and heuristics eases the engineering. It lowers the transformation of guidelines into heuristics and enables the automate transformation of heuristics into executable code.

3) **Automatic correction of design flaws**: Correction proposals which can be applied automatically help enhancing the quality and lowering the effort. The effective and efficient engineering of these proposals has to be explored. Especially, the automatic creation of correction proposals is planned using examples and their model differences.

4) **Capturing of guideline violations and their rationale**: There is always the case, that a guideline has to be violated, e.g., a very specific requirement. A violation should be documented and used for improving heuristics including the rationale instead of only disabling the heuristic [14].

VI. Industrial Study

We have conducted a feasibility study at Talanx Systeme AG using our service design environment. Answers to the following research questions are presented in the following.

RQ1: Is it possible to relieve designers and experts in the early stage of service design by checking guidelines automatically using heuristics?

RQ1.1: Are our implemented heuristics good enough in comparison to experts?

RQ1.2: Which kind of design flaws does an expert detect in comparison to our heuristics?

RQ2: Is it possible to utilize quality assured service models to improve our heuristics?

RQ3: Can the quality of a heuristic be assessed on quality assured models?

A. *Study 1: Applying Heuristics on new Service Models*

We chose a current but random service model which should be quality assured. In our service design environment all heuristics were deactivated and no expert gave design advices. The designer was not aware of the experiment as we have not introduced the automatic checks yet.

The service model was assessed in two ways, independently: 1) Using our design environment and 2) by asking an expert. To answer our research questions we clustered the findings into categories and compared them.

B. *Study 1: Results & Discussion*

The expert identified:
E1: 1 Input / Output parameter type misuse
E2: 5 general documentation hints
E3: 2 attribute naming convention violations
E4: 2 attribute type misuses
E5: 1 uncertain flaw (“Maybe the service is a duplicate.")

The implemented heuristics detected:

H1: 2 Input / Output parameter type naming convention violations
H2: 5 general documentation hints
H3: 2 concrete documentation hints (i.e. string length)
H4: 1 attribute type misuse
H5: 2 false positives concerning naming conventions

Our heuristics detected 10 flaws, whereas the expert found 11 flaws. The answers to RQ1.1 and RQ1.2 can be summarized as follows: 1) The application of our heuristics is comparable to manual reviews by experts. 2) Experts' findings are more specific and respect a hierarchy of criticality. 3) Experts identify additional flaws based on their experience.

Not all flaws can be detected using our design environment as not all experts' experience is captured. But several violations against known guidelines can be eliminated before having in-depth discussions with experts which saves time and effort. Hence, RQ1 can be answered positively. The following comparison and interpretation gives more details:

H1 and E1: Both findings correspond to the same flaw but with different arguments. The heuristic claims that naming conventions for input and output parameter types are violated. The expert claims that input and output parameter reference to the same type. The expert's remark is more specific to the underlying flaw than the heuristic's remark.

H2 and E2: Both findings are equal.

H3 remarks that the length of strings should be documented. The finding is correct but in the context of E4 and H4 obsolete as both claim that the used type "string" is already wrong. So, here is a hierarchy of criticality which should be considered in future research.

H4 and E4: The heuristic found one type misuse as the expert found two. This indicates that the heuristic lacks information for detecting all kind of type misuses.

H5: The knowledge behind this heuristic is wrong which leads to wrong detections. The heuristic has to be refined.

E5: The uncertainty was proven to be correct. A different service could be used. That means, this finding overrules all other. Here we have a hierarchy of criticality again. It would be enough to focus on this finding first.

C. Study 2: Improving Heuristics

In study 1 we observed that our heuristics can be improved using additional expert's knowledge to detect more flaws (H4) and to be more accurate (H5). In order to answer RQ2 we examine three existing quality assured data models and three heuristics. Data models are complex data types which are used in our services. We chose the models based on their complexity (low, medium, high) and three different heuristics.

The three models are:
M1: 21 attributes grouped in 4 subtypes
M2: 50 attributes grouped in 10 subtypes
M3: 314 attributes grouped in 25 subtypes

In order to examine if we are able to improve our heuristics we conduct the experiment in two scenarios:
S1: Using heuristics *without* additional knowledge: All heuristics are implemented as described in the knowledge base.

TABLE II
STUDY 2: RESULTS: HEURISTICS BEFORE (S1) AND AFTER (S2) ADDING EXPERT'S EXPERIENCE.

		M1	M2	M3
		(TP, FP, FN)	(TP, FP, FN)	(TP, FP, FN)
R1	S1	(0, 0, 0)	(0, 0, 0)	(0, 0, 2)
	S2	(0, 0, 0)	(0, 1, 0)	(2, 7, 0)
R2	S1	(20, 0, 4)	(14, 0, 11)	(75, 0, 2)
	S2	(24, 0, 0)	(25, 0, 0)	(77, 0, 0)
R3	S1	(0, 1, 0)	(0, 2, 0)	(0, 27, 0)
	S2	(0, 0, 0)	(0, 0, 0)	(0, 1, 0)

S2: Using heuristics *with* additional knowledge: Like S1 but with additional expert's knowledge.

Additional knowledge means, that an expert made a review on the guidelines and heuristic and specified them more precisely.

The three guidelines and their improved equivalents are:
R1: No attribute should contain unnecessary filler words.
S1: filler words = (data) / S2: filler words = (data, content, file, object, key, value, list, table)
R2: All model elements should be documented.
S1: check if attributes contain documentation / S2: check if attributes and complex types contain documentation
R3: The attribute name of an aggregation should be the same as the aggregated element.
S1: check all aggregations / S2: check only 1:1 aggregations

For evaluating our heuristics we applied them on our models. The findings were categorized by an expert into true positives (TP, correctly detected flaw), false positives (FP, wrongly detected flaw) and false negatives (FN, not found flaw). The goal is to have many true positives and few false positives. We did not count true negatives as they give us no more clues.

D. Study 2: Results & Discussion

RQ2 can be answered positively. Table II illustrates how our heuristics evolved. By adding knowledge to our heuristics we were able to raise the number of TP and to reduce the number of FP. For example, we were able to detect all flaws on all models for R2; and to reduce the number of FP for (M3, R3) from 27 to 1. For R1 we can see that the optimization of our heuristics is not optimal as not only the number of TP is raised but also the number of FP. A better method for improving heuristics is required.

Using heuristics R1 and R3 no TPs were detected in most cases. Hence, we cannot compute typical numbers for the quality of heuristics in general, i.e., precision and recall. That means, evaluations on quality assured models are not necessarily meaningful for evaluating the quality of heuristics. For the presented scenario we have to answer RQ3 negatively.

We identified two possible reasons for this behavior. 1) The vulnerability to design flaws depends on the modeled aspects. Not every flaw detectable by a heuristic has to be present in the current model. 2) High modeling quality. The manual quality checks have eliminated many design flaws beforehand.

E. Threats to validity

Both experiments consider only a small range of models and services. This reduces the significance of our statements but the indication is still given. To remedy this fact we chose a random model for study 1 and differently sized models for study 2.

The used heuristics are company specific and are not representative for all SOA projects. But we belief that this is not relevant for applying heuristics and detecting design flaws. Of course, experts' opinions and heuristics have to match in order to make valid statements about models.

VII. Related Work

The basic design principles for service design, e.g., by Erl [1][3] or Josuttis [2] can be assessed by hand. They are documented but only available in free text form. This can be compared to our knowledge base in which design guidelines are documented.

Design guidelines intent to detect potential design flaws. For the domain of static code analysis, tools exists, e.g., FindBugs [15] or PMD [11]. For the domain of UML models tools exists, too, e.g., ArgoUML [16] or SDMetrics [17]. To our knowledge no such tool tackles (SOA) service design or is able to do it in the way we proposed it.

A more formal method to assess the quality of service design was proposed by Gebhart et al. by defining and using quality metrics [6]. The metrics are a subset of the known SOA design principles and can be applied (semi-) automatically, i.e., on SoaML models. Gebhart also states, that metrics can be used to detect design flaws but some need manual intervention. Our concept uses heuristics which we belief are easier to engineer and give immediate feedback while modeling. Furthermore, we propose to use only automatic applicable heuristics without any manual intervention.

Kannan et al. propose to analyze the whole process to validate SOA design principles [18]. In comparison to our approach different quality aspects of a SOA are tackled. Furthermore, the approach has higher requirements as the whole process has to be modeled at first.

Moha et al. proposes a concept to define and detect SOA antipatterns [19]. Antipatterns are described within a domain specific language, automatically compiled into detection algorithms and then applied to a service based system (SBS). They focus on evaluating a whole system, not concrete service interfaces as we do.

Knauss et al. suggest to apply heuristics on requirements in order to detect errors [20]. Heuristics are based on experiences. Their concept is similar to ours, but based on a different domain and engineering phase.

VIII. Conclusion & Future Research

We illustrated how heuristics based on service design guidelines can be automatically applied to detect design flaws during design time. Therefore, we presented a design environment prototype, which gives immediate feedback to the designer. We motivated the benefits of our design environment with the help of a service design process. Our industrial study confirms that violations against known guidelines can be eliminated before having in-depth discussions with experts. This saves time and human resources doing manual reviews.

For our study we have engineered heuristics based on existing design guidelines. We presented a general applicable engineering process for designing guidelines and heuristics based on existing knowledge. A set of industrial guidelines and heuristics was presented for the domain of service design. The correlation between business goals & benefits and heuristics was illustrated with the aid of this set. We demonstrated that heuristics can be improved by adding knowledge and evaluating them on existing design models.

For future research we outlined necessary improvements of our prototype to support evolution and maintenance of guidelines an heuristics. Especially, the capturing of new and modified guidelines and heuristics using model differences will be the focus of our future research.

References

[1] T. Erl, *Service-oriented architecture*. Prentice Hall Englewood Cliffs, 2005.

[2] N. Josuttis, *SOA in practice: the art of distributed system design*. OReilly: Sebastopol, CA, 2007.

[3] T. Erl, *Soa: principles of service design*. Prentice Hall Upper Saddle River, 2008, vol. 1.

[4] B. Boehm and V. R. Basili, "Software defect reduction top 10 list," *Foundations of empirical software engineering: the legacy of Victor R. Basili*, vol. 426, 2005.

[5] OMG, *Service oriented architecture modeling language (SoaML)*, 2012. [Online]. Available: http://www.omg.org/spec/SoaML/1.0.1/

[6] M. Gebhart and S. Abeck, "Metrics for evaluating service designs based on soaml," *International Journal on Advances in Software*, vol. 4, no. 1, pp. 61–75, 2011.

[7] A. Arsanjani, S. Ghosh, A. Allam, T. Abdollah, S. Ganapathy, and K. Holley, "SOMA: A method for developing service-oriented solutions," *IBM Systems Journal*, vol. 47, no. 3, pp. 377–396, 2008.

[8] G. Gigerenzer and W. Gaissmaier, "Heuristic decision making." *Annual review of psychology*, vol. 62, pp. 451–82, Jan. 2011.

[9] K. Stapel and K. Schneider, "Managing knowledge on communication and information flow in global software projects," *Expert Systems*, 2012.

[10] *Xtext*, 2015. [Online]. Available: https://www.eclipse.org/Xtext/

[11] *PMD*, 2015. [Online]. Available: http://pmd.sourceforge.net/

[12] A. W. Brown, M. Delbaere, P. Eeles, S. Johnston, and R. Weaver, "Realizing service-oriented solutions with the ibm rational software development platform," *IBM systems journal*, vol. 44, no. 4, pp. 727–752, 2005.

[13] P. Könemann, E. Kindler, and L. Unland, "Difference-based model synchronization in an industrial mdd process," in *2nd ECMDA Workshop on Model-Driven Tool & Process Integration (MDTPI 2009)*, 2009.

[14] J. E. Burge and D. C. Brown, "An integrated approach for software design checking using design rationale," in *Design Computing and Cognition '04*, J. Gero, Ed. Springer Netherlands, 2004, pp. 557–575.

[15] D. Hovemeyer and W. Pugh, "Finding bugs is easy," *ACM SIGPLAN Notices*, vol. 39, no. 12, p. 92, Dec. 2004.

[16] *ArgoUML*, 2015. [Online]. Available: http://argouml.tigris.org/

[17] *SDMetrics*, 2015. [Online]. Available: http://www.sdmetrics.com/

[18] K. Kannan, A. Bhamidipaty, and N. C. Narendra, "Design Time Validation of Service Orientation Principles Using Design Diagrams," *Annual SRII Global Conference*, pp. 795–804, Mar. 2011.

[19] N. Moha, F. Palma, M. Nayrolles, B. Conseil, Y.-G. Guéhéneuc, B. Baudry, and J.-M. Jézéquel, "Specification and detection of soa antipatterns," in *Service-Oriented Computing*. Springer Berlin Heidelberg, 2012, vol. 7636, pp. 1–16.

[20] E. Knauss, D. Lübke, and S. Meyer, "Feedback-driven requirements engineering: the heuristic requirements assistant," in *International Conference on Software Engineering (ICSE'09)*. IEEE, 2009, pp. 587–590.