Towards Contractual Interfaces for Reusable Functional Quality Attribute Operationalisations

Jose-Miguel Horcas,
Mónica Pinto, and Lidia Fuentes
Universidad de Málaga, Andalucía Tech, Málaga, Spain
{horcas,pinto,lff}@lcc.uma.es

Steffen Zschaler
Department of Informatics, King's College London,
United Kingdom
steffen.zschaler@kcl.ac.uk

Abstract

The quality of a software system can be measured by the extent to which it possesses a desired combination of quality attributes (QAs). While some QAs are achieved implicitly through the interaction of various functional components of the system, others (*e.g.,* security) can be encapsulated in dedicated software components. These QAs are known as functional quality attributes (FQAs). As applications may require different FQAs, and each FQA can be composed of many concerns (*e.g.,* access control and authentication), integrating FQAs is very complex and requires dedicated expertise. Software architects are required to manually define FQA components, identify appropriate points in their architecture where to weave them, and verify that the composition of these FQA components with the other components is correct. This is a complex and error prone process.

In our previous work we defined reusable FQAs by encapsulating them as aspectual architecture models that can be woven into a base architecture. So far, the joinpoints for weaving had to be identified manually. This made it difficult for software architects to verify that they have woven all the necessary FQAs into all the right places. In this paper, we address this problem by introducing a notion of contract for FQAs so that the correct application of an FQA (or one of its concerns) can be checked or, alternatively, appropriate binding points can be identified and proposed to the software architect automatically.

Keywords Aspect-Orientation, Model-Driven Development, Quality Attributes, Weaving Patterns

1. Introduction

The critical Quality Attributes (QAs) of a software system must be well understood and articulated early during the development of a system, so that the architect can design a software architecture that satisfies them (Bachmann et al. 2005). Some QAs such as security, persistence, or usability, require the addition of specific components into the software architecture of the base application (e.g., a component implementing an encryption algorithm to satisfy the security QA). We refer to these QAs as Functional Quality Attributes (FQAs) (Horcas et al. 2014; Juristo et al. 2007). The association of a function (e.g., the encryption of a message) to a goal (e.g., providing security) is known as the *operationalisation* of the QA, in the sense that the function specifies how a goal can be made operational (Chung et al. 1999). For FQAs the operationalisation is encapsulated in a distinct set of additional components.

Each FQA (*e.g.,* security) can be composed by many concerns (*e.g.,* authentication, encryption, or integrity), and different applications may require a customised subset of each FQA (*e.g.,* only encryption). Also, different FQAs may have dependencies between them (*e.g.,* usability and security), which should be taken into account during architecture elicitation. Moreover, FQAs normally crosscut the system architecture since their operationalization may entail the injection of several functional components in different places. For instance, the encryption concern requires a place where to introduce the encrypt functionality to encrypt the data, and another place where to decrypt the same data.

To better modularize FQAs, in (Horcas et al. 2014, 2016) we model them separately from the base architecture following an aspect-oriented (AO) approach. We defined a family of FQAs, following a Software Product Line (SPL) approach, with the purpose of automatically generating a customised configuration of the FQAs required by a concrete application. For example, the security FQA is defined in terms of access control, authentication, privacy, etc., being each of them optional features (i.e., we define a variability model with FQAs variants), but a concrete application may require only authentication (i.e., the customised configuration of FQAs for this application is composed by authentication). But, in order to weave a customised configuration of FQAs the software architect (SA) must manually identify the points in the application where the FQAs components must be incorporated. This is a complex and error prone task since the SA has no mechanism to verify that the composition of a set of FQA components in the places (i.e., the *join points*) selected by them is correct. This means that after injecting a FQA with the approach of (Horcas et al. 2014, 2016) the SA has no guarantee that the resulting architecture complies the desired quality attribute. Similar existing approaches (Alam et al. 2013; Kienzle et al. 2009) have also strong support for modelling and customize FQAs, but weaving FQA models into software architectures is still quite challenging also for them. That is, the join points are manually selected and no checking is performed of the weaving validity.

Although there are clear benefits of FQA modularisation and reuse (Alam et al. 2013; Horcas et al. 2016), manually selecting the join points where to weave them can lead to corrupted architectures. For example, consider the case of the encryption that operationalized the confidentiality property. This can be easily modelled by a component with an `encrypt` and a `decrypt` methods, which can be interpreted as *advice* in AO terms. Imagine that to incorporate

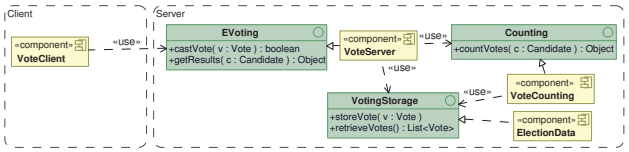

Figure 1. E-Voting application architecture.

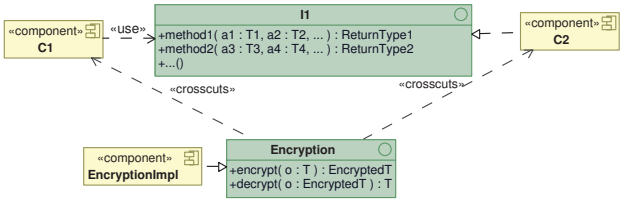

Figure 2. Encryption FQA model.

these functions to the system architecture we define a weaving transformation that injects them at the start and at the end of a concrete component interaction. However, note that confidentiality property is only achieved when this operationalisation is applied to all interactions between components deployed on different hosts (or even between all components regardless of deployment, depending on the attacker model). In this case, if confidentiality components are not woven in all and correct places, the process cannot fairly guarantee that the application architecture possesses the desired quality; and the benefits of reusing an automatically built FQA configuration could be lost. Furthermore, it may not be very efficient to apply encryption/decryption to component connections that do not connect components deployed on different hosts, so injecting confidentiality components to all interactions may lead to some inefficiency problems. Therefore, there is a need to support software architects in making this selection decision.

In this paper, we aim to provide a solution that helps the software architects choose the join points where to weave FQAs and to verify the semantic correctness of the woven application architecture, in order to assure the overall quality of the system. We extend the current Aspect-Oriented SPL approach (Horcas et al. 2016) with contracts that support the software architect to (1) automatically identify the architecture join points where a particular FQA can be woven, and (2) check whether a given FQA has been applied semantically correctly to a software architecture model. To do that, we enhance the current FQA models with additional models for structural pre-conditions and invariants.

After this introduction, in Section 2 we motivate our work with a real case study and an example of an encryption FQA. Section 3 explains our approach. In Section 4 we discuss the related works and compare them with our approach. Finally, Section 5 concludes the paper and sets our future work.

2. Motivation and Case Study

Our case study is an electronic voting (e-voting) application which is one of the environments where quality attribute requirements are complex. Figure 1 shows a simplified software architecture in UML with the main functionality of the e-voting application. This architecture does not include any component related to the security requirements. The `VoteClient` component allows clients to cast their votes to the server as well as to consult the election results, by using the `EVoting` interface. The `VoteServer` component receives the votes and the `ElectionData` stores them in a digital ballot box through the `VotingStorage` interface. The `VoteCounting` component through the `Counting` interface provides access to the election results.

Apart from the base functionality shown in Figure 1, the e-voting application requires a list of security FQAs. Concretely, it is of paramount importance to guarantee the confidentiality of the votes as well as the integrity of them, even in the counting process. Moreover, the voters must be authenticated within the application in order to cast their votes. This means that some security components must be incorporated into the e-voting architecture to satisfy the security requirements: an `Encryption` component to encrypt/decrypt the votes, an `Integrity` component to check their integrity, and a `Authentication` component to authenticate the voters. To do that, the SA can use a reusable library of FQAs independent from the application, customise them to the application requirements and add them to the base application architecture (Horcas et al. 2016). However, this is not an easy and straightforward task for the SA as we show in the rest of the section. Thus, the process has to guarantee that the FQAs have been applied semantically correctly to the software architecture of the application. In this paper, we focus on the encryption FQA.

2.1 The Encryption FQA

The encryption FQA (Figure 2) is defined as two pieces of advice: `encrypt` and `decrypt`, being applied in different join points of the architecture. The join point where the `encrypt` advice is applied must intercept a "sender component" (`C1` in Figure 2). This means that the `EncryptionImpl` component crosscuts the base application in the `C1` component through the `Encryption` interface (`I1`). Then the SA only has to decide the specific message type that must be sent encrypted (or whose confidentiality must be preserved). This requires to specify the specific method to intercept (`method1`, `method2`,...), the message (`a1`, `a2`,...), and the message's type (`T1`, `T2`,...) in the communicating interface (`I1`). If there are many sender components, interfaces, methods, or messages, this means that the encryption weaving pattern may be applied many times inside the software architecture, but in different join points. The join point where the `decrypt` advice is applied must intercept a "receiving component" (`C2` in Figure 2). The signature of the `decrypt` input parameter must be the same as the `encrypt` method's output.

If the message to encrypt is a result of the communication — i.e., a return type such as `ReturnType1` or `ReturnType2` in Figure 2, communicating components switch their roles. The "sender component" becomes `C2` whose encrypt the result before sending it, while the "receiving component" becomes `C1` and will decrypt the returned message after receiving it. Potentially, some such interactions may not need to be woven (*e.g.,* because the communication happens all in one host). However, at a minimum, the SA needs to identify all potential such points and verify whether the FQA needs to be woven there.

2.2 Applying the Encryption FQA to the E-Voting Application

The conditions to preserve confidentiality in the software architecture of the e-voting application are (1) "a message of type `Vote` must be always sent encrypted to guarantee confidentiality"; and (2) "the message sent before encryption must be equal to the received message after decryption".

To weave the encryption FQA into the software architecture of the e-voting application, we follow our current Aspect-Oriented SPL approach (Horcas et al. 2016) (summarized in Figure 3)[1] in which we separate the `Application Software Architecture`

[1] `FQAs' Constraint Models` is the enhancement of our approach and the main contribution of this paper.

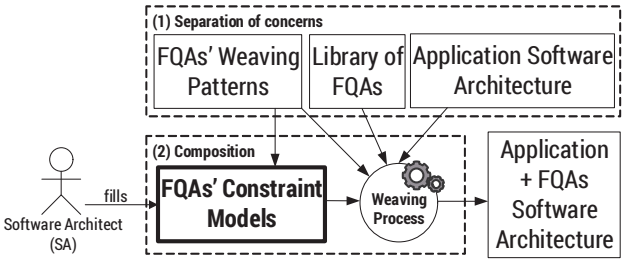

Figure 3. Our approach enhanced with constraints models.

from the `Library of FQAs` customised for the application requirements and ready to be incorporated into the application architecture. The transformation rules in charge of weaving the FQAs are defined in the `FQAs' Weaving Patterns` asset. Applying the `FQAs' Weaving Patterns`, the `Weaving Process` generates automatically the final software architecture of the application with the FQAs woven (`Appli-cation + FQAs Software Architecture`).

SAs are currently provided only with some pre-defined transformations for weaving FQAs, but not with any means of identifying the relevant join points and checking the correct application of the transformations to those join points. For example, the encryption pattern needs to be applied at the start and end of the communication between the `VoteClient` and `VoteServer` components, since they are deployed on different hosts (see e-voting application in Figure 1). However, in order to preserve confidentiality of the votes also in the counting process, the encryption pattern needs to be also applied between the inside components of the server (`VoteServer`, `ElectionData`, and `VoteCounting`) to avoid an internal attack. Moreover, the methods to be intercepted by the encryption pattern will be those related to the votes such as the `castVote` method of the `EVoting` interface, or the `storeVote` and `retrieveVotes` methods of the `VotingStorage` interface in charge of storing the votes and retrieving them for the counting process. The messages to be encrypted/decrypted will be the votes whose type is `Vote` in the application. Furthermore, it may not be efficient to apply the encryption pattern to join points where it is not needed, such as in the method to consult the results of the elections (`getResults`).

So, the SA has to identify and decide the join points where to apply the weaving pattern, and instantiate the parameters correctly to apply the pattern (*e.g.,* the message, the communicating components, etc.). If the encryption pattern is not woven in all and correct places, the resulting software architecture cannot fairly guarantee that the required confidentiality property is satisfied.

3. Defining Pre-Conditions for the FQAs Weaving Patterns

In order to help the SA in the process of choosing the correct join points where to apply the weaving patterns of the FQAs, and verifying the semantic correctness of the woven application architecture, we enhance the current FQA models (top of Figure 3) with additional models for structural pre-conditions and invariants (`FQAs' Constraint Models`). To this end, we extend FQA models with additional model fragments (called *constraint models*) that give an abstract description of the key architectural structures expected in a target software architecture.

For example, Figure 4 shows a constraint model for the encryption FQA. Structural elements of Figure 4 represent a simple pre-condition extracted from the encryption FQA pattern of Figure 2. The pre-condition is independent from the application, and can be read as "given a message type (called it `|T`), every component interaction in the target software architecture, where messages of type `|T` are exchanged, needs to be woven with the `encrypt/decrypt` advice." The chosen type can be the type of an input parameter of a method (`|PT`) or the return type of a method (`|RT`). The pre-condition explicitly defines that the weaving pattern will weave the `encrypt` advice "before" the message is sent and the `decrypt` advice "after" the message is received.

When weaving the FQA into a target architecture, these constraint models will be mapped on structures in the application software architecture to demonstrate that the FQA has been applied in the correct way. Checking of the mappings can be incorporated with the already existing weaving transformations or can be treated as a separate previous step. Once constraint models have been defined, they can be used in two complementary ways:

1. guiding SAs in selecting join points; and
2. verifying the choices of join points made by SAs.

We discuss each option in more detail in the following.

3.1 Matching Pre-Conditions to Identify Join Points

Following with our case study, the SA wants to weave encryption, but does not know where all the potentially relevant places in the application architecture would be. Checking the encryption pre-condition (Figure 4) with our e-voting application architecture (Figure 1) finds all possible *completions* where the encryption pattern can be applied (see Figure 5). Every interaction between two components through an interface matches the pre-conditions and represent a completion in the architecture where the encryption FQA pattern is suitable to be applied. When multiple different completions can be found, the application of the weaving is under-specified and the SA needs to provide more detail in a guided manner (*e.g.,* the component's name in case of multiple possible joint points). Thus, this is a largely manual approach requiring the SA to provide the majority of parameters for the application of a weaving pattern.

To facilitate this task to the SA, we can interpret the pre-conditions as an invariant/implicit for-all in a more traditional graph-transformation manner. The SA would provide specific instantiations for the desired parameters of the pre-conditions that conform the `FQAs' Constraint Models` (in graph-transformation terms a *partial match*) such as the method name `|m` through the communication of the target message is done, the name (*e.g.,* `|param`) or the type of the method's parameter that represents the message (*e.g.,* `|T` or `|RT`). In our case study, the SA might only want to provide an instantiation of the parameter `|T` with the value `Vote` and the interpretation would be that the weaving should be applied for all completions of this partial match: `Matchings 1`, `7`, `8`, `9` and `10` in Figure 5. In a supporting tool this might be realised as an automatic application of the weaving in all the right places.

3.2 Checking the Weaving Pattern's Application

The pre-conditions can be also used for checking the validity of the weaving pattern's application, depending on how safely the morphism can be completed. For example, there might be multiple completions concerning the same two components communicating such as in `Matchings 1`, `2`, `5`, and `6` of Figure 5; and only one of which would be correct. In such a case we may wish to give the SA the opportunity to make an explicit choice rather than automatically weaving to all of these completions. In our case study, `Matching 1` is the correct completion to preserve confidentiality of the votes in the communication between the `VoteClient` and `VoteServer` components. This is because the information to be encrypted (*i.e.,* the votes) are only in `Matching 1`, while `Matchings 2`, `5`, and `6` do not deal with the votes. Thus, the encryption pattern is applied to the completion represented as `Matching 1`. In a supporting tool, instantiating the pre-condition might be also realised as a check that

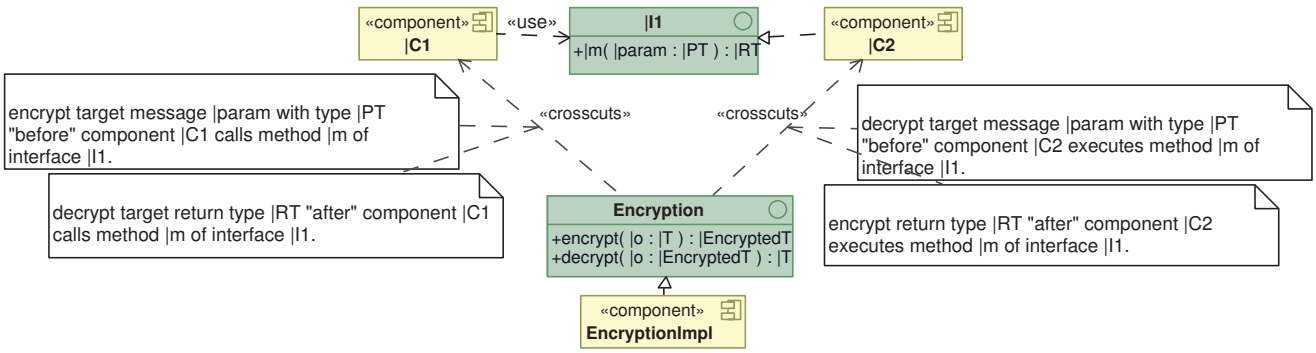

Figure 4. Pre-condition for the encryption FQA.

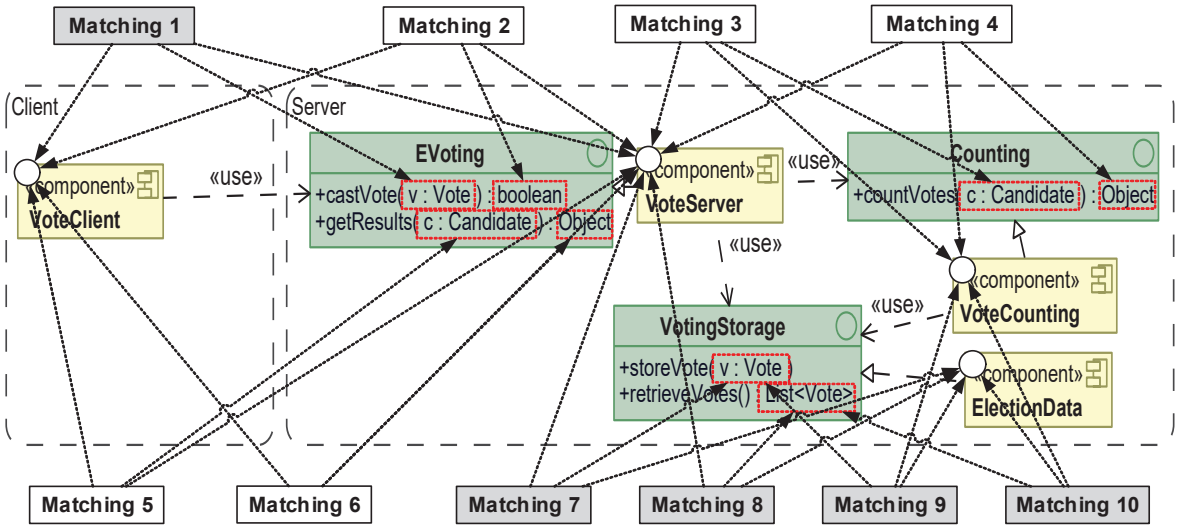

Figure 5. Completions of the encryption pre-condition in the e-voting application.

the SA has manually applied the weaving correctly in all correct places.

Similarly, the communications between the `VoteServer` and the `VoteCounting` need to be also encrypted to preserve the confidentiality of the votes in the counting process. So, the SA only has to instantiate the parameters of the pre-conditions for those completions represented in Figure 5 as `Matching` 7, 8, 9, and 10.

Figure 6 shows the final architecture of the e-voting application with the encryption FQA woven in the correct completions to satisfy the specified confidentiality requirements. Note that although the final application respects the confidentiality requirements specified in Section 2, the votes are stored in the storage device without encryption. This is because the pre-condition we defined specifies that encryption must be applied between two communicating components. In order to allow applying encryption between two independent (no-communicating) components (*e.g.,* to encrypt the votes before storing them, and to decrypt the votes after retrieving them) we need to defined a more complex pre-condition involving independent components.

We have tested the validity of the approach by implementing the pre-conditions of the FQAs' constraint models using the Henshin transformation language (Arendt et al. 2010).

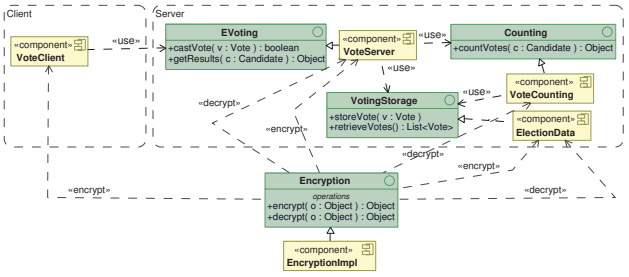

Figure 6. Correct completions for the encryption weaving pattern.

4. Related Work

There are few approaches that address the variability of the QAs from a functional point of view as we do using SPLs (Horcas et al. 2014, 2016). Juristo et al. (Juristo et al. 2007) propose an elicitation and specification approach for usability features with major implications on software functionality (*e.g.,* feedback, contextual help). They specify pre-conditions for the usability patterns but do not provide any automatic process or mechanism as we do, to incorporate the usability functionalities into the application

and to verify that the resulting application architecture satisfies the required QAs.

The RiPLE-DE (Cavalcanti et al. 2011) process is a domain design process for SPL that can be extended to model the FQA variability as part of a family of products. The variability of the FQAs is represented in feature model diagrams and in order to achieve desired quality levels, the models are enhanced with information of the base application. This makes the variability of the FQA components directly depend on the base application, preventing the FQA components from being reused. Our approach, in contrast, models separately the FQAs from the base application, improving the modularisation of the software architectures. As a consequence of a better modularisation, our approach improves the maintainability of the global system because changes in an FQA component affect only that component; and also the reusability improves because both base architecture and FQAs can be reused easier in different systems.

A similar approach built on the benefits of modularisation is the concern-oriented reuse (CORE) process (Alam et al. 2013). However, they model separately the variability of the interfaces of the concerns (*e.g.,* interfaces of frameworks or components) instead of modelling the variability of the internal functionality of the components as we do. Also, they focus on the impacts of the concerns on non-functional qualities (*e.g.,* access time, efficiency, etc.) by using goal models, while we focus on the operationalisation of the quality attributes (*e.g.,* the functional description of the encryption functionality). Moreover, they do not provide any mechanism to verify the semantic correctness of the woven application architecture, delegating the weaving process to the Reusable Aspect Models (RAM) weaver (Kienzle et al. 2009).

QADA (Matinlassi et al. 2002) is a specific method for designing SPL architectures by transforming systematic functionality and FQAs into software architectures, but this proposal does not explicitly take into account the requirements of the FQAs, so the semantic correctness of the final architecture cannot be checked, in order to assure the quality of the system.

5. Conclusions and Future Work

In this paper we have presented an approach towards the automation of the weaving process between FQAs models, already defined and taken from a library of FQAs, and the software architectures that needs to satisfy them. Concretely, the approach consists on defining for each FQA a constraint model with the pre-conditions for the correct weaving of that FQA. This means that we provide the SA with some help for identifying the join points where to insert each FQA. So, instead of manually doing the join point identification as existing approaches propose, the system suggests the SA a set of join points where to weave the FQAs and to verify the semantic correctness of the woven application architecture, in order to assure the quality of the system.

As regards future work, we plan to model the pre-conditions as part of the variability model of the FQAs, including the cross-tree constraints that relate the pre-condition models with the other FQA models previously defined (Horcas et al. 2016). This would allow the SAs to customise the constraint models for different systems that may require different pre-conditions. We also plan to automate the checking/guidance process of our approach to minimize the effort of instantiating the parameters of the pre-conditions by SAs, specially for those SAs that are not experts on aspect-orientation or quality attributes.

Acknowledgments

Work funded by the Spanish TIN2012-34840 (co-funded by EU with FEDER funds), and MAGIC P12-TIC1814 projects.

References

O. Alam, J. Kienzle, and G. Mussbacher. Concern-oriented software design. In *Model-Driven Engineering Languages and Systems*, volume 8107 of *LNCS*, pages 604–621. 2013. ISBN 978-3-642-41532-6. doi: 10.1007/978-3-642-41533-3_37. URL http://dx.doi.org/10.1007/978-3-642-41533-3_37.

T. Arendt, E. Biermann, S. Jurack, C. Krause, and G. Taentzer. Henshin: Advanced concepts and tools for in-place emf model transformations. In *Model Driven Engineering Languages and Systems*, volume 6394 of *LNCS*, pages 121–135. 2010. ISBN 978-3-642-16144-5. doi: 10.1007/978-3-642-16145-2_9. URL http://dx.doi.org/10.1007/978-3-642-16145-2_9.

F. Bachmann, L. Bass, M. Klein, and C. Shelton. Designing software architectures to achieve quality attribute requirements. *Software, IEE Proceedings* -, 152(4):153–165, 2005. ISSN 1462-5970. doi: 10.1049/ip-sen:20045037.

R. d. O. Cavalcanti, E. S. de Almeida, and S. R. Meira. Extending the RiPLE-DE process with quality attribute variability realization. In *QoSA-ISARCS*, 2011.

L. Chung, B. A. Nixon, E. Yu, and J. Mylopoulos. *Non-Functional Requirements in Software Engineering*. The Kluwer international series in software engineering. Kluwer Academic Publishers Group, Dordrecht, Netherlands, 1999. ISBN 0-7923-8666-3.

J. M. Horcas, M. Pinto, and L. Fuentes. Injecting quality attributes into software architectures with the common variability language. In *Component-Based Software Engineering*, CBSE, pages 35–44, 2014. doi: 10.1145/2602458.2602460. URL http://doi.acm.org/10.1145/2602458.2602460.

J. M. Horcas, M. Pinto, and L. Fuentes. An automatic process for weaving functional quality attributes using a software product line approach. *Journal of Systems and Software*, 112:78–95, 2016. doi: 10.1016/j.jss.2015.11.005. URL http://dx.doi.org/10.1016/j.jss.2015.11.005.

N. Juristo, A. Moreno, and M.-I. Sanchez-Segura. Guidelines for eliciting usability functionalities. *IEEE Transactions on Software Engineering*, 33 (11):744–758, 2007. ISSN 0098-5589. doi: 10.1109/TSE.2007.70741.

J. Kienzle, W. Al Abed, and J. Klein. Aspect-oriented multi-view modeling. In *Aspect-Oriented Software Development*, AOSD, pages 87–98, 2009. ISBN 978-1-60558-442-3. doi: 10.1145/1509239.1509252. URL http://doi.acm.org/10.1145/1509239.1509252.

M. Matinlassi, E. Niemelä, and L. Dobrica. *Quality-driven Architecture Design and Quality Analysis Method: A Revolutionary Initiation Approach to a Product Line Architecture*. 2002.