Master Thesis

May 14, 2014

SQA-Collaboration

A Real-Time Collaboration Framework for Distributed Software Quality Reviews

Robert Richter

of Grossenhain, Germany (11-782-232)

supervised by

Prof. Dr. Harald C. Gall

Martin Brandtner

University of Zurich UZH

s.e.a.l. software evolution & architecture lab

Master Thesis

SQA-Collaboration

A Real-Time Collaboration Framework for Distributed Software Quality Reviews

Robert Richter

University of Zurich UZH

s.e.a.l. software evolution & architecture lab

Master Thesis

Author: Robert Richter, robert.richter@uzh.ch

Project period: 15.11.2013 - 15.05.2014

Software Evolution & Architecture Lab
Department of Informatics, University of Zurich

Acknowledgements

I would like to thank Prof. Dr. Harald Gall for giving me the opportunity to write a thesis in his research group. Special thanks to Martin Brandtner for the general support and valuable feedback. Last but not least, many thanks to Dorothea Golze for her patience, feedback and repeated proof reading.

Abstract

Software quality reviews are an important part of the software development process. A software quality review is done by one or multiple stakeholders who have to collaborate. Either no single person has all the necessary knowledge or different kinds of stakeholders are responsible for different aspects of the software quality. Since distributed development is a common situation, the collaboration happens often remote via the Internet. We developed the web application SQA-Collaboration, which supports real-time collaboration in distributed software quality reviews. Furthermore, it supports the transfer and comprehension of the review results. The application provides user with shared workspaces, which enable them to query and arrange data as well as establish a shared view on it. The data of the repositories Jira, Sonar and Github can be imported and integrated in a semantic model. By means of the semantic information, queries are recommended to the users and relations between the data on a workspace are highlighted. An event history allows for the reconstruction of operations on the workspace and a replay of conducted reviews.

Zusammenfassung

Ein Review der Softwarequalität ist ein wichtiger Teil eines Softwareentwicklungsprozesses. Ein solches Review wird von einem oder auch mehreren zusammenarbeitenden Projektbeteiligten durchgeführt. Entweder hat keine einzelnen Person alles notwendige Wissen oder verschiedene Beteiligte sind verantwortlich für unterschiedliche Aspekte der Softwarequalität. Da verteilte Entwicklung eine häufige Situation ist, wird die Zusammenarbeit zwischen Projektbeteiligten oft mittels des Internets erfolgen. Wir haben die Web-Anwendung SQA-Collaboration entwickelt, die Echtzeit-Zusammenarbeit in verteilten Softwarequalitätsreviews unterstützt. Des Weiteren unterstützt die Anwendung den Transfer und das Verständnis der Review-Ergebnisse. Geteilte Arbeitsbereiche werden den Nutzern zur Verfügung gestellt. Sie ermöglichen es Auswertungen durchzuführen, Daten anzuordnen und eine gemeinsame Sicht auf die Daten herzustellen. Daten aus Jira, Sonar und Github können importiert und in einem semantischen Model integriert werden. Die semantischen Informationen werden genutzt um Nutzern Auswertungen vorzuschlagen und um Beziehungen zwischen Daten hervorzuheben. Eine Ereignishistorie ermöglicht es Operationen auf einem Arbeitsbereich nachzuvollziehen und durchgeführte Reviews zu wiederholen.

Contents

1 Introduction **1**
1.1 Motivation . . . 1
1.2 Goal . . . 3
1.3 Structure . . . 4

2 Related Work **5**
2.1 Code- and Software Quality Reviews . . . 5
2.2 Semantic Modelling in Software Engineering . . . 7
2.3 Computer Supported Cooperative Work . . . 8
2.4 Information Visualization and Interface Design . . . 10

3 SQA-Collaboration **13**
3.1 Application Concept . . . 13
3.1.1 Data Sets . . . 13
3.1.2 Data Integration and Queries . . . 16
3.1.3 Spaces . . . 18
3.1.4 Space Views and Space View Pages . . . 20
3.1.5 Space Types and Space View Types . . . 20
3.1.6 Space Events . . . 21
3.2 Application Design . . . 21
3.2.1 Control Panel . . . 22
3.2.2 Working Area . . . 24
3.2.3 Queries and Data Set Rendering . . . 27
3.3 Limitations . . . 35

4 Technical Design **37**
4.1 Used Technologies . . . 37
4.1.1 Scala . . . 37
4.1.2 Play 2 . . . 38
4.1.3 Apache Jena . . . 39
4.1.4 Javascript and CSS Libraries . . . 39
4.2 Implementation Details . . . 40
4.2.1 Semantic Model and Access of Data Repositories . . . 40
4.2.2 Spaces, Space Views and Data Sets . . . 41
4.2.3 Event Handling . . . 44
4.2.4 History Access . . . 45
4.2.5 Query Implementation . . . 46

4.2.6 Application Configuration 47
4.2.7 Data Set Rendering 48
4.3 Conclusion of Technical Details 48
4.3.1 Lessons Learned 48
4.3.2 Limitations 49

5 Evaluation 51
5.1 Study Setting 51
5.1.1 Study - Part 1 52
5.1.2 Study - Part 2 54
5.2 Data Collection 55
5.2.1 Study Population - Experience and Knowledge 55
5.2.2 Collaboration And Coordination 55
5.2.3 Feedback 56
5.2.4 SQAC User Behavior 56
5.3 Study Results 56
5.3.1 Study Population - Experience and Knowledge 56
5.3.2 Collaboration And Coordination 59
5.3.3 Feedback 61
5.3.4 SQAC User Behavior 62
5.4 Final Discussion 65

6 Final Remarks 67
6.1 Conclusion 67
6.2 Future Work 68
6.2.1 Support of Further Repositories 68
6.2.2 Review Process Support 68
6.2.3 Role Based Views 68
6.2.4 Mobile Device Support 69
6.2.5 Extended Workspace Organization 69
6.2.6 Extended Query Recommendations 69
6.2.7 Interaction Pattern Identification 69

A Handout Study Part 1 71

B Handout Study Part 2 73

C Self-Assessment Questionnaire 75

D Feedback Questionnaire 77

E Feedback Questionnaire Answers 79

F Contents of the CD-ROM 83

List of Figures

2.1 **The software evolution ontology pyramid.** Each layer represents a set of related ontologies. Generic concepts are represented by the top and very specific concepts by the bottom. Picture: [WGH+12] 8

3.1 **Overview of the SQAC application concept.** 14
3.2 **The concept of data sets.** This data set can be rendered as pie chart and as table. Each data set contains the semantic of its values. A superior ontology defines the semantic. 15
3.3 **The links between the different categories of data.** The data imported from Jira, Github and Sonar are linked in the semantic model. 16
3.4 **Query recommendation support by semantic information.** Since the semantic of the supported query parameters is defined, queries can be recommended based on the values of data sets. In the example, the query *List all sonar issues of the severity Info* can be recommended for a user that has clicked on the value *Info*. 17
3.5 **Example for space usage.** 1.) Alice (A) and Bob (B) have established a shared view on a data set. They are talking about it via Skype. Alice is not sure about it and she needs more information. She transfers the data set to space 2 in order to perform further investigations. 2.) Alice joins space 2. In the mean time, Bob is analyzing another data set. He is individually executing a query that creates a new data set. Alice is doing the same. After she found the necessary data set, she transfers it back to space 1. 3.) Alice joins space 1 and explains the transferred data set to Bob. . . . 19
3.6 **Entity relationship model of space, space view and their types.** Each space has one space type. Each space view has one space view type. A space type can support multiple space view types. A space can support multiple space views. 21
3.7 **The complete default space view page.** The page consist of a nested control panel (1) on the top and a working area (2) below it. The control panel and its sub-panels can be collapsed. In the example, the sub-panels are collapsed. The working area can be compared to a whiteboard. Two data sets are shown on the working area. Only a part of the working area can be seen. The menu bar (3) on the very top is not part of the space view page. The menu bar can be used to create, join, export and import spaces. Furthermore, a description of all queries can be accessed. . . . 22
3.8 **The control panel of the default space view.** The query panel contains a drop-down list of available queries (1). A form for the selected query has been rendered. The clients panel is showing that currently only one user is connected to the space. In the history panel a previously executed query event has been selected (2). The space could be temporarily reset to the selected state by clicking *Set* (3). The currently initialized state is drawn in grey (4). The latest state is on the very right end of the event list. In the example, it is *REMOVE (Bob)*. 23
3.9 **A data set container with framed areas.** The structure of each data set container is as follows: A menu bar on the top (1), a content area in the middle (2) and a comment area in the bottom (3). 25
3.10 **A data set with context menu.** By clicking on the values of a data set, a context menu (1) is created. This contains the option to highlight related data sets and a list of query recommendations. In the example, the recommendations have been framed green. A selected DS and a selected value are highlighted by a blue frame on the working area. 26

3.11 **A working area with highlighted data sets.** Four data sets are display on the working area. All data sets related to a specific commit have been highlighted: The details of a commit (3), the diff of a file change (4) and the details of a Jira issue (5). One data set is not related to the commit (6). Therefore, it has been made transparent and placed in the background. The button *Remove Highlighting* (1) will deactivate the highlighting. The button *Organize Space* (2) will organize the data sets on the space. 27

3.12 **Example of data traversal via queries.** Queries: 1.) Code of Component with marked Sonar issues 2.) Commits related to a specific component 3.) Details of Jira issue. 28

3.13 **Data set showing Sonar issues in a project.** The table contains one issue per row. Issues with the severity MAJOR in the project CAMEL and with a creation date before the 30.03.2014 are listed. Each row, the project and the severity can be clicked, in order to create the value context menu. 29

3.14 **Data set showing code with marked Sonar issues.** Sonar issues are marked red in the code. The description of the issue is displayed below the marker. The issues and the component name can be clicked in order to display the value context menu. 30

3.15 **Data set showing the details of a Sonar issue.** The project (CAMEL), the component name, the description of the issue, the corresponding code snippet and the issue key are displayed. All values can be clicked in order to display the value context menu. 31

3.16 **Data set showing the process of Sonar issues.** The process of the number of Sonar issues in the project CAMEL between the 25.03.2014 and the 05.04.2014 is shown. Each severity is displayed in a single data series. 32

3.17 **Data set showing Jira issue details.** The project (CAMEL), the issue key (CAMEL-7145) and the issue name are shown on the very top of the content area. Four main sections exist: Details, Description and Activity. The Activity section can contain the sub-sections JIRA Comments and Related Commits. 33

3.18 **Data set showing the details of a commit.** The key of the commit, the committer (author), the commit time, the commit message and the name of all affected files are shown. 34

3.19 **Data set showing the diff of a file change.** The key of the related commit, the commit time, the name of the affected file and the file diff are shown. 34

4.1 **Entity relationship model of space and related entities.** Relation between Space, SpaceType, SpaceView, SpaceViewType, DataSet and ViewDataSet. Each DataSet is associated to one ViewDataSet. Each SpaceView is associated to one ViewDataSet per DataSet, that is associated with the corresponding Space. 42

5.1 **Schedule of the evaluation study.** 52

5.2 **Box-plot of development experience of the participants.** 57

5.3 **Box-plot of tool and SQA knowledge of all participants.** 58

5.4 **Box-plot of tool and SQA knowledge of group A.** 58

5.5 **Box-plot of tool and SQA knowledge of group B.** 58

5.6 **Result of group A in session one, part one.** 63

5.7 **Result of group A in session two, part one.** 64

List of Tables

2.1 **Johansen's time-space matrix.** Groupware can be categorised in respect to the time and the place of user interaction [Joh88]. 9

5.1 **Details of change request CAMEL-7333.** 53

5.2 **Details of change request CAMEL-7327.** 53

5.3 **Details of change request CAMEL-7312.** 53

5.4 **Details of change request CAMEL-7334.** 53

5.5 **Absolute number of answers concerning tool and SQA knowledge.** The scale of Jira, Github and Sonar knowledge was 0 to 3. For SQA a scale of 1 to 4 was given. 57

5.6 **Overview of communication of SQAC groups.** In part 1 the groups A and in part 2 the groups B used SQAC. 60

5.7 **Overview of communication of SOTA groups.** In part 1 the groups B and in part 2 the groups A used the SOTA tools. 60

5.8 **Comparison of communication during SQAC and SOTA usage.** 60

List of Listings

4.1 Examples of Scala language features 38

4.2 Example of Play 2 JSON handling 39

4.3 Example of Apache Jena data import 39

4.4 Specification of the class ValueSemantic 41

4.5 Specification of the trait SpaceViewType 41

4.6 Specification of the trait SpaceType 42

4.7 Shortened specification of DataSet and SingleDataSet 43

4.8 Shortened version of the DataSetModifyEvent handling 44

4.9 Specification of Query, QueryInput and SupportedInput 46

4.10 The trait ApplicationConfiguration 47

List of Abbreviations

CE	Client Event
CEM	Client Event Message
CQ	Code Quality
CR	Code Review
CRQ	Change Request
CSCW	Computer Supported Cooperative Work
DS	Data Set
DSL	Domain Specific Language
DSME	Data Set Modification Event
DST	Data Store
DSTP	Default Space Type
DSVP	Default Space View Page
DSVT	Default Space View Type
ED	Event Dispatcher
EH	Event History
EHI	Event History Item
GOFP	Gang Of Four Design Patterns
JVM	Java Virtual Machine
MVC	Model-View-Controller
RDF	Resource Description Framework
SE	Space Event
SEON	Software Evolution Ontology
SI	Supported Input
SM	Server Message
SOTA	State-Of-The-Art
SQ	Software Quality
SQA	Software Quality Assurance
SQAC	SQA-Collaboration
SQR	Software Quality Review
ST	Space Type
SV	Space View
SVE	Space View Event
SVP	Space View Page
SVT	Space View Type
URI	Uniform Resource Identifier
VCM	Value Context Menu
VDS	View Data Set

Chapter 1

Introduction

1.1 Motivation

The quality of software is of importance for both, the users and the producer of the software [KP96]. The users want to satisfy whatever need is addressed by the software. The producer want to develop and sell the software with a high return. *Software Quality* (SQ) is a rather fuzzy concept and very dependent of the specific point of view [KP96]. Whatever definition of SQ is used, regular reviews of the software are necessary in order to assess the current level of quality. Based on the review results it can be decided, if the quality level is satisfying or actions to increase it are necessary.

The development of software is an endeavour that includes multiple stakeholders, who assume different roles and have different responsibilities [MF13]. Those stakeholders are interested in various kind of information and have specific knowledge about the software [RL02]. For example, the product owner may be interested in the specification compliance of the software. The chief of user experience may know why a specific part of the user interface specification was not implemented. Hence, a *Software Quality Review* (SQR) will most likely include multiple stakeholders. Either, no single person has all the necessary knowledge or different kinds of stakeholders are responsible for different aspects of the SQ.

The overall goal of this thesis is to support collaborative SQRs. It is motivated and shaped by three use cases, that are related to this area.

Use Case 1: Collaborative Data Exploration

It can be necessary for different stakeholders to collaborate in an SQR in real time. Distributed development with scattered teams is a common situation, especially in international companies. This can be due to a lack of local manpower or the try to save money by off-shoring [SSEB10]. Parts of a team may also have to travel a lot or work in home office, regularly. Hence, a collaboration will often not happen in person, but remote via the internet. Use case 1 assumes such a scenario: A real-time SQR that is collaboratively performed by participants that are geographically distributed. The following scenario gives an example of such a situation.

Scenario. Frank implemented a *Change Request* (CRQ). The impact of the changes on the software quality shall be assessed. The task has been assigned to Bob. Since Bob is not very familiar with the corresponding software, he is supposed to ask the more experienced Susan for help. Susan works in home office most of the time and is seldom available in the office. Therefore, Bob and

Susan have to perform the collaborative quality review distributed. They want to be able to see the same data in order to enable Susan to give advice to Bob. It is possible they find some piece of data that Susan cannot explain without further investigation. For example, she needs to look up some old user stories in order to be able to justify a specific design decision. In such a case, she wants to search for the necessary information without keeping Bob from preparing another part of the review.

Implications. A scenario such as the one just described includes challenges. A shared view can be established rather straight forward by means of a remote desktop session or via a Skype screen sharing. Though, this is only true as long as only two participants collaborate. No support for real collaboration without stopping the other participants work is given. A read only view prevents interaction of the remote participant. It must be possible for Susan, in the example above, to access data and interact with the data. She must be able to transfer interesting data to her computer without the loss of context. In the scenario above, Susan is looking for a user story. Assumed, that a diagram that shows a massive increase in the application response time was the reason for this. Probably, she wants to see this diagram while searching. E.g. to look up the specific date or as origin for further search actions.

Use Case 2: Transfer of Review Results

An SQR is only the first step in an overall process. The results of the review have to be transferred to somebody who is responsible for deciding further steps. A review result could be the source of information for developers who must implement improvements. A product manager could need to review the results in order to identify tasks and adapt a project schedule accordingly. The second use case addresses such transfers of review results. In the following, an extension of the first scenario will describe an instance of this use case.

Scenario. Frank implemented a CRQ. The impact of the changes on the SQ shall be assessed. After Bob and Susan performed the corresponding review, they have to transfer the results to Frank. He must examine the results in cooperation with Alice, the product manager. Frank and Alice are not sure about the validity of some of the review results. They need to understand how Bob and Susan produced these results. Afterwards, Alice and Frank select a number of valid issues, which shall be addressed by future actions. In order to decide this actions, they have to look for further data related to the issues.

Implications. The transfer of a review result is just one example for a necessary knowledge transfer in a software project. Such transfers have to happen multiple times between different stakeholders in different situations [RL02]. An inefficient or incomplete transfer of knowledge can affect the success of a software project. The persons depending on the quality review results have to understand the data and must be able to comprehend how the results have been produced. A successful transfer has two sides. It should be easy for the reviewers to create a complete and comprehensible result as well as transfer it to other stakeholders. Ideally, the creation of the result is part of the review process itself and supported by a tool. On the other side, it should be easy for the receivers to work with the result.

Use Case 3: Analysis of Quality Reviews

It can be interesting to analyse which actions have been undertaken in an SQR. This can not only be interesting for practitioners, but also for researchers. Such an analysis can help to answer different questions. Is it possible to identify interaction patterns? Can this patterns be applied in general? Can the process be made more efficient? If yes, how? The analysis of SQRs is not in the direct scope of this thesis. Nevertheless, such an analysis should be supported.

Scenario. Peter is supposed to identify typical steps in SQRs that are performed in the company he is working for. The goal is to identify a set of best practices that can increase the efficiency of such reviews. Furthermore, it shall be identified which kind of data is normally needed and used by the reviewers.

Implications. A proper tool support can ease a task like the one assigned to Peter. It must be possible to identify not only single steps that have been performed in the review, but also what kind of data has been accessed. The necessary information will most likely not be available in a manual review protocol, since it would require participants to actively comment and describe each of their actions. An ethnographic approach would be possible but complex. The actions of the participants and the used data must be categorized manually.

1.2 Goal

The goal of this thesis is to design and implement an approach that supports the use cases described in the Section 1.1 and addresses the identified implications:

1. Participants must be able to establish a shared view.
2. Both, individual and collaborative data exploration must be supported.
3. For individual data exploration, data must be transferable without the loss of context.
4. The creation of a review result must be supported directly.
5. The understanding and comprehension of review results must be supported.
6. It must be possible to analyze and comprehend the steps of a review.
7. It must be possible to analyze what kind of data has been accessed during a review.

To summarize: The approach should support the collaborative data exploration in an SQR as well as the easy creation and transfer of the review results. Furthermore, it should support the analysis of the review process. These goals are summarized in the following research question:

How can a distributed software quality review be documented to enable a replication of the review?

After a prototype has been implemented, a first evaluation will be conducted. It will generate user feedback and usage data that helps to identify strengths and shortcomings of the developed application.

1.3 Structure

The thesis is structured as follows: Work related to the topic of this thesis is discussed in Chapter 2. The application *SQA-Collaboration* (SQAC) was developed in the context of this thesis. A description of this application is given in Chapter 3. The technologies used for the application development as well as technical details of SQAC are presented in Chapter 4. An exploratory study has been performed with the developed application. In Chapter 5, the design and the results of this study will be discussed. The Chapter 6 contains a conclusion and some topics of future work.

Chapter 2

Related Work

Code Reviews (CRs) are an important part of a software development process. However, the *Code Quality* (CQ) is only a subset of the SQ. CRs can be seen as a special case of SQRs and existing work in this area must be considered. The distinction between the both types of reviews and relevant work in the respective areas is discussed in Section 2.1. The usage of a semantic model can enable functions that may support collaborative quality reviews. The approach to be developed in this thesis will try to utilize such a model. Related work in the area of semantic modelling will be discussed in Section 2.2. The support of the cooperation of people in work processes by means of software is the research topic of *Computer Supported Cooperative Work* (CSCW). CSCW and its relation to this thesis is discussed in Section 2.3. An SQR will include the analysis of different kinds of data. Such analysis must be supported by means of good user interface design and proper information visualization. This topics and related work are discussed in Section 2.4.

2.1 Code- and Software Quality Reviews

CRs are a crucial part of the assurance of SQ [Bak97]. They can be performed on different scales and intervals. A typical approach is the review of code changes before they are committed back to the repository. Such an approach is used in the Mozilla Firefox development process, where patches must be reviewed before they are delivered as part of a release [NNKR09]. A lot of work has especially concentrated on the CR processes of open source software [RG06, BC12, NNKR09]. A benefit of regular CRs is not only the assurance of the SQ, but also the implicit transfer of code knowledge [NNKR09].

Nevertheless, the quality of a software is more than the quality of its code. SQRs can already be executed in a phase, where no code has been written so far. For example, in [SSEB10] a process is described that requires different pre-code artefacts to pass specific quality gates. In [Gar84] five views of product quality in general have been identified. For example, one could define quality relative to the satisfaction of the users. However, the quality of a product could also dependent on its specification. The higher the degree of specification fulfilment, the higher the product quality. A high quality from one perspective must not necessarily correlate with a high quality from a different point of view. User can be very satisfied with a product, that only fulfilled half of its original specification. The different views on quality can also be assumed in the context of SQ. There is no single true definition of SQ and different stakeholder will take different points of view [KP96]. Hence, an SQR must consider very different kind of data.

Multiple tools exists that support collaborative CR processes, as *Review Board*[1], the *Atlassian Crucible*[2] and *Codestriker*[3]. The Review Board was the subject of a scientific survey [NNKR09] as well as the application Codestriker [Rem05]. Though, none of those tools addresses the more general case of an SQR or distributed real-time collaboration. The application *Smartbear Collaborator*[4] allows to establish a shared view on data. The commenting of different kinds of documents is supported. Though, no dynamic or collaborative exploration of underlying data is possible. Furthermore, the aspect of the transfer of review results is not addressed by this tool.

In [JT93] a tool named *CSRS* is described, that supports the collaborative review of different software artefacts. Nevertheless, the authors did focus on CRs. The underlying concept is a network of different kinds of nodes. Examples of such nodes are *Source Nodes*, *Comment Nodes*, *Issue Nodes* and *Action Nodes*. By means of the application a user can review a code artefact represented by a source node. If she identifies a code issue, she can create a new issue node and link this node with the corresponding source node. An underlying process organizes the review, that consists of multiple serial steps. First, a moderator creates the source nodes to be reviewed (*Source node generation*). Second, participants are prepared for the review (*Orientation*). Third, all participants review the source nodes (*Private review*). In this step, participants do not have access to issues, comments etc. created by other participants. Each node must be explicitly marked as reviewed by each participant. In the fourth step, all nodes created in the previous step can be accessed by all participants (*Public review*). New nodes can be created, that can comment, criticise or extend existing ones. The step will end, if all nodes have been marked as reviewed by the reviewers. The moderator will create a consolidation document in the fifth step (*Consolidation*). It is created with the support of the CSRS tool and contains the current results of the review. An optional sixth step will be conducted if conflicts between the participants opinions are identified via the result document (*Group review meeting*). In a group review meeting the corresponding conflicts are reviewed and discussed. The process ends with the implementation of actions that have been defined during the review (*External development*). None, but the optional sixth step, needs participants to collaborate in real-time. The different steps can be performed asynchronous.

The approach described above differs in ambition and implementation from the one developed in this thesis. The authors of former do not concentrate or particularly support real-time collaboration in reviews. No support for the establishment of a shared view is given. The creation of the review result is a distinct step and not a by-product of the review itself. Exploration of data means traversing a predefined node network, while this thesis targets on more dynamic data exploration. Nevertheless, a certain interception exists in the goal of overhead reduction and ease of review result creation. Furthermore, CSRS would support the reviewing of different kinds of artefacts. Although, the implementation of a specific review process is not in the scope of this thesis, the guided review process supported by CSRS could be inspiring for future work. The authors try to support the analysis of the process itself by the creation of user action statistics.

In [MWFG12] an approach is presented, that focus on real-time collaboration during CRs. The authors describe the tool *SmellTagger* that is utilizing a shared multi-touch device. The general approach is well suited for CRs and could be part of a superior SQR. SmellTagger supports the automatical creation of review results. This intercepts with the goals of this thesis, regarding efficient result transfer. In comparison to our work, the focus of SmellTagger is on the support of in-persona reviews with one shared device.

[1]http://www.reviewboard.org
[2]https://www.atlassian.com/software/crucible
[3]http://codestriker.sourceforge.net
[4]http://smartbear.com/products/software-development/code-review

The topic collaborative CRs is tightly coupled with SQRs, but only a subset of the latter. Existing work to such CRs can be used as inspiration and as marker for future work. However, to our best knowledge no work exists that addresses the topic of collaborative SQRs and focuses on the use cases described in Section 1.1.

2.2 Semantic Modelling in Software Engineering

The approach to be developed in this thesis will utilize a semantic model. Data stored in such a model can easily be transferred and processed, since its semantic is uniquely described.

Data in a semantic model is stored as *Resource Description Framework* (RDF) triples [KC04]. Such a triple corresponds to a statement of the format *subject predicate object*. In a relational model, a concept like a car would be stored in a table *Car* with columns like *Vendor*. In an RDF based model the information would be stored as statements like *Car1 hasVendor Honda*. The subject of a statement is a resource. It represents one single thing like a specific car [WRDG10]. A resource is part of a specific *class* of things. Such resources are uniquely identified by a *Uniform Resource Identifier* (URI) [BLFM98]. In the example above, *Car1* is the URI of a resource that represents one instance of a car. The object of a statement can be a resource itself or a simple value like a string. Modelling data with RDF means defining classes of things and *properties* that can be used as predicates in statements. The *SPARQL Protocol And RDF Query Language* (SPARQL) can be used to query RDF data. RDF does not constrain the way a specific concept is modelled. Hence, a car could be represented by a class *Car* in one case and a class *Automobile* in another. In order to deal with this problem, an *ontology* can be defined and used. It models the concepts of a specific area. Therefore, it defines what statements are possible in this area [Gru93]. Ontologies exist for different application areas and can be used in order to describe corresponding data uniformly.

In [WRDG10] the usage of semantic models in the area of software engineering has been proposed. The authors argue, that it would be beneficial to describe the data of software repositories with a uniform semantic model. This would require the definition of an ontology for the domain of software engineering. Such an ontology has been described in [WGH$^+$12] and is of high importance for this thesis. The domain of SQRs is the domain of software engineering. Hence, software artefacts, their properties and their relation must be stored in the underlying semantic model.

The *Software Evolution Ontology* (SEON) is structured as a layered pyramid. This SEON pyramid is shown in Figure 2.1. From the top to the bottom, the following layers exist: *General Concepts*, *Domain Spanning Concepts*, *Domain Specific Concepts* and *System Specific Concepts*. Hence, the represented concepts get more specific from the top to the bottom. The more specific ontologies are based on the more generic ones. The layered and modular design of SEON allows to select only the subset of ontologies that is needed. In the context of this thesis, SEON will be used for the description of the underlying data.

As written above, a semantic model supports the transfer of data. A further benefit is the statement-based data representation in general. It supports queries in a nearly natural language. This has been shown in [WGRG10] in the context of an Integrated Development Environment. The possibility to query SQ related data in such a way could support and ease the data exploration in SQRs. Implicit knowledge in a semantic model can be inferred by means of reasoning [WGG13]. For example, the inverses of specific statements can be added to the model automatically.

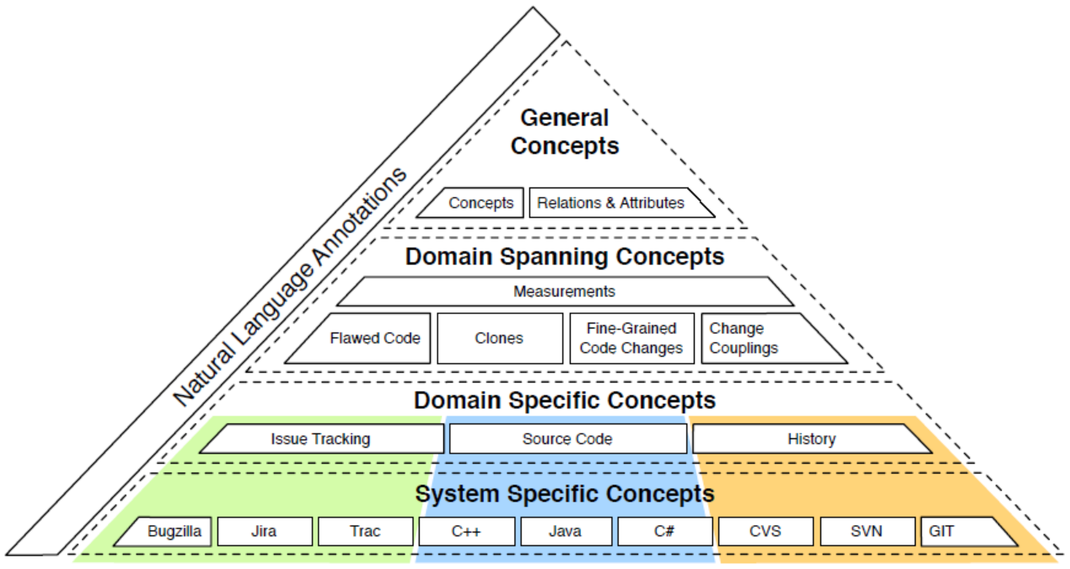


Figure 2.1: **The software evolution ontology pyramid.** Each layer represents a set of related ontologies. Generic concepts are represented by the top and very specific concepts by the bottom. Picture: [WGH+12]

2.3 Computer Supported Cooperative Work

The research topic of the field of CSCW is the support of group work processes by means of software. Such software is named groupware [Gru94].

Groupware can be categorized in multiple ways. Johansen [Joh88] did introduce a time-space matrix. It supports a categorization based on the time and the place of the interaction of users. The matrix is shown in Table 2.1. The categories of the matrix are *same time*, *different time*, *same place* and *different place*. Hence, an email program would be categorized as a different time - different place groupware.

In [DFAB93] a categorization based on the primary function of the groupware can be found. It distinguishes between the support of direct communication between participants (*computer-mediated communication*), the support of a common understanding (*meeting and decision support systems*) and the support of the interaction with shared work objects (*shared applications and artefacts*).

A further classification is based on the type of interaction, as described by [TSMB95]. The used categories of interactions are *communication*, *coordination* and *collaboration*. A groupware is categorized by its level of support of the different kinds of interactions. Nevertheless, an application can support multiple of the interaction types by different degrees. Depending on its focus it can be collaboration-, coordination- or communication-oriented.

It can be useful to categorize a specific groupware in order to identify approaches and challenges which are common for this category of application. The application to be developed in the context of this thesis can be described as collaboration-oriented. It will mainly focus at a different

	Same Time	Different Time
Same Place	Face to face interaction	Asynchronous interaction
Different Place	Synchronous distributed interaction	Asynchronous distributed interaction

Table 2.1: **Johansen's time-space matrix.** Groupware can be categorised in respect to the time and the place of user interaction [Joh88].

place and same time (hence, synchronous) interaction. However, a clear categorization is difficult. For instance, the transfer of the review results is not matching this categories.

In [SSU01] a set of properties of *collaboration-oriented synchronous groupware* can be found. According to the authors, collaboration should be supported by means of *shared objects*, *views*, *sessions* and *awareness*:

Shared objects are the means and the targets of the collaboration. In the context of a cooperative text editor, as provided by GoogleDocs, a document as well as its text would be the objects. The participants interact with and modify those objects. A problem introduced by shared objects is the overall consistency. Hence, different user operations could lead to conflicts and inconsistent object states. This can be addressed by means of optimistic or pessimistic methods.

Users must be able to establish a shared and synchronized view on the objects. The views of all users must not be synchronized all the time. Users should be able to work on their own without interfering with the work of other users. It can be distinguished between loose- and tight coupled views. A loose coupled view ensures the same state of the shared objects, but allows users to assume different points of view. In a shared document, the text would be in the same state, but users could scroll to different parts of the document. Tightly coupled views enforce an identical view on the shared objects.

It should be possible to structure the collaboration in multiple work sessions. This is due to the fact, that a work process may consist of multiple different steps. These steps might be performed by different group constellations. Work sessions can apply different user roles and interaction rules.

In a collaborative work scenario a group of people are working together. Participants need a certain awareness of each other and each others actions. Furthermore, it is necessary that a participant is aware of the current state of the overall work process. Without this information, a coordination of work and hence a real collaboration is hardly possible. Which level of awareness is needed, depends on the specific kind of work.

A groupware focusing on collaborative synchronous work must not only support collaboration but also coordination and communication. The coordination includes restrictions regarding the access of shared objects, for example, by means of roles. Communication will always be needed in a collaborative scenario. Participants need to communicated in order to coordinate their work and to establish a shared understanding. The communication can be supported via the shared objects itself or by external mechanisms.

2.4 Information Visualization and Interface Design

In an SQR, complex data must be comprehended by the reviewers. This can be eased by good information visualization. This is the *"computer generated interactive graphical representations of information"* [Che10]. For instance, such visualizations allow the viewer to comprehend huge amounts of data and supports the identification of patterns [War04].

Information visualization is not a research topic of this thesis. Though, central principals of efficient visualization are important for every area that deals with data analysis. Pie- and time series charts are some of the most common ways of data rendering. As discussed in detail by Tufte [Tuf86], the former helps to compare categories of data and the latter to comprehend the process of data. A further strength of time series charts is the possibility to identify correlations in the process of different categories of data. The visualization of aggregated data in the context of this thesis will focus on this basic chart types.

Since multiple kinds of data will be needed to perform an SQR, a single way of visualization is not enough. Instead, multiple ways of information visualization will be combined. A pie chart may be rendered next to a time series chart and a table. A central question of the user interface (UI) design is, how to support the comprehension of such combined data. Shneiderman [Shn96] describes a set of seven useful design guidelines: A user should be provided with a general *overview* of the data. It should be possible to *zoom* into the data and examine details. The user should be able to request interesting *details on demand.* If a set of data is not interesting for the user, he should be able to *filter* it. *Relations* between presented data should be visualized. A *history* should allow the user to comprehend his actions or maybe even undo them. It should be possible to *extract* visualized data for future investigation.

As one can see by the guidelines described above, the visualization of information and the general user interface design are tightly related. Most likely, a user must interact with an application before information is visualized and he must be able to interact with the visualization. Raskin [Ras00] describes a set of design principals that can help to build interfaces, that are suited for human beings. According to him, one has to consider the human cognition and its shortcomings in order to create such a *human* interface. For example, an interface designer must be aware of the fact, that regular work with an interface will and must create user habits. Due to such habits, the check of a delete action via a confirm dialogue provides only limited security. A habitual user, who is performing the delete action multiple times every day, will most likely simply press OK even if he is going to delete the wrong object. A selection of Raskin's design principals is shortly described in the following:

Visibility/ Affordance: Two important design concepts are *affordance* and *visibility*, as first introduced by Norman in [Nor02]. They address the fact, that an interface should always clearly indicate what actions are possible (visibility) and how this actions can be performed (affordance). A function that can only be executed by a key combination has no visibility. An example of a control element that has no affordance would be a button, that must be turned instead of pushed.

Modelessness/ Monotony: An interface is in a mode in respect to the effects of an operation. If the rubber is selected in Microsoft Paint, a mouse click will erase content. If the pen is selected, the same action will create content. Hence, both states are modes of the interface. Modes should be avoided as far as possible. This is due to the fact, that a user's attention can always be only at one thing at a time. The user has only one *locus of attention.* In order to operate the software without errors, the user has to be aware of his current action and of the mode of the UI. This cognitive overhead should be avoided. An interface is monotonous if each possible effect can only be produced in one way. The user should not need to decide in which way to achieve the same result. Modelessness and monotony are related, since both address unnecessary cognitive overhead for the user.

Unification: Every UI element that looks the same, should also trigger the same kind of action. The same should be true for general operations. The same kind of operation should always be performed in the same way. For example, if a selection of a picture in the file system is performed via a left mouse click, than every selection should be performed in this way. Hence, also the selection of folders, text files and audio files. Unification is also called *Duck Designing*. It is tightly related to the concept of affordance.

The design of the application to be developed in this thesis will be guided by the design rules of Shneiderman, Raskin and Norman. Information visualization and user interface design are tightly related to the CSCW concepts described in Section 2.3. For example, awareness must be supported via information about actions of other users. How this information is presented is defined by the general interface design and the used information visualization concept.

Chapter 3

SQA-Collaboration

In order to support the use cases described in Chapter 1 and to investigate the corresponding research question, the application SQAC has been developed. The general concept of SQAC is described in Section 3.1. The design of the application and its functions are described in Section 3.2. Limitations are discussed in Section 3.3.

3.1 Application Concept

In the following, a short overview of the application concept is given. A graphical representation is shown in Figure 3.1. The different aspects of the concept are described in detail in the next sections.

A central concept of our approach is to provide users with shared *workspaces* (short: *spaces*). Shared spaces haven been successfully applied in other areas of CSCW. For example, applications for conference support, that utilize shared spaces, are described in [SSU01]. A well known example for a shared space application is the word processor provided by GoogleDocs. Basically, a shared space is some kind of working area that can be utilized by multiple users simultaneously. The general concept of SQAC is based on the *collaboration-oriented synchronous work* paradigm introduced in Section 2.3.

Spaces contain *Data Sets* (DSs) which are self-contained collections of data. For instance, a DS could represent all bugs that have been fixed at a specific day. DS are created by means of predefined queries. Queries are executed on a central *Data Store* (DST), which integrates the data of multiple repositories like Sonar, Github and Jira. Multiple users can use a space simultaneously in order to execute queries and interact with the DSs. User access spaces via *Space Views* (SVs). They define how the DSs of a space are rendered. The rendering happens by means of HTML pages, named *Space View Pages* (SVP).

3.1.1 Data Sets

As described in Section 2.3, shared artifacts are the central means of collaboration provided by a groupware for synchronous work. In a shared text editor, the text itself is the shared artifact. The users can write and modify the text collaboratively and establish a shared view on the same piece of text in order to discuss it. In an SQR a multitude of very different kinds of data must be investigated. This can be a single piece of information like the date of a commit or aggregated data, e.g.,

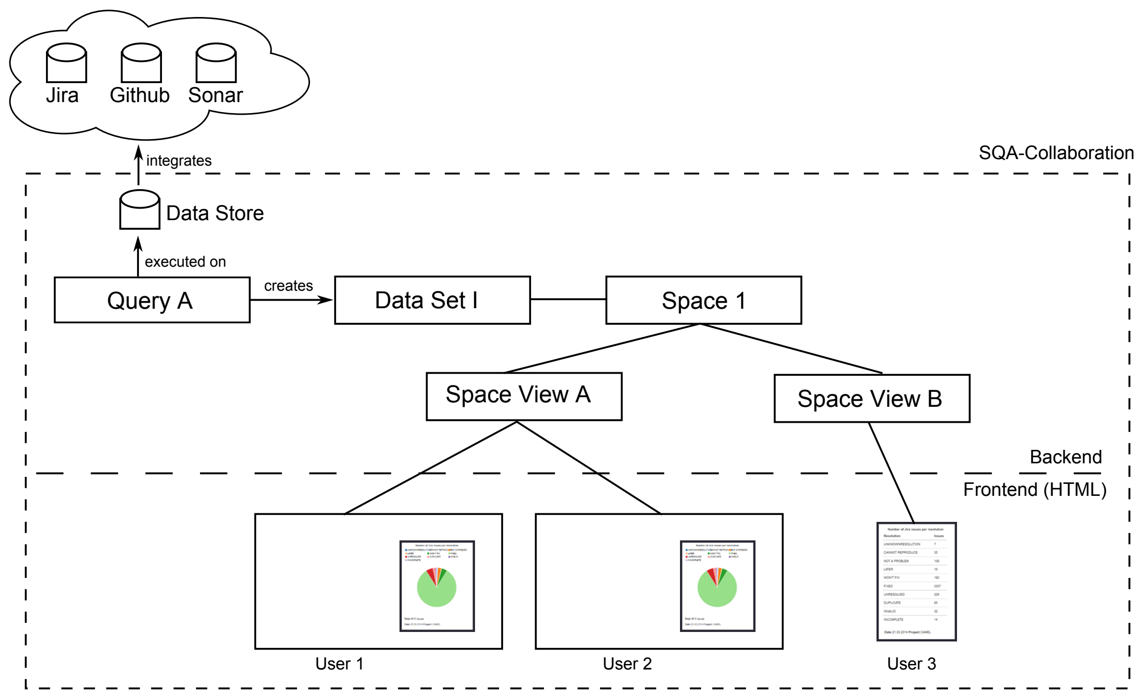


Figure 3.1: **Overview of the SQAC application concept.**

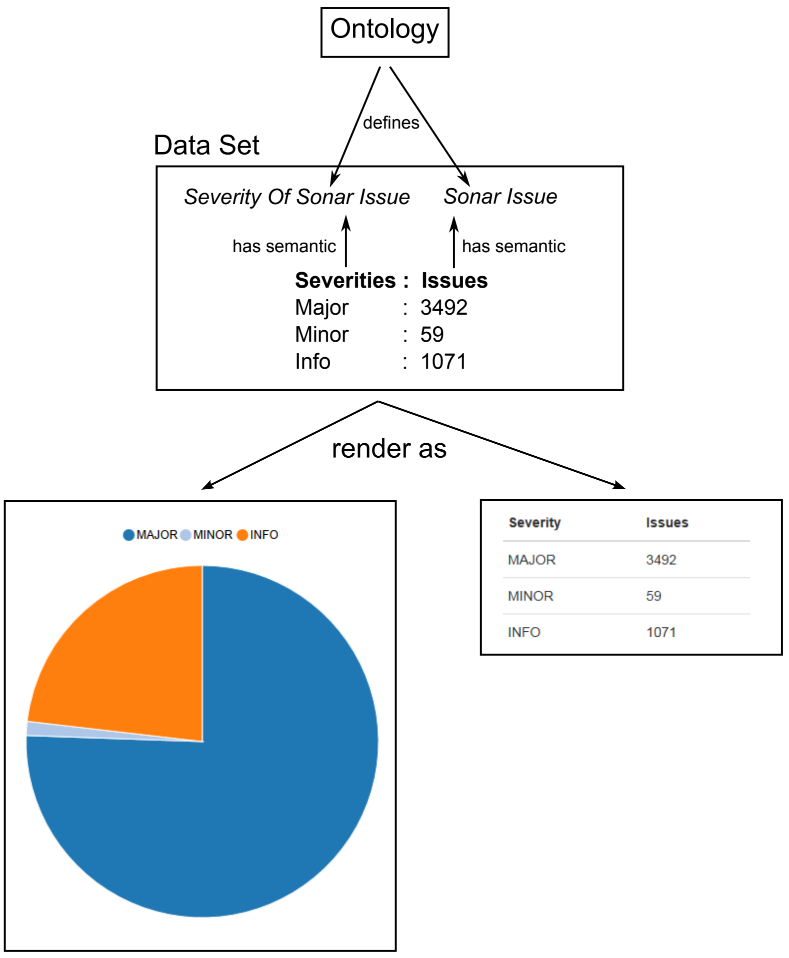
the number of bugs per severity. Depending on the type of data and data granularity, different ways of visualization are needed. In a collaborative SQR, the participants want to see, access and discuss these different types of data. Since participants of an SQR deal with such diverse data, the central shared artifact of SQAC is not as specific as a text or a table. Instead, a generic concept is used, which is referred to as DS in the following.

DSs are the basic unit of data presented to a user. Multiple DSs can exists at the same time. A DS is a collection of arbitrary values. For instance, it could contain the details of a commit, consisting of the commit message, time, author and list of changed files. It could also contain the number of Sonar issues per severity at a specific day. A DS is self-contained, i.e., it contains all the information that is necessary to understand and process it: its values, the semantic of its values and the information how it can be rendered. Figure 3.2 illustrates this idea.

It is not defined in which format the actual values of a DS are stored. Nevertheless, specific ways of rendering are connected to specific data formats. For example, in order to render a pie chart, a set of key-value mappings are needed. Each DS must announce in which way it can be rendered. Examples for ways of rendering are pie charts, bar charts or tables. Different DSs can be rendered in different ways. Even the same DS could be displayed differently. Figure 3.2 illustrates an DS which can be rendered as pie chart and as table. The semantic information of the DS values are based on ontologies. A central benefit of the usage of ontologies is the possibility to describe data unambiguously. The data model of SQAC (see Section 3.2.3) is based on the SEON ontology that has been described in Section 2.2.

The design of the DS is motivated by different aspects. Self-contained DSs can be easily exported and transferred. Other applications could process them, as long as they support the way of rendering and the used ontology. SQAC is a web application, but due to the self-contained DS, offline data viewer could be supported. The semantic information enables a variety of further useful functions. For example, relations between DSs can be identified or matching operations can be recommended. Since a DS can support multiple ways of rendering, different views on the same data are enabled (see Section 3.1.4).

DSs are a flexible unit of data. They are not constraining the actual data granularity. A DS could represent a complex graph as well as a single word. However, DS are conceived as a collection of related or aggregated data. A rendered DS should be a single independently understandable document. It must be comprehensible by human users. In most cases, one single value - like the date of a commit - will not ensure this. The comprehensibility is especially important in a collaborative work context, where multiple users must interact with the same data. Furthermore, it supports the result transfer addressed by use case 2 in Section 1.1. This topic will be discussed in detail in Section 3.1.3.



Severities :	**Issues**
Major	: 3492
Minor	: 59
Info	: 1071

Severity	Issues
MAJOR	3492
MINOR	59
INFO	1071

Figure 3.2: **The concept of data sets.** This data set can be rendered as pie chart and as table. Each data set contains the semantic of its values. A superior ontology defines the semantic.

3.1.2 Data Integration and Queries

DSs are created by queries that can be executed on the underlying DST. The DST integrates the data of multiple repositories. This integration happens through a semantic model and the usage of ontologies (see Section 2.2). Currently, the repositories Jira, Github and Sonar are supported by SQAC. The SEON ontology provides the necessary constructs to describe Github data and Jira data. SQAC extends SEON in order to enable Sonar support.

During the import, the data of the different repositories are linked with each other. The data linking is shown in Figure 3.3. The link between the Github data and the Jira data is create via the Jira issue key. It is best practice to integrate this key in the commit message. The commit message *CAMEL-7145 Added support for basic authentication* refers to a Jira issue with the key CAMEL-7145. Sonar data is linked with Github data by a component key. Sonar is using the term component for artifacts like projects, class files or modules. A Github commit contains the list of changed files. By means of the file names, the corresponding component can be identified and the link to the Sonar data can be created. Jira data is linked with Sonar data via the project key. Each Jira issue contains such a key. A project is represented as a component in Sonar. All three kinds of data are linked via the authors. A commit has a committer, a Jira issue is created or modified by a user and a Sonar issue can be commented by a user. In all three cases, the author can be identified via its email address.

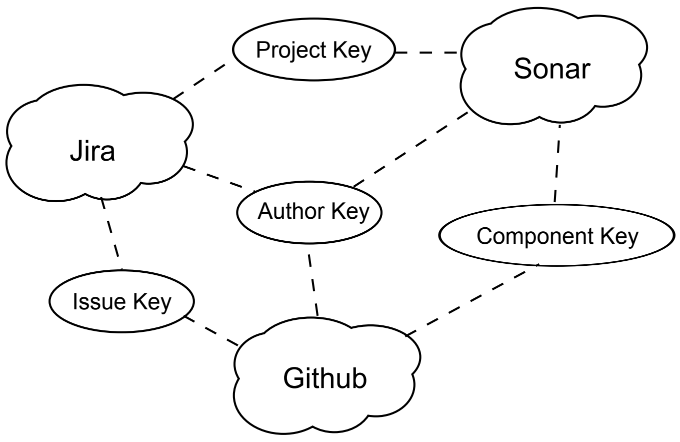


Figure 3.3: **The links between the different categories of data.** The data imported from Jira, Github and Sonar are linked in the semantic model.

An SQAC query is of a higher abstraction than a single SQL query or SPARQL query. However, SPARQL queries are used by SQAC queries to access the underlying semantic model. A query in the context of SQAC is a collection of SPARQL queries and further operations. It represents some arbitrary complex data analysis operation. Basically, one SQAC query could create the information for a complete dashboard showing all kinds of information to a, e.g., Jira project.

Each executed query creates a single complete DS. DSs and queries are tightly related, since two queries will not created the same kind of DS. A query may support a number of optional or mandatory parameters. These parameters must be described by the query. Besides the names of the parameters, it provides also their semantics. Similar to the semantic of DS values, the semantic of query parameters is based on central ontologies. The parameters that have been used for the execution of a query will be stored in the resulting DS. This information allows to reconstruct how a DS was created. Each query has a unique name and a human readable description.

The actual data analysis operations are an important part of an SQR. Queries represent these operations and can be regarded as individual entities. The assumption is, that the used query concept helps to design useful analysis operations of a sensible granularity. The implementation is eased by a clear encapsulation (see Section 4.2.5). The design is also influenced by the general concept of functional programming. As is a function, a query is an atomic and side-effect free operation. The conceptional and technical loose coupling between a query and all other entities allows for the support of different combinations of queries in different situations (see Section 3.1.5).

The semantic information of the query parameters can be used to recommend queries to the user. This can be based on the values of a DS. It is possible, since each of the values itself have also an unambiguously described semantic. The recommendation can be based on the available values. Hence, a query that can handle a specific value combination can be recommended. This idea is shown in Figure 3.4. A more advanced approach is discussed in Section 6.2.6.

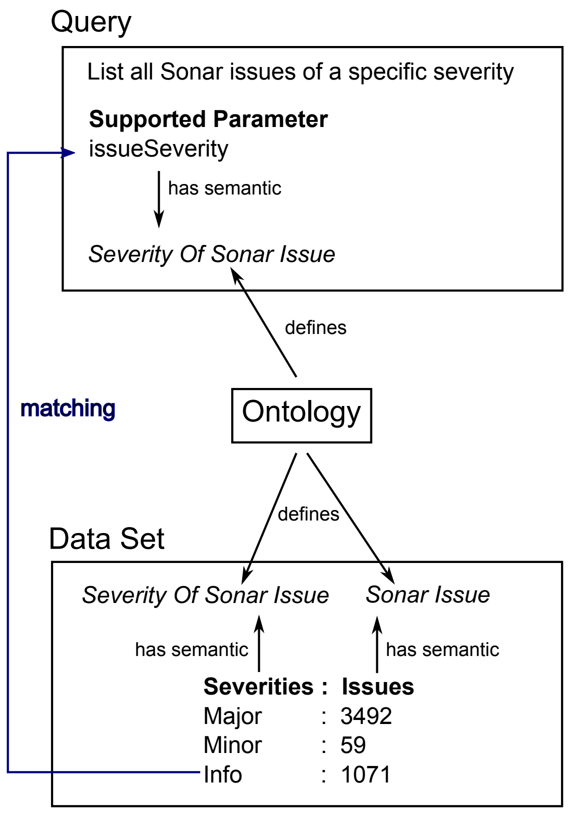


Figure 3.4: **Query recommendation support by semantic information.** Since the semantic of the supported query parameters is defined, queries can be recommended based on the values of data sets. In the example, the query *List all sonar issues of the severity Info* can be recommended for a user that has clicked on the value *Info*.

3.1.3 Spaces

Users that want to collaborate via SQAC have to join the same space. In analogy to the shared text editor example used before, the space represents the editor itself and the opened text document. The adding of text to the shared text documents corresponds to the adding of DSs to the shared space. Each DS is associated with exactly one space and a space can contain multiple DSs. In order to create DSs, a space allows for the execution of queries by users. As described in the last Section, a query will create a DS. It will be added to the space. The Figure 3.1 shows an example of one space that contains one DS.

A space itself is not bound to a specific way of representation. How the space and the DSs on the space are displayed to the user is defined by SVs. In the terms of the Model-View-Controller (MVC) pattern (see [GHJV94]), the space is the model and the SV a mixture of view and controller. SVs are discussed in the next Section. One can imagine an SQAC space as an open working area - like a whiteboard - on which multiple boxes can be rendered. Each of these boxes represents one DS. This corresponds to the actual implementation of the space visualization currently used by SQAC (see Section 3.2.2).

By the usage of spaces, user can establish a shared view on DSs. They can discuss the DSs and execute queries in order to investigate further data. Multiple spaces can exist at the same time. Each user can only be connected to one space at a time. Users can leave a space and come back later. The space will not vanish if no user is connected to it anymore. It will stay in the same state.

Beside the execution of queries, a space supports the removal of DSs and the modification of DS properties. Properties are not part of the DS contents that have been described in Section 3.1.1. They are not added by a query. A new DS has no properties at all. However, arbitrary properties can be added to a DS, after it was added to a space. They can be changed and removed. The position of a DS on the working area is an example for such properties. A comment that has been added to a DS by a user would also become a property of the DS. DS can be transferred between spaces. The target space will receive a full copy of the DS. After the transfer has been executed, no connection between the two DS exists anymore. Transferred DS keep their properties.

The basic space concept is very flexible. No specific way of usage is defined, but multiple ways are supported. The concept supports the use cases that have been defined in Section 1.1. Use case 1 did address the collaborative exploration of data via a shared view. With spaces, user can establish a shared view but will still be able to explore data on their own. They have different possibilities for doing that. For instance, they can simply work on the same space all the time. Alternatively, each participant could create his own space. The participants could use one of the spaces to discuss results. If individual data exploration is needed, users can switch to their private spaces. Results of this individual work can be transferred to the shared space and discussed. Figure 3.5 shows an approach based on the utilization of multiple spaces.

The state of a space after an SQR can be seen as the result of the review. This covers the requirements entailed from use case 2 in Section 1.1. The creation of the review result is eased for the reviewers. They create the result with the same tool, that is used for the data exploration and collaboration. As described in Section 3.1.1, rendered DS should be understandable units of data. They can be arranged and commented. No additional document is needed. The recipients of the results can just join the result space. The understanding of the data can be supported by means of the semantic information provided by the DSs. These can be used to identify and highlight relations. If further information is needed, the recipients can just continue working with the DSs on the space.

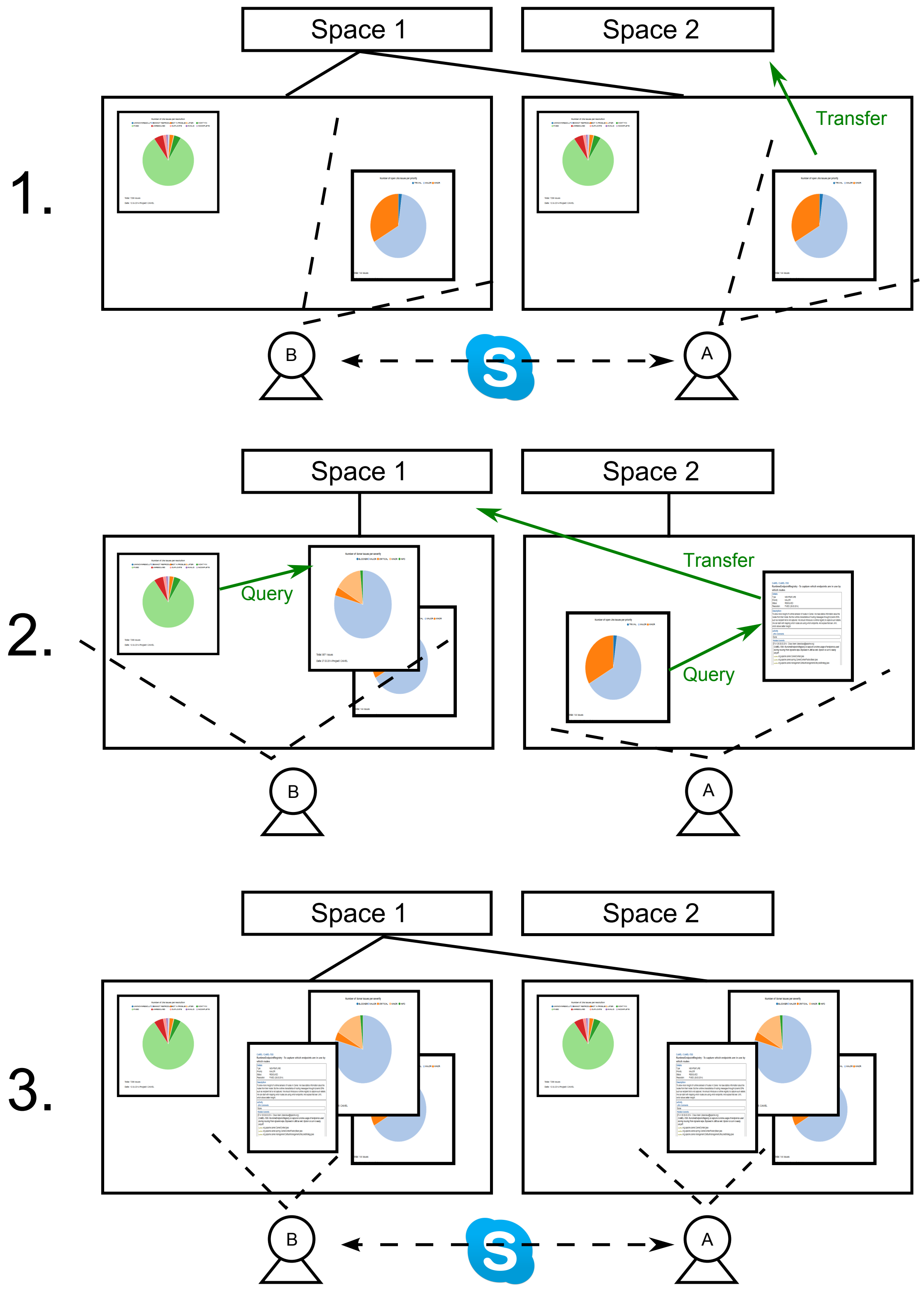


Figure 3.5: **Example for space usage.** 1.) Alice (A) and Bob (B) have established a shared view on a data set. They are talking about it via Skype. Alice is not sure about it and she needs more information. She transfers the data set to space 2 in order to perform further investigations. 2.) Alice joins space 2. In the mean time, Bob is analyzing another data set. He is individually executing a query that creates a new data set. Alice is doing the same. After she found the necessary data set, she transfers it back to space 1. 3.) Alice joins space 1 and explains the transferred data set to Bob.

3.1.4 Space Views and Space View Pages

In order to make the space approach more flexible, users are not interacting with the space directly, but by means of SVs. The relation between SV and space can be compared to the relation between the model and the view in the MVC pattern. Though, a SV combines view and controller aspects. SVs are responsible for rendering the space and the data of the space. They provide interaction functionalities for the connected users. One space can support multiple SVs. The Figure 3.1 shows a space with two SVs.

An SV has a frontend and a backend representation. In the frontend, it is represented by a SVP. This is an HTML page that is rendered for the user. In the backend, the SV is a container for information that is needed to synchronize the SVPs of different users. The basic operations that have been described in Section 3.1.3 - like the execution of queries - are provided and handled by the space. Operations on the SVP are mapped onto this basic operations. For example, the change of the position of a rendered DS would be mapped to the change of DS properties.

The support of SVs increases the flexibility, since data can be rendered in different ways and different operations can be supported. For example, role based SVs could provide optimized views on the same space data for different types of stakeholders. Also, views that have been optimized for specific devices could be provided. All users connected to one SV establish a shared view. Obviously, this is not necessarily true for users that are connected to the same space via different SVs. It is a topic of future research, if collaboration between users with different views is beneficial. In Figure 3.1 user 1 and user 2 are connected to the space via the same SV. Hence, their SVP is identical and synchronized. The user 3 is connected via an alternative SV. The existing DS is presented to him in a different way.

3.1.5 Space Types and Space View Types

Spaces and SVs have types. A space is actually an instance of a specific *Space Type* (ST) and an SV an instance of a specific *Space View Type* (SVT). An SVT contains rules that define which kinds of users will be assigned to a SV of this type. These rules will be used if a user joins a space that supports multiple SVs. The decision could be based on the role of the user or the kind of device he is using. Furthermore, the SVT defines which SVP will be rendered for the connected users. The ST defines which SVTs are supported by a space. Multiple STs can support the same SVTs. Multiple instances of STs and SVTs can exist. The relation between spaces, STs, SVs and SVTs is shown in Figure 3.6.

The types increase the flexibility by supporting multiple configurations of spaces. They enable the future extensions of the application concept. For example, an ST could be used to establish multiple restrictions. It could define what types of queries can be executed. STs and SVTs support studies in the field of SQRs. With future versions of the application, one could define multiple STs that apply different rules and support different SVTs. The results of reviews performed with these STs could be compared. This topic will be discussed in more detail in the future work section of Chapter 6.

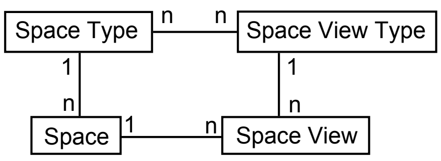


Figure 3.6: **Entity relationship model of space, space view and their types.** Each space has one space type. Each space view has one space view type. A space type can support multiple space view types. A space can support multiple space views.

3.1.6 Space Events

The interaction between users and spaces happens by means of events. A user triggers events by her actions on a SVP. Each action is mapped to a specific kind of event. The SVP defines which user actions will trigger events. Each event is related to a specific operation on the space. For instance, a query event is triggered in order to execute a query. A history of all events will be stored by each space. The SVs of all users will be informed about the executed events and can adapt accordingly.

The event based concept has different benefits. The event history can be used to reconstruct old space states. Hence, the steps undertaken on the space can be replayed or be undone later. This is useful for a user, that was not able to attend to an SQR and it supports the transfer of review result. Users can replay the operations performed in an SQR, in order to comprehend the review. The event history can also be used to analyze the actions performed on a space, as described by use case 3 in Section 1.1. This can support the understanding and optimization of review processes.

The general application concept described in this section is actually not specific for collaborative SQRs. It can be seen as a generic approach for the collaborative analyses of data. SQRs are just a specific area of application. Generally, it would be possible to collaboratively review business results, patients histories or other data with the same approach. The only thing that would change are the data repositories that are integrated and the queries that are used.

3.2 Application Design

SQAC is a web application which can be accessed by a regular browser. It is based on the general concept described in the last Section. It allows users to create multiple spaces and join them. If a user joins a space, he will be assigned to a SV and the corresponding SVP is rendered. Currently, one ST and one SVT have been implemented. They are referred to as *Default Space Type* (DSTP) and *Default Space View Type* (DSVT), respectively. All created spaces will be of the DSTP and support one SV of the DSVT. One SVP exists. It is associated with the DSVT and will be referenced as the *Default Space View Page* (DSVP).

The DSVP allows for the execution of queries and the interaction with rendered DSs. The page consists of two main areas: A nested control panel on the top and a working area below this panel (see Figure 3.7). The control panel is described in Section 3.2.1 and the working area in Section 3.2.2. The different queries that are provided by SQAC are described in Section 3.2.3.

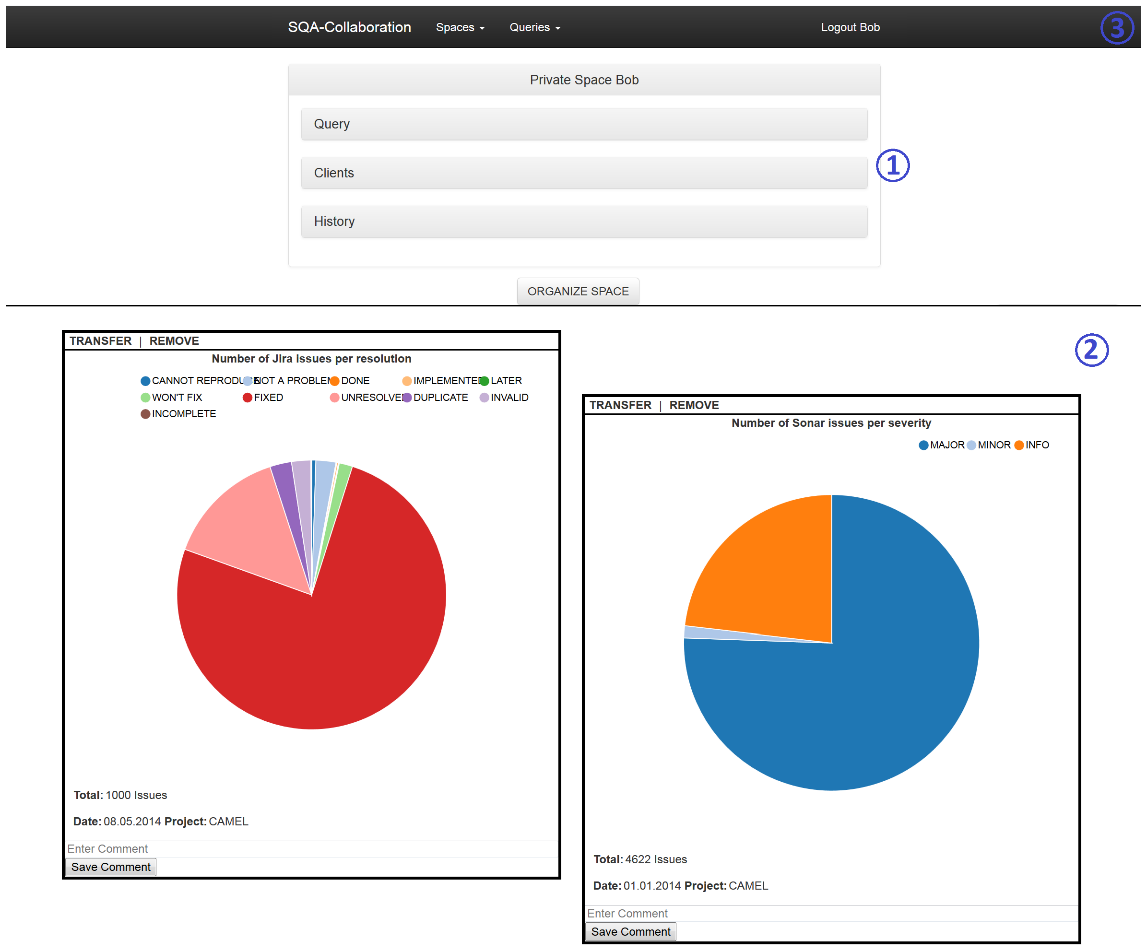
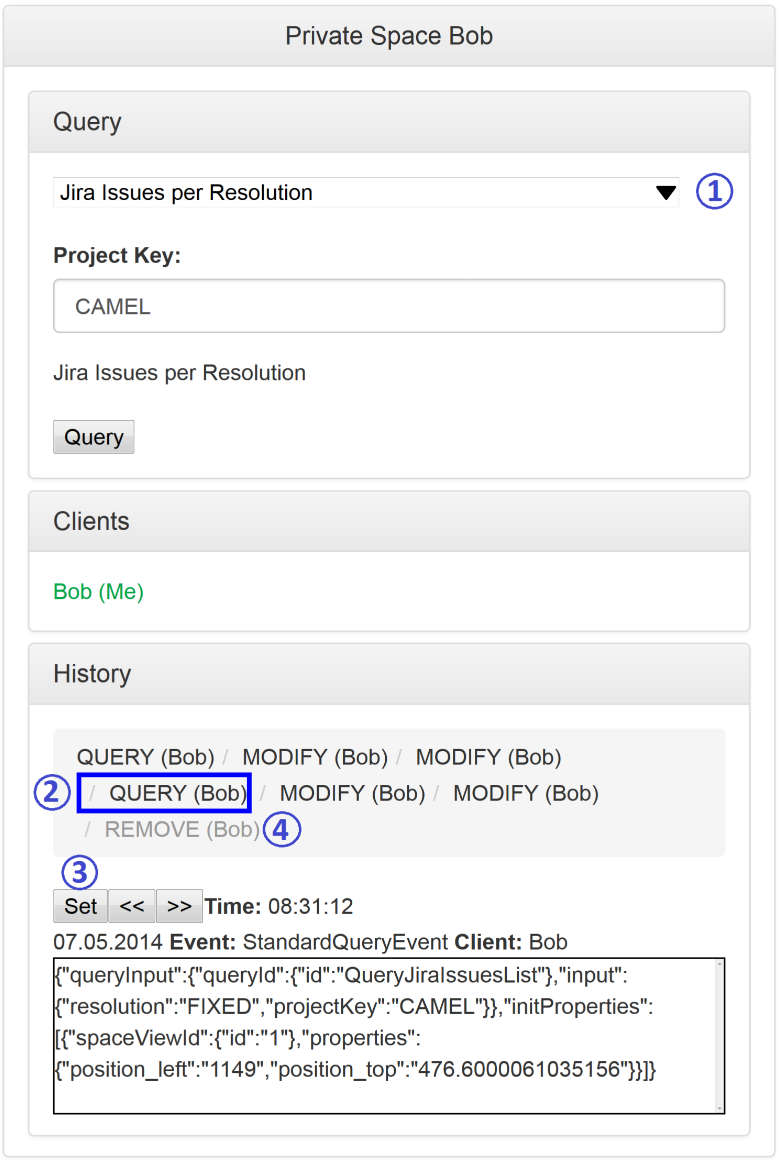


Figure 3.7: **The complete default space view page.** The page consist of a nested control panel (1) on the top and a working area (2) below it. The control panel and its sub-panels can be collapsed. In the example, the sub-panels are collapsed. The working area can be compared to a whiteboard. Two data sets are shown on the working area. Only a part of the working area can be seen. The menu bar (3) on the very top is not part of the space view page. The menu bar can be used to create, join, export and import spaces. Furthermore, a description of all queries can be accessed.

3.2.1 Control Panel

A control panel is shown in Figure 3.8. It contains three sections that provide different information and functions.

A drop-down list with available queries is provided by the panel *Query*. If one of them is selected, an input form will be rendered. The form can be used to enter parameters and execute the query. After a query has been executed, the resulting DS will appear in the working area. If the query could not be executed or did not produce a result, a message will be displayed at the top of the page.


Private Space Bob
Query
Jira Issues per Resolution
1
Project Key:
CAMEL
Jira Issues per Resolution
Query
Clients
Bob (Me)
History
QUERY (Bob) / MODIFY (Bob) / MODIFY (Bob)
2
/ QUERY (Bob) / MODIFY (Bob) / MODIFY (Bob)
/ REMOVE (Bob)
4
3
Set
<<
>>
Time: 08:31:12

07.05.2014 **Event:** StandardQueryEvent **Client:** Bob

```
{"queryInput":{"queryId":{"id":"QueryJiraIssuesList"},"input":
{"resolution":"FIXED","projectKey":"CAMEL"}},"initProperties":
[{"spaceViewId":{"id":"1"},"properties":
{"position_left":"1149","position_top":"476.6000061035156"}}]}
```

Figure 3.8: **The control panel of the default space view.** The query panel contains a drop-down list of available queries (1). A form for the selected query has been rendered. The clients panel is showing that currently only one user is connected to the space. In the history panel a previously executed query event has been selected (2). The space could be temporarily reset to the selected state by clicking *Set* (3). The currently initialized state is drawn in grey (4). The latest state is on the very right end of the event list. In the example, it is *REMOVE (Bob)*.

All users who are currently connected to the space are listed in the panel *Clients*. The current user itself is marked by the post-fix *Me*. Currently, the panel simply list the names of all users in order to support the group awareness. Other functions - like the initialization of a chat - are not provided, yet.

The panel *History* contains the event history of the space. The latest event is at the very right end of the history. In Figure 3.8 it is *REMOVE (Bob)*. Each of the events can be selected via a mouse click. The selected event will be highlighted by a blue frame. The selection will show the details of the event and a button named *Set*. The details consist of the event type, the event time, the responsible user and the message that has been sent to the server. In Figure 3.8 a query event that has been executed by Bob is selected. By clicking on the *Set*-button, the space can be temporarily reset to the selected state. The currently initialized state is drawn in grey. A state change will block the space for all actions. Only further state changes will be still possible. The blocking is necessary in order to ensure the consistency of the space. The space will be blocked until it is returned to the latest state. A blocked space is indicated by a red framed working area and a corresponding message above it. State changes have only an effect on the current space. Therefore, DS transfers will not be undone by it. The history function should be of high value for users, who want to understand the sequence of steps by which DSs have been created. A person not able to attend a quality review could just join the space and step through the complete review. The latter is eased via the buttons » (next step) and « (previous step) which are located next to the *Set* button.

3.2.2 Working Area

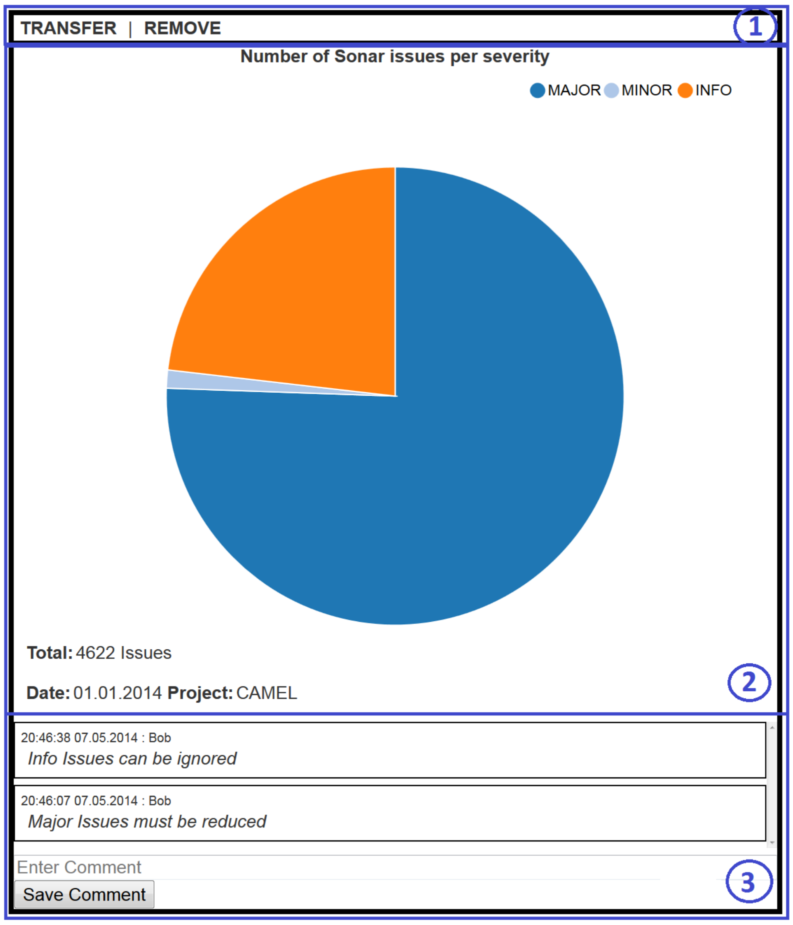
The working area is the main area of interaction on the page. It is indicated with the number 2 in Figure 3.7. Currently, the working area has a dimension of 2500 x 2500 pixels in order to provide enough space for multiple users. Therefore, not the complete working area can be seen at once on normal sized screens. A user can utilize the scroll or the zoom function of the browser to change his point of view.

DSs are displayed on the working area. They are rendered inside a container box that consists of a menu bar on the top, a content area in the middle and a comment area on the bottom. Figure 3.9 shows a DS container with blue framed areas.

The menu bar of a DS container provides two operations. *Remove* will remove the DS from the space permanently. If *Transfer* has been selected, a context menu will list other available spaces. A click on one of the list elements will transfer the DS to the corresponding space. The transfer will not remove the DS form the current space. The target space will receive a full copy of it. A DS can be moved inside the boundaries of the working area by means of drag and drop. In order to do so, the menu bar must be selected. Currently, a view synchronization of the DS position will only be performed after the drop action. This improves the application performance via a reduction of messages and page re-rendering. Furthermore, a user is not permanently distracted by DSs that are moved by other users. A drawback is the reduced awareness (see Section 2.3) of other user's actions. The latter topic is discussed in Section 3.3.

The actual data of the DS is shown in the content area. It will contain a chart, a table or another form of visualized data. How the data is visualized depends on the way of rendering that is supported by the corresponding DS.

The comment area allows users to add comments to a DS. They can enter a message in the input field and click *Save Comment* or press *Enter*. Comments will be added to the DS. They will be directly visible for all other users. If a DS is transferred to another space, its comments will be transferred, too.



Total: 4622 Issues

Date: 01.01.2014 Project: CAMEL

20:46:38 07.05.2014 : Bob
Info Issues can be ignored

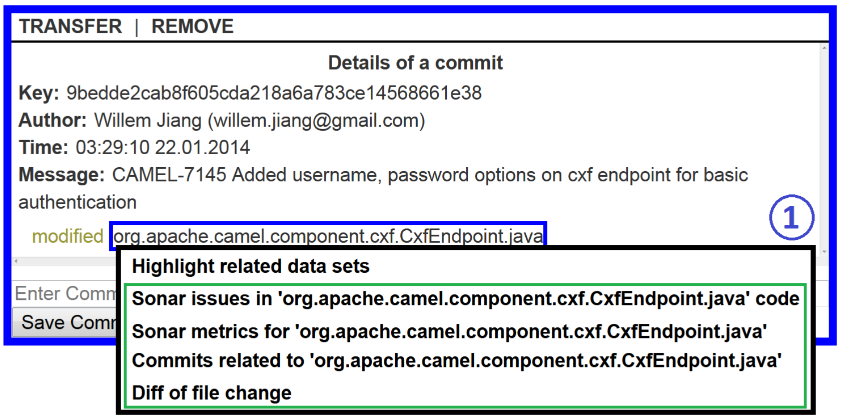
20:46:07 07.05.2014 : Bob
Major Issues must be reduced

Enter Comment

Figure 3.9: **A data set container with framed areas.** The structure of each data set container is as follows: A menu bar on the top (1), a content area in the middle (2) and a comment area in the bottom (3).

The query panel is not the only tool that can be employed to execute queries. By clicking on a value of a rendered DS, a context menu is displayed that contains a list of recommended queries. In the following, this menu will be addressed as *Value Context Menu* (VCM). The recommendations are based on the selected value and its semantic. A recommended query can be executed by clicking on the corresponding list item. An example of a DS with rendered VCM is shown in Figure 3.10. In this example, the DS contains information about a specific commit. Part of this information is the file that has been changed by the commit. A click on the file name produces the VCM containing multiple recommendations and the option *Highlight related data sets*. The latter is explained in detail below. By clicking on, e.g., *Diff of file change*, the corresponding query is executed and the result DS appears on the working area. A user can now continue the exploration of the data by selecting values of the new DS and by executing recommended queries.

Depending on the number of users and the working process, the working area can be filled very quickly with many DSs. Since multiple DSs can overlap each other, it can be difficult to keep the overview or find a specific DS on the working area. In order to address this problem, a function for the organization of the DSs is provided. It can be executed by clicking on the button


TRANSFER | REMOVE

Details of a commit

Key: 9bedde2cab8f605cda218a6a783ce14568661e38
Author: Willem Jiang (willem.jiang@gmail.com)
Time: 03:29:10 22.01.2014
Message: CAMEL-7145 Added username, password options on cxf endpoint for basic authentication

modified org.apache.camel.component.cxf.CxfEndpoint.java

1
Enter Comm
Save Comm
Highlight related data sets
Sonar issues in 'org.apache.camel.component.cxf.CxfEndpoint.java' code
Sonar metrics for 'org.apache.camel.component.cxf.CxfEndpoint.java'
Commits related to 'org.apache.camel.component.cxf.CxfEndpoint.java'
Diff of file change

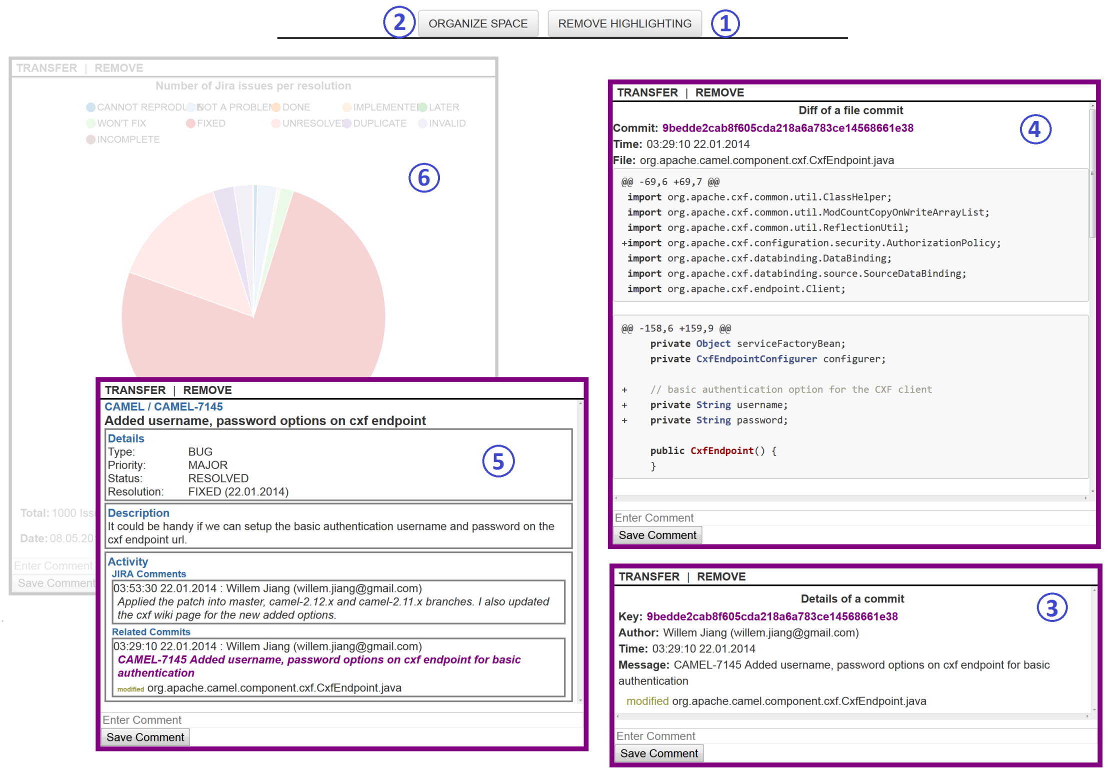
Figure 3.10: **A data set with context menu.** By clicking on the values of a data set, a context menu (1) is created. This contains the option to highlight related data sets and a list of query recommendations. In the example, the recommendations have been framed green. A selected DS and a selected value are highlighted by a blue frame on the working area.

Organize Space, that is positioned direct above the working area (see Figure 3.11). A click on the button will arrange all DSs in rows and thereby given an overview of all of them. Since the working area will be organized for all users, this operation should only be executed in agreement with them. The function is particularly useful, if the users want to discuss all DSs one by one. They can organized the space and handle the DSs row by row.

It is possible to highlight DSs that are containing values with a specific semantic. The VCM contains the option *Highlight related data sets*. This function will highlight all DSs containing values with the same semantic as the selected value. Figure 3.11 shows the highlighting of DSs that are related to a specific commit. The related DSs are marked with a purple border and put on top. The relevant values are highlighted in purple, too. All DSs that are not related with respect to the initially selected value are made transparent and put in the background. It is still possible to use all other functions and interact with all DSs, when the highlighting is activated. If a new DS is created, it will also be affected by the highlighting. To deactivate the highlighting again, the button *Remove Highlighting* must be clicked. The button will appear next to *Organize Space* when the highlighting is activated (see Figure 3.11). Only one value at a time can be highlighted and the pages of all users will be affected by the operation.

The possibility to highlight related DSs should be especially useful if a space contains the result of an SQR, as described in use case 2 of Section 1.1. The persons who are supposed to work with the results can identify the relations between the displayed DSs. If the organize space operation is executed while the highlighting is activated, all unrelated DSs will be stacked in the back and only the related DSs will be organized. Hence, the two functions can be combined in order to filter DS from the working area temporarily.

The DSVP is based on the loose coupled view paradigm, that has been described in Section 2.3. The state of the working area and the DSs will be synchronized, since these are the shared objects. Users can select which part of the working area they want to see, individually. If another user changed a DS by, e.g., moving it or adding a comment, the corresponding DS will be put on top and highlighted green for some seconds. This shall help to produce awareness of other user's



TRANSFER | REMOVE

Diff of a file commit

Commit: 9bedde2cab8f605cda218a6a783ce14568661e38
Time: 03:29:10 22.01.2014
File: org.apache.camel.component.cxf.CxfEndpoint.java

```
@@ -69,6 +69,7 @@
 import org.apache.cxf.common.util.ClassHelper;
 import org.apache.cxf.common.util.ModCountCopyOnWriteArrayList;
 import org.apache.cxf.common.util.ReflectionUtil;
+import org.apache.cxf.configuration.security.AuthorizationPolicy;
 import org.apache.cxf.databinding.DataBinding;
 import org.apache.cxf.databinding.source.SourceDataBinding;
 import org.apache.cxf.endpoint.Client;
```

```
@@ -158,6 +159,9 @@
     private Object serviceFactoryBean;
     private CxfEndpointConfigurer configurer;

+    // basic authentication option for the CXF client
+    private String username;
+    private String password;

     public CxfEndpoint() {
     }
```

Enter Comment
Save Comment

TRANSFER | REMOVE

CAMEL / CAMEL-7145

Added username, password options on cxf endpoint

Details

Type:	BUG
Priority:	MAJOR
Status:	RESOLVED
Resolution:	FIXED (22.01.2014)

Description

It could be handy if we can setup the basic authentication username and password on the cxf endpoint url.

Activity

JIRA Comments

03:53:30 22.01.2014 : Willem Jiang (willem.jiang@gmail.com)
Applied the patch into master, camel-2.12.x and camel-2.11.x branches. I also updated the cxf wiki page for the new added options.

Related Commits

03:29:10 22.01.2014 : Willem Jiang (willem.jiang@gmail.com)
CAMEL-7145 Added username, password options on cxf endpoint for basic authentication
modified org.apache.camel.component.cxf.CxfEndpoint.java

Enter Comment
Save Comment

TRANSFER | REMOVE

Details of a commit

Key: 9bedde2cab8f605cda218a6a783ce14568661e38
Author: Willem Jiang (willem.jiang@gmail.com)
Time: 03:29:10 22.01.2014
Message: CAMEL-7145 Added username, password options on cxf endpoint for basic authentication

modified org.apache.camel.component.cxf.CxfEndpoint.java

Enter Comment
Save Comment

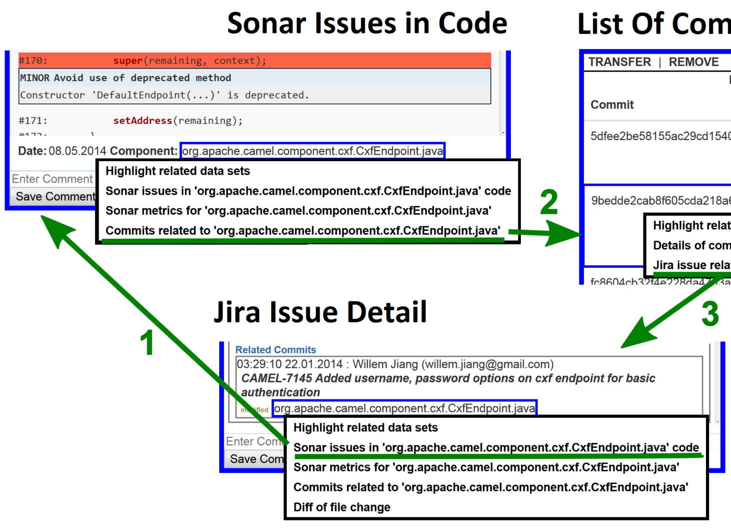
Figure 3.11: **A working area with highlighted data sets.** Four data sets are display on the working area. All data sets related to a specific commit have been highlighted: The details of a commit (3), the diff of a file change (4) and the details of a Jira issue (5). One data set is not related to the commit (6). Therefore, it has been made transparent and placed in the background. The button *Remove Highlighting* (1) will deactivate the highlighting. The button *Organize Space* (2) will organize the data sets on the space.

actions. Displayed context menus and interactions with the control panel will not be mirrored on the pages of other users. The assumption is, that too many information about the actions of other users would only be irritating. Solely actions that change the state of the space, the SV or a DS will produce optical feedback for other users. Nevertheless, the current support of awareness, coordination, collaboration and communication has limitations. They are discussed in Section 3.3.

3.2.3 Queries and Data Set Rendering

SQAC currently supports Github, Jira and Sonar as data repositories. A number of queries have been implemented to analyze the data of those repositories. The queries can be executed by means of the control panel (see Figure 3.8) or via the VCM (see Figure 3.10). Each query is related to one of the repositories. As described in Section 3.2.3, the data of the repositories are linked. By means of the recommended queries, related data of other repositories can be accessed. A review of technical aspects of SQ is enabled with the supported repositories and queries. As discussed in Section 6.2.1, future version of the application need to support more repositories. With the current queries, the usage of SQAC resembles the usage of a browser. Each DS can be seen as a hypertext document showing pieces of the data of one repository. The executing of a recommended query can be regarded as the following of a link. The available queries allow to traverse the data. An example for the traversal of the different types of data is shown in Figure 3.12. Future versions of SQAC could provide more queries that aggregate data of multiple repositories in single DSs. Fine grained DSs, e.g., one showing the diff of one specific file change, support more fine grained review results. I.e., one can select and show only the relevant piece of data. However, due to the recommended queries, DSs with more aggregated data could easily be supplemented by more specific DSs.

In the following, a description of each query is given. Additionally, resulting DSs are shown. SQAC provides a generic rendering format for tables, pie charts and time series charts. For each of these types of rendering, only one example will be given. One can click the sections of pie charts (see Figure 3.9), the data points of time series charts (see Figure 3.16) and the rows of tables (see Figure 3.13) to show the VCM. The two chart types include a legend. The legend elements can be clicked in order to hide the corresponding data. Each table will contain a maximum of 150 rows. For all the generic rendering formats is true: The used query parameters will be displayed at the bottom of the DS. They can clicked just as other DS values. For example, a date, a project and a severity has been used as query parameter when the DS in Figure 3.13 was created.


Sonar Issues in Code
List Of Commits
Jira Issue Detail
1
2
3
Enter Comment
Save Comment
TRANSFER | REMOVE
Highlight related data sets
Sonar issues in 'org.apache.camel.component.cxf.CxfEndpoint.java' code
Sonar metrics for 'org.apache.camel.component.cxf.CxfEndpoint.java'
Commits related to 'org.apache.camel.component.cxf.CxfEndpoint.java'
Details of commit
Jira issue related to commit
Diff of file change
Related Commits
modified

```
#170:           super(remaining, context);
MINOR Avoid use of deprecated method
Constructor 'DefaultEndpoint(...)' is deprecated.

#171:           setAddress(remaining);
```

Date: 08.05.2014 **Component:** org.apache.camel.component.cxf.CxfEndpoint.java

List of commits related to component

Commit	Time	Message
5dfee2be58155ac29cd1540cf9d7d3b10406216e	12:38:42 13.02.2014	CAMEL-6783 Updated the Features get/set method with the change of CXF
9bedde2cab8f605cda218a6a783ce14568661e38	03:29:10 22.01.2014	CAMEL-7145 Added username, password options on cxf endpoint for basic authentication

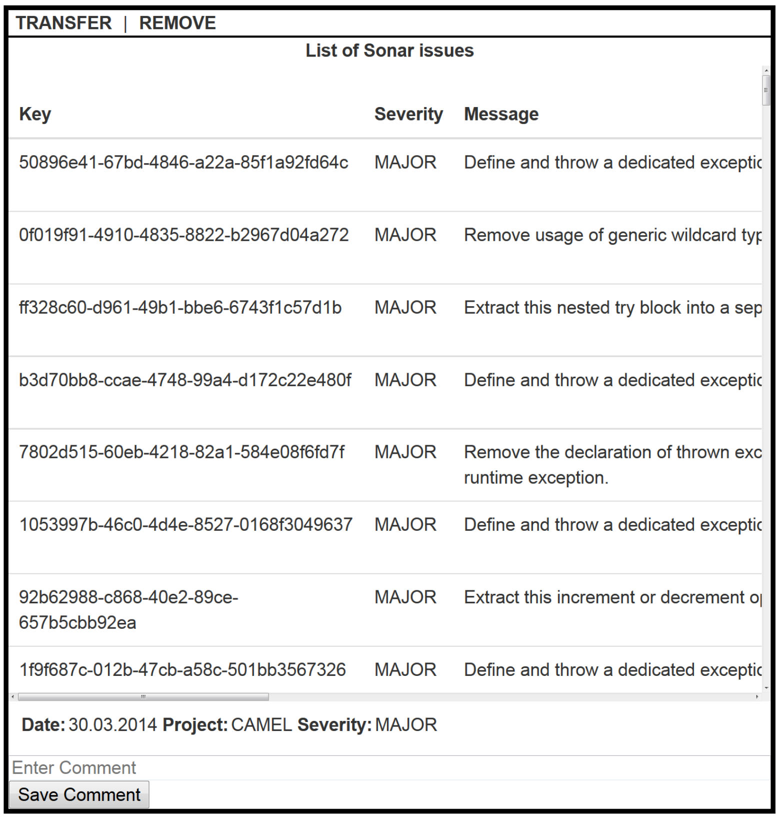
03:29:10 22.01.2014 : Willem Jiang (willem.jiang@gmail.com)
CAMEL-7145 Added username, password options on cxf endpoint for basic authentication
org.apache.camel.component.cxf.CxfEndpoint.java

Figure 3.12: **Example of data traversal via queries.** Queries: 1.) Code of Component with marked Sonar issues 2.) Commits related to a specific component 3.) Details of Jira issue.

Sonar Queries

The Sonar data provides a wide range of code quality information. The code quality is an important part of the technical aspect of SQ. The provided queries allow to access Sonar data for projects and single components. Sonar data can be the starting point of an SQR. For example: From an overview of Sonar issues in a project a user can traverse to details of the issues. The causes of the issues can be identified by means of the Github queries. The rationals of the changes can be identified with the Jira queries.

Sonar issues in a project. A table containing the key, the description, the severity and the creation time of Sonar issues is rendered as result of this query. The key of a project must be provided as mandatory parameter. The query accepts a severity in order to solely list issues of this severity. Furthermore, the creation date can be passed as parameter. The result table is shown in Figure 3.13.



List of Sonar issues

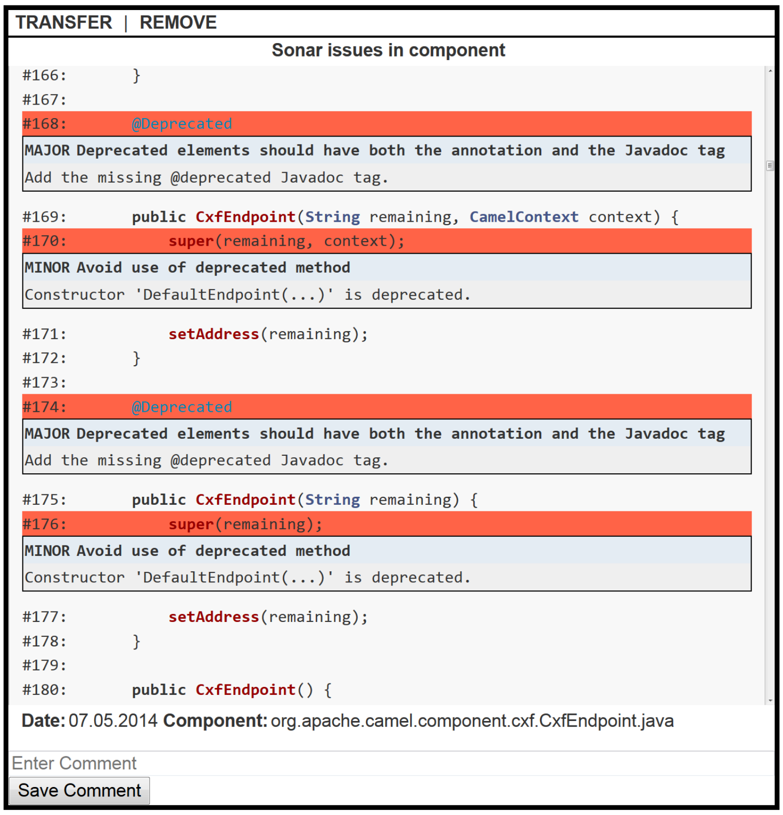
Key	Severity	Message
50896e41-67bd-4846-a22a-85f1a92fd64c	MAJOR	Define and throw a dedicated exceptic
0f019f91-4910-4835-8822-b2967d04a272	MAJOR	Remove usage of generic wildcard typ
ff328c60-d961-49b1-bbe6-6743f1c57d1b	MAJOR	Extract this nested try block into a sep
b3d70bb8-ccae-4748-99a4-d172c22e480f	MAJOR	Define and throw a dedicated exceptic
7802d515-60eb-4218-82a1-584e08f6fd7f	MAJOR	Remove the declaration of thrown exc runtime exception.
1053997b-46c0-4d4e-8527-0168f3049637	MAJOR	Define and throw a dedicated exceptic
92b62988-c868-40e2-89ce-657b5cbb92ea	MAJOR	Extract this increment or decrement o
1f9f687c-012b-47cb-a58c-501bb3567326	MAJOR	Define and throw a dedicated exceptic

Date: 30.03.2014 **Project:** CAMEL **Severity:** MAJOR

Figure 3.13: **Data set showing Sonar issues in a project.** The table contains one issue per row. Issues with the severity MAJOR in the project CAMEL and with a creation date before the 30.03.2014 are listed. Each row, the project and the severity can be clicked, in order to create the value context menu.

Code of a component with marked Sonar issues. This query requires the full name of a component, e.g., org.apache.camel.DefaultContext.java, as parameter. As result, the code of the component will be rendered. In the code, Sonar issues will be marked. The severity, the name and a short description of the issues is shown. Figure 3.14 presents a corresponding DS.

Details of one Sonar issue. The key of a Sonar issues must be provided as parameter for this query. The project, the component as well as the issue's severity, name and description will be shown. Furthermore, the corresponding code snippet will be displayed. An example DS is shown in Figure 3.15. The style of rendering is based on the presentation of single issues in Sonar.



```
        }

        @Deprecated
MAJOR Deprecated elements should have both the annotation and the Javadoc tag
Add the missing @deprecated Javadoc tag.

        public CxfEndpoint(String remaining, CamelContext context) {
            super(remaining, context);
MINOR Avoid use of deprecated method
Constructor 'DefaultEndpoint(...)' is deprecated.

            setAddress(remaining);
        }

        @Deprecated
MAJOR Deprecated elements should have both the annotation and the Javadoc tag
Add the missing @deprecated Javadoc tag.

        public CxfEndpoint(String remaining) {
            super(remaining);
MINOR Avoid use of deprecated method
Constructor 'DefaultEndpoint(...)' is deprecated.

            setAddress(remaining);
        }

        public CxfEndpoint() {
```

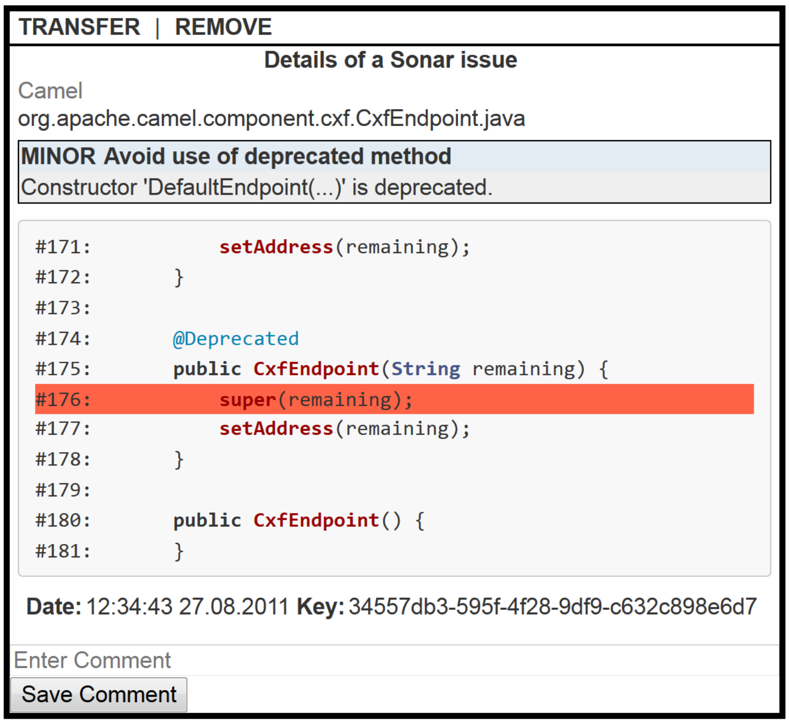
Date: 07.05.2014 **Component:** org.apache.camel.component.cxf.CxfEndpoint.java

Figure 3.14: **Data set showing code with marked Sonar issues.** Sonar issues are marked red in the code. The description of the issue is displayed below the marker. The issues and the component name can be clicked in order to display the value context menu.

Number of Sonar issues per severity. The query will display the number of Sonar issues per severity in a project. The project key must be passed as query parameter. The result will be rendered as pie chart. The sections represent the severities and the section size the number of corresponding issues. The Figure 3.9 shows an corresponding DS.

Process of the number of Sonar issues. The result of this query will be a time series chart, showing the process of the number of issues per severity. As shown in Figure 3.16, each severity is rendered as single data series. A date range and a project key must be passed as parameters.

Metrics of a component or project. A table of metrics for a component or for a project can be displayed by this query. Each row contains the metric key, the metric description and the actual metric value. Either a project key or the full name of a component must be passed as a parameter. The metrics are provided by Sonar and not calculated by SQAC. The most common metrics like number of classes, number of functions, non commenting lines of code, number of comment lines and cyclomatic complexity are provided.



TRANSFER | REMOVE

Details of a Sonar issue

Camel
org.apache.camel.component.cxf.CxfEndpoint.java

MINOR Avoid use of deprecated method
Constructor 'DefaultEndpoint(...)' is deprecated.

```
            setAddress(remaining);
        }

        @Deprecated
        public CxfEndpoint(String remaining) {
            super(remaining);
            setAddress(remaining);
        }

        public CxfEndpoint() {
        }
```

Date: 12:34:43 27.08.2011 **Key:** 34557db3-595f-4f28-9df9-c632c898e6d7

Figure 3.15: **Data set showing the details of a Sonar issue.** The project (CAMEL), the component name, the description of the issue, the corresponding code snippet and the issue key are displayed. All values can be clicked in order to display the value context menu.

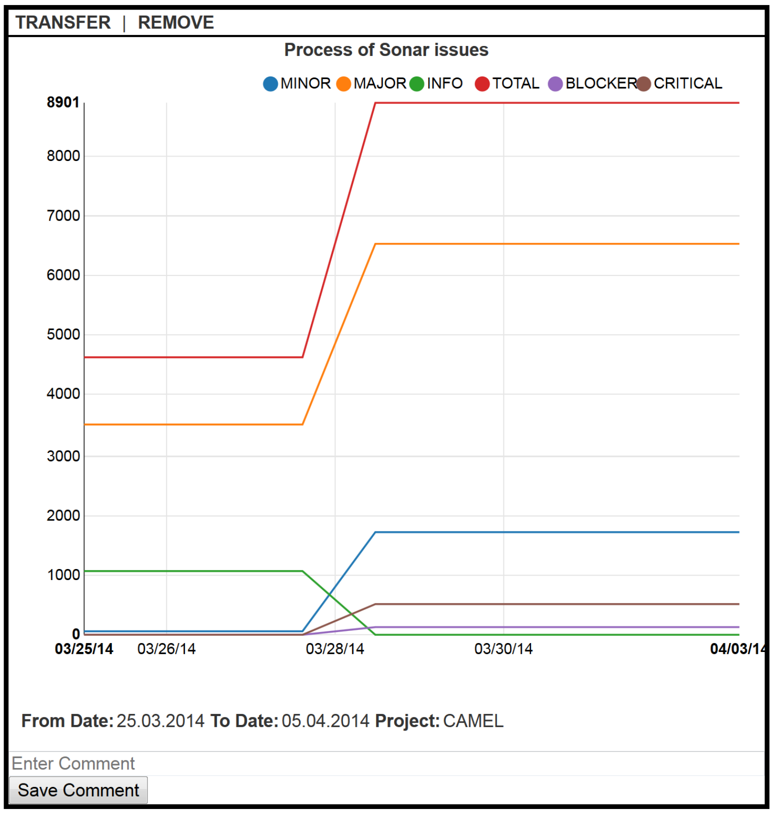
Jira Queries

The information in the issue tracking system is important for an SQR. If Jira is used well, it will contain important rationals of all changes made to the software. The number of existing issues can help to assess the general state of a project. A Jira issue can not only contain the rationals of the changes made, but also the commits that have been performed. Jira data can be the starting point of an SQR. For example: All resolved issues can be displayed and reviewed one by one. Alternatively, Jira data can be a source of rationals, if the SQR started with a Github data or Sonar data. In order to support the utilization of Jira data, four queries are provided.

Number of open Jira issues per priority. The number of open Jira issues per priority in a project will be displayed as the query result. The project key must be passed as a parameter. The results will be rendered as pie chart. Each section represents the number of open issues of one priority.

Number of Jira issues per resolution. This query needs a project key as parameter and will display the number of Jira issues per resolution. The resulting DS is rendered as pie chart.

Jira issues in a project. A table containing they key, title, priority, resolution, status and creation time of Jira issues will be displayed as result of this query. A project key must be passed as mandatory parameter. The results can be filtered by passing a resolution, a priority, a status or



From Date: 25.03.2014 **To Date:** 05.04.2014 **Project:** CAMEL

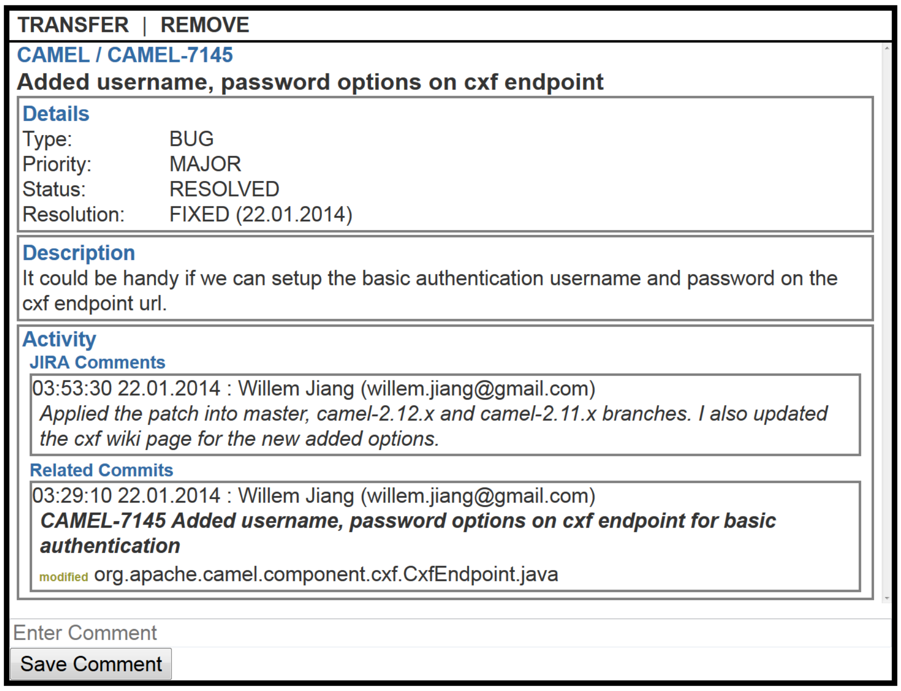
Figure 3.16: **Data set showing the process of Sonar issues.** The process of the number of Sonar issues in the project CAMEL between the 25.03.2014 and the 05.04.2014 is shown. Each severity is displayed in a single data series.

a creation date as parameter. Only Jira issues matching the given parameter combination will be selected by the query.

Details of a Jira issue. The details of a Jira issue can be displayed by means of this query. The issue can be identified via its key or the key of a related commit. Either of them has to be passed as a query parameter. An example for the details of a Jira issue can be seen in Figure 3.17. The style resembles the presentation of issues in Jira, since it is well known and structured. It consists of three main sections: Details, Description and Activity. The project, the key of the issue and the name of the issue are shown above these sections. The Details section contains the type, priority, status and resolution of the issue. The description is displayed in the Description section. The Activity section contains two sub-sections: JIRA Comments and Related Commits. The latter will list the details of all commits: Commit time, committer, commit message and affected files. Note that not every Jira issue contains comments or commits.

Github Queries

The changes made to a software project are reflected by commits. Commits can be an important link in an SQR. If changes in some quality aspects of a component are noticed, the commits can



CAMEL / CAMEL-7145

Added username, password options on cxf endpoint

Details

Type:	BUG
Priority:	MAJOR
Status:	RESOLVED
Resolution:	FIXED (22.01.2014)

Description

It could be handy if we can setup the basic authentication username and password on the cxf endpoint url.

Activity

JIRA Comments

03:53:30 22.01.2014 : Willem Jiang (willem.jiang@gmail.com)

Applied the patch into master, camel-2.12.x and camel-2.11.x branches. I also updated the cxf wiki page for the new added options.

Related Commits

03:29:10 22.01.2014 : Willem Jiang (willem.jiang@gmail.com)

CAMEL-7145 Added username, password options on cxf endpoint for basic authentication

modified org.apache.camel.component.cxf.CxfEndpoint.java

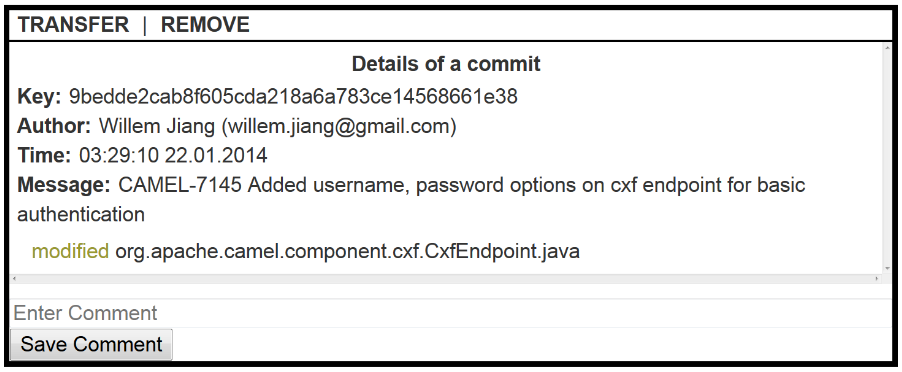
Figure 3.17: **Data set showing Jira issue details.** The project (CAMEL), the issue key (CAMEL-7145) and the issue name are shown on the very top of the content area. Four main sections exist: Details, Description and Activity. The Activity section can contain the sub-sections JIRA Comments and Related Commits.

help to identify the technical reasons. Furthermore, they are a link to the rationals behind the changes. Assumed that Jira is used as issue tracker, a well formed commit will contain the key of a Jira issue and therefore, allow to identify it. If the impact of a specific CRQ must be reviewed, the commits help to identify which components have been changed and by whom. Three queries have been provided, that help to utilize the commit data in the way that has just been described.

Details of a commit. By providing a commit key as query parameter, the details of the corresponding commit can be displayed. The name of the committer, the commit text, the commit time and a list file changes are shown. The details of a commit are shown in Figure 3.18.

Diff of a file change. The diff of a specific file change can be displayed by means of this query. This is shown in Figure 3.19. The key of the related commit, the commit time, the name of the affected file and the file diff are part of the DS. The key of the file change needs to be passed as query parameter.

Commits related to a specific component. In order to list all commits that have changed a specific component, this query can be executed. The full name of the component must be passed as parameter. The result will be rendered as table. The key, time and message of the commit as well as the committer will be shown in the table.

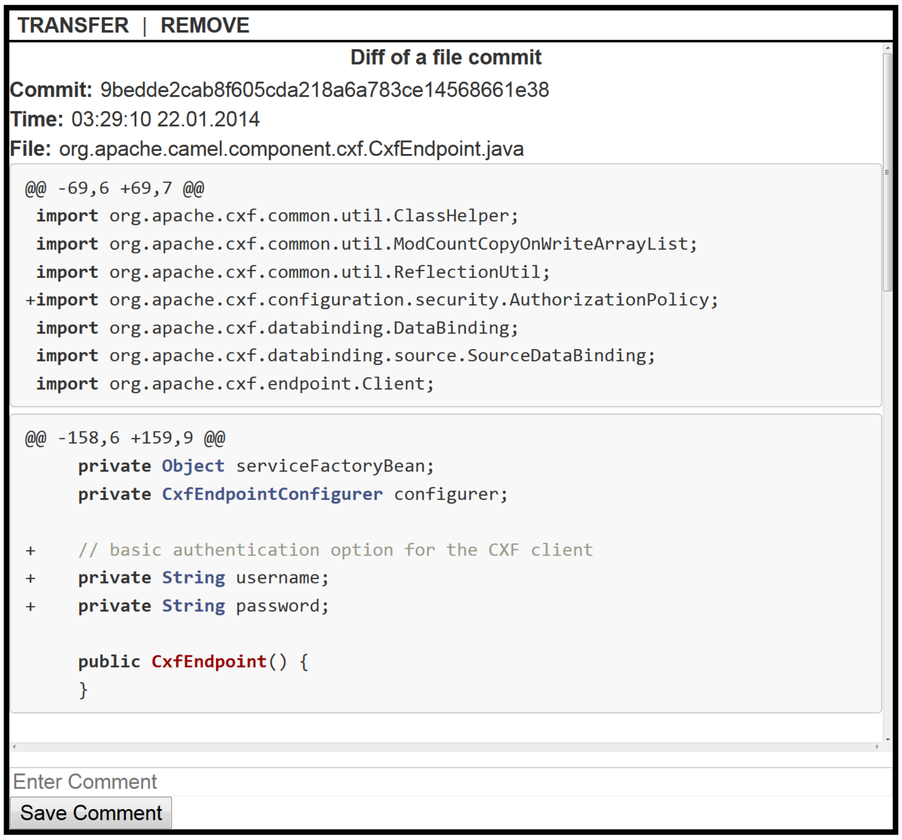


Details of a commit

Key: 9bedde2cab8f605cda218a6a783ce14568661e38
Author: Willem Jiang (willem.jiang@gmail.com)
Time: 03:29:10 22.01.2014
Message: CAMEL-7145 Added username, password options on cxf endpoint for basic authentication

modified org.apache.camel.component.cxf.CxfEndpoint.java

Figure 3.18: **Data set showing the details of a commit.** The key of the commit, the committer (author), the commit time, the commit message and the name of all affected files are shown.



Diff of a file commit

Commit: 9bedde2cab8f605cda218a6a783ce14568661e38
Time: 03:29:10 22.01.2014
File: org.apache.camel.component.cxf.CxfEndpoint.java

```
@@ -69,6 +69,7 @@
 import org.apache.cxf.common.util.ClassHelper;
 import org.apache.cxf.common.util.ModCountCopyOnWriteArrayList;
 import org.apache.cxf.common.util.ReflectionUtil;
+import org.apache.cxf.configuration.security.AuthorizationPolicy;
 import org.apache.cxf.databinding.DataBinding;
 import org.apache.cxf.databinding.source.SourceDataBinding;
 import org.apache.cxf.endpoint.Client;
```

```
@@ -158,6 +159,9 @@
     private Object serviceFactoryBean;
     private CxfEndpointConfigurer configurer;

+    // basic authentication option for the CXF client
+    private String username;
+    private String password;

     public CxfEndpoint() {
     }
```

Figure 3.19: **Data set showing the diff of a file change.** The key of the related commit, the commit time, the name of the affected file and the file diff are shown.

3.3 Limitations

The application in the current state suffers from a set of drawbacks and limitations. This is due to the time constraints of a master thesis and the prototypical state of the application. This limitations should be addressed by future work. However, the general application design should support the easy implementation of necessary features. Some of the described limitations will be further discussed in the future work section of Chapter 6.

Awareness and communication support. When designing an application for collaborative work, a central problem is the support of awareness (see Section 2.3). When multiple users work with the same data or on the same space, they must get informed about other users actions. However, it is hard to provide the right amount of awareness information. Too much and too frequent information about other users actions may disturb a user's concentration. Too less information may interfere with the efficient collaboration among users. Currently, a very limited set of awareness supporting information is given on the DSVP. A user can not see if a DS is currently selected or modified by another user. It is not possible to know, which part of the working area is currently observed by another user. Furthermore, the possibilities of direct communication between users are limited. Users are not able to point on DSs or mark a section of the working area. Furthermore, user can not chat with each other without the utilization of other tools as Skype. More awareness supporting information and possibilities of direct communication are necessary and should be supported.

Visibility, affordance, application feedback and help. The general visibility and affordance of application features must be increased. For example, it is currently not obvious with which values of a DS a user can interact. Furthermore, not enough feedback to certain actions is given. This is especially true for executed queries. Until the DS is rendered, no information about the progress of the executed query is given. The lack of instructions and help prevents a fast learning of the concepts and functionalities of SQAC.

Overview and organization. Even though functionalities for organizing DSs on the space are given, their usefulness is limited. It can still be challenging to work with very crowded spaces, especially on small notebook screens. Future work should enhance the usage of the semantic information to provide high level views on the existing data. DSs are only rendered in one fixed size. They should become resizeable in order to improve the readability or to decrease the space taken.

Review process, space constraints and guidance. The DSTP and the DSVT are not putting any constraints on the way user collaborate. There is no intrinsic process that guides the users, no hierarchy between users or ownership of DSs. A more structured review process and more constrained spaces can potentially improve the reviews. By the concept of STs and SVTs, the fundamentals for more structured spaces are given.

Chapter 4

Technical Design

In this chapter, details of the technical design and the implementation of SQAC will be described.

Since the application is built based on the object oriented paradigm, most classes, traits and objects are representing a specific entity or concept. In order to enable a more fluent way of writing, the following convention is used in this chapter: An entity name refers to an instance of the corresponding class. Example: *a car* means an instance of the class *Car*. The used code snippets are not necessarily complete. Parts that are not necessary to understand the general structure and concept have been omitted. In the following, the term *backend* refers to the logic on the server side. *Frontend* means all logic executed in the browser of a user and most importantly the SVPs.

4.1 Used Technologies

In the following, a short introduction of some of the used technologies is given and their usage will be motivated. The chapter will conclude with some of the experiences made with these technologies.

4.1.1 Scala

Scala is used as the programming language for the backend of the application. A good general introduction into Scala is given in [OSV08]. The Scala compiler is producing Java byte code and hence, Scala applications are running on the *Java Virtual Machine* (JVM). A JVM based language was selected in order to create an application that is independent of a specific operation system. Even though Scala is not a pure functional programming language, it supports functional programming patterns. Functions are first-class objects in Scala, i.e., they can be passed as parameters to other functions. This is a powerful and helpful feature that can reduce a lot of implementation overhead. Especially, the implementation of many of the classical *Gang Of Four - Design Patterns* (GOFP) [GHJV94] is eased. Some of the patterns are even obsolete. See [BL13] for a detailed discussion of this topic. Another example for a powerful feature of Scala is its Collection-API. It offers functions like map and fold, that are based on lambda expressions[1] and allow for the very efficient implementation of many algorithms. The class *Option[E]* and the *None* object help to avoid problems introduced by the usage of Java's *null* value [BL13]. *Option[E]* is related to the null-object design pattern that has been described in [GHJV94]. Scala's pattern matching mechanism and its parser combinators allow for the development of expressive *Domain Specific Languages* (DSLs). See [Gos11] for an introduction to Scala DSL development and a good

[1]http://en.wikipedia.org/wiki/Lambda_calculus

comparison to other JVM languages. Some of the named Scala features are presented in Listing 4.1.

```
// Collection API Examples
val list = List(1,2,3,4,5,6)
// list: List[Int] = List(1, 2, 3, 4, 5, 6)
val doubled = list.map( listElement => listElement * 2 )
// doubled: List[Int] = List(2, 4, 6, 8, 10, 12)
val sum = list.foldRight(0)( (listElement,total) => listElement + total )
//sum: Int = 21

// Pattern Matching and None Object Example
// getUserById : ( Int => Option[User] )
getUserById( userId ) match {
  case Some(theUser) => println("Found: " + theUser)
  case None => println("No user found")
}
```

Listing 4.1: Examples of Scala language features

4.1.2 Play 2

Play 2 is a fullstack web application framework. Therefore, it contains all components necessary to develop such an application. Play 2 ships with an application server, a build environment, a dependency injection service and different libraries that support the implementation of web application functions. The framework exists in a Java and Scala version. The latter has been selected since Scala is used as basic programming language. Play 2 offers a very convenient way to develop REST-based[2] web applications. A complete introduction to the Play 2 framework can be found in [HBC14].

Play 2 supports web sockets via Scala's Enumeration and Iteratee API. A good support of web sockets was a critical selection criteria since it is used as the basic technology for the synchronisation of SVPs. The AKKA[3] library is part of Play 2. This library supports the easy distribution of web applications. Furthermore, it supports the implementation of message queues and distributed concurrent algorithms via the Actor[4] programming pattern.

One of the biggest strengths of the Play 2 framework is its support for JSON[5] handling. JSON is the primary data format used for the communication between the frontend and the backend of SQAC. A set of very convenient functions and classes are provided, that allow to read and write JSON. The Listing 4.2 explains the basic mechanism. The central classes for JSON handling are *Format[A]*, *Reads[A]* and *Writes[A]*. A *Writes[A]* is needed to write a JSON version of an object. To create a Scala object based on JSON, a *Reads[A]* is needed. A *Format[A]* is a composition of a reads and a writes. Play 2 allows for the automatically creation of reads and writes for case classes. It is also easily possible to create own *Reads[A]* and *Writes[B]* implementations.

[2]http://de.wikipedia.org/wiki/Representational_State_Transfer
[3]http://akka.io
[4]http://doc.akka.io/docs/akka/snapshot/general/actors.html
[5]http://www.json.org

```scala
case class Car ( vendor : String , price : Double )
val myCar = Car( "Honda" , 10000 )
// A Format[Car] includes a Reads[Car] and a Writes[Car]
val format : Format[Car] = Json.format[Car]
// Json.toJson[Car] needs a Writes[Car] as parameter or in the implicit scope
val carToJson = Json.toJson[Car](myCar)(format)
// carToJson: play.api.libs.json.JsValue = {"vendor":"Honda","price":10000.0}
// Json.fromJson[Car] needs a Reads[Car] as parameter or in the implicit scope
val carFromJson = Json.fromJson[Car](carToJson)(format)
// carFromJson: play.api.libs.json.JsResult[Car] = JsSuccess(Car(Honda,10000.0))
```

Listing 4.2: Example of Play 2 JSON handling

4.1.3 Apache Jena

Apache Jena[6] (Jena) is a Java framework that supports the development of semantic web applications. It is used for the implementation of the semantic model of SQAC. An introduction to this topic has been given in Section 2.2. Jena supports the storage of RDF triples. Data can be imported via RDF files or by manually adding statements to the model. The latter is shown in Listing 4.3. Jena distinguishes between *DatatypeProperties* and *ObjectProperties*. The former is used, if the object of a statement is a literal value and the latter, if it is a resource. The web ontology language can be used to define ontologies. Reasoners are supported and can be used to infer information. Jena supports the execution of SPARQL queries and has been successfully used in other projects [WGG13].

```scala
// Example: Add statement "myCar hasVendor 'Honda'" to model
// OntModel represents an RDF model with ontology and reasoning support
val model : OntModel = [ ... ]
// Create a car resource
val car = model.createClass("Car")
val myCar = model.createIndividual("myCar", car)
// Add a vendor property
val hasVendor = model.createDatatypeProperty("hasVendor")
myCar.addProperty(hasVendor,"Honda")
```

Listing 4.3: Example of Apache Jena data import

4.1.4 Javascript and CSS Libraries

Javascript and a set of Javascript libraries are used to implement the frontend functionalities of SQAC. The basic library used is jQuery[7]. It is an established industry standard library and enables a high number of additional libraries like jQuery UI[8]. Additional infrastructure libraries like backbone.js[9] have been considered, but they did not seem to offer benefits in the context of SQAC. The NVD3.js[10] library is used for rendering the charts. It is built on top of the popular D3.js[11] library. NVD3.js provides helper functions and predefined styles for the creation of different chart

[6]https://jena.apache.org
[7]http://jquery.com
[8]https://jqueryui.com
[9]http://backbonejs.org/
[10]http://nvd3.org
[11]http://d3js.org

types. For browser side syntax highlighting, the rainbow.js[12] library has been used. It is very easy to use and supports the registration of custom handlers for specific code patterns. The bootstrap[13] CSS library was used for the design of the fronted.

4.2 Implementation Details

SQAC is a Play 2 web application. The functions of the application can be accessed by means of a REST API. Users interact with the application via an HTML frontend. The high-level architecture consists of two layers. These are the framework layer and the application layer.

The framework layer contains the implementation of the basic concepts described in Chapter 3. Therefore, it contains the code artefacts representing spaces, SVs, DSs, STs, SVTs, events and queries as well as corresponding default implementations. Furthermore, the infrastructure which allows users to join spaces and the event handling infrastructure are implemented in the framework layer. A set of Javascript classes are also provided. These classes support the development of SVPs by offering functionalities for the communication with the server, the rendering of DSs and the creation of queries and events. Data storage and data handling are also a responsibility of the framework layer. Nevertheless, it does not provide classes for accessing specific data repositories.

The application layer contains the domain specific implementations of the application that is built on top of the framework layer. Such an application has to implement interfaces for importing data from repositories and to specify a corresponding semantic model. It must also implement matching queries. The framework provides default classes for the backend representation of spaces and SVs. The SVP rendered for an SV and the corresponding SVT must be implemented by the application. Even though, SQAC is bound to the domain of SQRs, it would be easily possible to build applications of alternative domains on top of the framework layer.

4.2.1 Semantic Model and Access of Data Repositories

As discussed in Section 4.1, the data model used by SQAC is a semantic model based on Jena. The framework layer provides a class named *DataStore* which represents the DST introduced in Chapter 3. It is a wrapper around a Jena *OntModel*. A DST is created during the application start-up. It is the access point to the data used by the application. In order to import data from a specific repository, a data source adapter has to be implemented by extending the trait *DataSourceAdapter*. Each of these adapters will be called during the initialization of the application. An adapter has access to the DST and can write data into the underlying model. Currently, adapter for Git-, Sonar- and Jira-repositories have been implemented.

The underlying ontology has to be provided by the application layer. The framework contains no such information for a specific domain. Each data source adapter is responsible for the correct usage of the provided ontology and the right linking of the data. The model of SQAC is based on the SEON ontology that has been introduced in Section 2.2. The components main, issues, clones, history and jira are currently used, since they apply to the domain of the imported data. The ontology has been extended in order to support Sonar data.

DSs and queries need to provide semantic information. The former for its values, the latter for its input parameters. In order to describe the semantic of a specific value, a statement based

[12]http://craig.is/making/rainbows
[13]http://getbootstrap.com

format is used. Assuming that the value *bob@example.ch* has the semantic of the email address of a stakeholder, its description would be: *main#Stakeholder main#hasEmail*. In this example, *main* represents a namespace of the SEON ontology. *Stakeholder* is a class and *hasEmail* a data type property. The usage of this format has the following reasons: Data type properties do not have types that are specific enough to be used for the identification of relations or the recommendation of queries. The email address in the example above has the XML schema type *string*, as several other values, that are not email addresses. Modelling the email address as a class is not a solution, since the identifier of a resource (URI) is in most cases not the value that is displayed to the user. The URI of the email above could be *main#bob_example_ch*. Hence, the email class would most likely need a data type property containing the readable email address. It would be the same situation as before. By describing the relation of a value to its parent class, its semantic is unambiguous. The class *ValueSemantic* is used to store this information for one value. This class is shown in Listing 4.4.

```
case class ValueSemantic (
  nsClass : String,
  className : String,
  individualName : String,
  nsProperty : String,
  propertyName : String
)
```

Listing 4.4: Specification of the class ValueSemantic

4.2.2 Spaces, Space Views and Data Sets

Spaces and SVs are represented by the class *Space* and the trait *SpaceView*, respectively. An SV has an SVT represented by the trait *SpaceViewType*, which is shown in Listing 4.5. If a user joins a space, he will be automatically assigned to an SV. The SVT determines which kind of users can be assigned to an SV. Furthermore, it defines the SVP that will be rendered for a user connected to the corresponding SV. An implementation of the trait *SpaceType* contains meta information about a space. The trait is shown in Listing 4.6. Each space is associated with one ST. The type defines which SVTs are supported by the space. A space is associated with one SV of each SVT. If a new space is created, its SVs will be created, too. The relations between *Space*, *SpaceView*, *SpaceType* and *SpaceViewType* are shown in Figure 4.1.

```
trait SpaceViewTypeDescription {
   val name : String
   val description : String
   // Rules are not used, yet
   val rules: scala.collection.immutable.List[ViewEventRule]
}

trait SpaceViewType extends SpaceViewTypeDescription {
   // Create the HTML page
   val page :
     (SpaceDescription,SpaceViewDescription,Client,Request) => Html
   // Decide, if a user will be assigned
   val supports : Client => Boolean
}
```

Listing 4.5: Specification of the trait SpaceViewType

```
trait SpaceType {
  val availableSpaceViewTypes : Set[SpaceViewType]
  val description : String
  val name : String
  val id : SpaceTypeId
  // Rules are not supported, yet
  val rules : scala.collection.immutable.List[SpaceEventRule]
}
```

Listing 4.6: Specification of the trait SpaceType

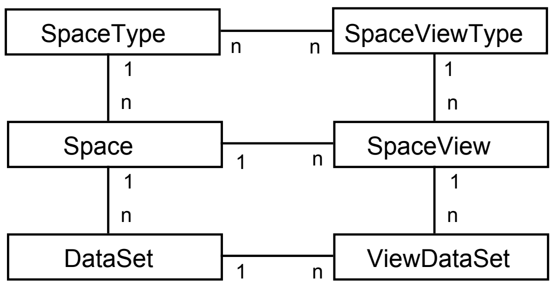


Figure 4.1: **Entity relationship model of space and related entities.** Relation between Space, SpaceType, SpaceView, SpaceViewType, DataSet and ViewDataSet. Each DataSet is associated to one ViewDataSet. Each SpaceView is associated to one ViewDataSet per DataSet, that is associated with the corresponding Space.

The traits *SpaceEventRule* (see Listing 4.5) and *ViewEventRule* (see Listing 4.6) will support a rule system in a future version of SQAC. They will enable the definition of rule classes, that will be checked before an event is allowed to change a space or an SV. The support of rules is not implemented yet.

Each space as well as each SV has properties. This properties are implemented as a Map of the Scala Collection-API. It can contain arbitrary key-value mappings. The properties can be changed by specific events (see Section 4.2.3) that are triggered by user actions on an SVP. The properties are used to store information that are necessary for the view synchronization. For example, as described in Section 3.2.1, a space gets blocked, if an old history state is loaded. A corresponding flag will be written into the properties of a space. If a new user joins the space, the properties will be passed to its SVP. It can read the information and initialize itself accordingly. This approach has the drawback, that no type safety is ensured. Furthermore, since different SVTs are free to use any properties, conflicts are possible. However, it is very flexible and allows a fast development of different SVPs. Arbitrary complex data can be stored in the properties, since the map values can be JSON strings. A possible improvement is the rule mechanism described above. Rules can prevent property conflicts or constrain the property values otherwise. Likewise, the used properties can become part of the static ST or SVT definition.

A space can be associated with multiple DSs which are represented by the trait *DataSet*. Like spaces and SVs, DSs have properties that are also implemented as a map. While the actual DS content - the result of the query - is immutable, the properties can be changed by events. For each DS multiple corresponding *View Data Sets* (VDSs) exist. They are instances of the class *View-DataSet*. Each SV is associated with one VDS per DS on the space. The relation is shown in Figure 4.1. A VDS is a wrapper around the original DS, that can store additional SV specific information about it. For this purpose, a VDS has its own properties. The properties of a DS and the properties of a VDS are supposed to contain different types of information. DS properties should contain solely information that are relevant for all SVs. The VDS properties should store information that are only relevant for the parent SV. The reason for this is the basic idea behind SVs. They enable multiple kinds of representation of the DSs on a space. Not only the representation can differ, but the whole concept of interaction. As described in Section 3.2, the DSVP is realizing a working area on which DSs can be moved. The position of the DS must be stored, otherwise it is not possible to synchronize the SVP for new users who are joining the space. This position information will be stored in the properties of the VDSs. A different type of SV may represent DSs as boxes with fixed positions. Therefore, the position information has no meaning in the context of this SV. On the contrary, a comment should be displayed in each SV. They are added to the properties of the DS.

Two implementations of the trait *DataSet* exist: *SingleDataSet* and *DataSetCollection*. The former is the default implementation used in the application. DS collections will support the aggregation of multiple DSs. The corresponding function is not fully implemented yet. The actual data of the DS is stored as a JSON object. The SVP entails its structure via the announced ways of rendering. They are stored as a list of strings in the DS. The JSON data-object has to contain the semantic information of its values. Section 4.2.1 gives a description of the representation of value semantics. Another part of a DS is the context information. This is meta data about the DS. It is currently solely used to store the parameters that have been passed to the query, that created the DS. Context data is represented by a list of *ContextValue* objects, which contain the parameter value and its semantic. *DataSet* and *SingleDataSet* are shown in Listing 4.7.

```
trait DataSet {
  val dataSetType : DataSetType // Enum: SingleDataSet, DataSetCollection
  val id : DataSetId
  var properties : Map[String,String]
  val description : String
}

case class SingleDataSet(
  id : DataSetId,
  data : DataSetData,
  var properties : Map[String,String],
  description : String = ""
) extends DataSet

case class DataSetData(
  drawableAs : List[String],
  data : JsValue,
  context : List[ContextValue]
)
```

Listing 4.7: Shortened specification of DataSet and SingleDataSet

4.2.3 Event Handling

The interaction between users and spaces is event-driven. User actions on an SVP can trigger *Client Events* (CE), that will be published to the space, as part of an *Client Event Message* (CEM). Each event is represented by a singleton object that is extending the trait *ClientEvent*. The term event, as used here, has a slightly different meaning than in the classical publish-subscribe context. An event has actually characteristics of a command object corresponding to the GOFP [GHJV94]. Each CE represents a specific action that will be performed on the target space. Depending on the kind of event, this action is implemented as part of the space logic, or implemented by the object representing the event. The data that is needed to perform the event action, is part of the CEM that is sent by the SVP. The data is stored as a JSON object.

Currently, the following types of client events exist: *Execution of a query*, *adding of a DS*, *removal of a DS*, *change of the properties of the space*, *change of the properties of a DS*, *change of the properties of a SV*, *change of the properties of a VDS*, *transfer of a DS* and *reload of a history state*. All functions provided by an SV, respectively the SVP, are mapped to those types of events.

An example for the general process: Adding a comment to a DS will trigger a *Data Set Modification Event* (DSME), which will write the new comment into the properties of the DS. The space will identify the event by means of the event ID. The corresponding event singleton object provides a function *performeModify(Map, JsValue) => Map*. It receives the properties of a DS as well as a JSON object. It will return a new instance of properties. The space needs to load the properties of the DS that is supposed to be changed. Hence, the CEM must containt the DS ID. The trait that is representing the DSME specifies an attribute of the type *Reads[DataSetId]*. This is used to read a DS ID from a JSON object. The space will use the reads to extract the id from the JSON data that is part of the CEM. Afterwards it will load the identified DS. The *performeModify* function will be executed and the resulting properties will be set as the new DS properties. A shortened version of the corresponding code is shown in Listing 4.8. The handling of other kinds of events is similar to the process that has just been described.

```
trait DataSetModifyEvent extends SpaceEvent {
  val dataSetIdReads : Reads[DataSetId]
  val performeModify : (Map[String,String],JsValue) => Map[String,String]
}

case object GenericDataSetModifyEvent extends DataSetModifyEvent {
  val id = EventId("DataSetModifyEvent")
  val dataSetIdReads : Reads[DataSetId] = // [...]
  val performeModify =
        (properties: Map[String,String], input: JsValue) => {
           // [...]
        }
}

// In class Space
def handleDataSetModifyEvent(e: DataSetModifyEvent, message: JsValue) = {
   val dataSetId = Json.fromJson[DataSetId](message)(e.dataSetIdReads).get

   this.dataSets.get(dataSetId) match {
      case Some(ds) => {
        val modifiedProperties = e.performeModify(ds.properties, message)
        ds.properties = modifiedProperties
      }
    }
```

Listing 4.8: Shortened version of the DataSetModifyEvent handling

CE are categorized as *Space Events* (SE) and *Space View Events* (SVE). SEs are all events that change the state of a space. The effect of an SE is relevant for all users, independently to which SV they have been assigned. Therefore, all users connected to the space will be notified about the event. In contrast, SVEs change the state of an SV. The SVP of users who are assigned to the affected SV will be notified. All events named above except the change of SV properties and the change of VDS properties are SEs. The distinction reduces communication overhead. It increases the performance in scenarios, where multiple SV are provided by a space and many users are connected at the same time.

The event handling infrastructure is based on web sockets. When an SVP has been rendered, a web socket connection to the server is created. By means of this connection, messages can be sent and received. In order to push a CE to the space, a CEM must be created and sent via the web socket connection.

The class *EventDispatcher* is the central point of communication on server side. Each space is associated to its own *Event Dispatcher* (ED). EDs are managing the web socket connections. They receive the CEM and put them on a message queue. All messages on the queue will be handled serially. When a CEM is pulled from the queue, it will be passed to the space. The space will identify the event and execute the corresponding action. When the event handling is done, a *Server Message* (SM) is created that contains information about the event that has just been executed. The SM will be put in the message queue of the ED. It will be pushed to the SVPs via the web socket connections. They will adapt itself accordingly. Since CEM as well as SM are added to the same queue, the correct ordering of events is guaranteed.

The described message queue is implemented using the AKKA[14] library that is provided by Play 2. The library supports the implementation of the Actor-pattern[15]. An actor is an object that can receive and send different kinds of message. The actor will process one message at a time. Putting a message on the queue actually means to send a message to the actor. Each ED is an actor. The processing of the messages is performed asynchronously. A thread is not blocked by sending a message.

4.2.4 History Access

Each space is associated with an *Event History* (EH) that contains all CE received and executed by the space. It is possible to navigate through the EH and temporarily set the space back to an old state (see Section 3.2.1). This function is implemented via undo events and redo events. For each type of CE an undo event class and a redo event class exists. This classes are extending the traits *UndoClientEvent* and *RedoClientEvent*. An instance of these two classes is created for each CE which is handled by the space. The EH consists of *Event History Items* (EHIs), which are implemented by the class *EventHistoryItem*. An EHI contains the original CE, the corresponding event data, the undo event and the redo event.

[14]http://akka.io
[15]http://doc.akka.io/docs/akka/snapshot/general/actors.html

A space can be set to a specific state by an event named *GoToHistoryItemEvent*. Part of the CEM must be the ID of the target EHI, that is supposed to be set. Navigating backward relative to the current state, is implemented as a series of undo event executions. Going forward in the history means executing all redo events between the current state and the target state. Undo and redo events will not be added to the EH.

In most cases, the redo event will be just an execution of the original CE. The redo of a query event is implemented as an *add data set* event. The query redo event will store the result DS. This is necessary since the DS needs to keep the same content and the same ID. This would not be guaranteed, if the original query would just be executed again.

The implementation of the space state change needs future optimization in terms of scalability. Performing big jumps in the history can trigger a considerable number of events. An optimization would be possible by skipping unnecessary events. For example, it is not necessary to undo a set of DS modifications, if the corresponding DS will not exists any more, when the target state is reached.

4.2.5 Query Implementation

A query can be implemented by extending the trait *Query*. The specification of the trait and other classes described in this section is shown in Listing 4.9. Beside a set of attributes for describing the query, the trait defines a list of *SupportedInput*, that represents the parameters which will be accepted by the query. A *Supported Input* (SI) contains information that can be used to automatically render a matching input form. This is done for the queries that are listed on the DSVP (see Section 3.2). The semantic of the corresponding parameter is part of an SI. It is represented by the class *ValueSemantic*, that has been described in Section 4.2.1.

The *Query* trait specifies a function *executeOn(DataStore,QueryInput) : Option[DataSet]* that will be called when a query is executed. The parameters that are used for the query are stored in instances of the class *QueryInput*. The query can access the underlying semantic model via the DST reference that is passed as a parameter. The actual underlying queries are implemented via SPARQL.

The case class *GenericQuery* is extending the *Query* trait and can be used for the definition of queries. The implementation of the *executeOn* function is passed as a strategy (see [GHJV94]), supported by Scalas first order functions.

Each query is supposed to be a stateless singleton object, similar to the implementation of events. It can be seen as an operation that is executed without side effects on the DST. If a space receives a query event, it will load and execute the query by means of the *QueryManager* object.

```
trait Query extends QueryDescription {
  def executeOn(dataStore : DataStore, input: QueryInput) : Option[DataSet]
}

trait QueryDescription {
  val id : QueryId
  val category : String // Jira , SonarQube , Commit ...
  val description : String
  val supportedInput : List[SupportedInput]
}

// input: Parameter -> Value, E.g. 'creationDate' -> '01.01.2014'
case class QueryInput(queryId: QueryId, input: Map[String,String])

case class SupportedInput(
  name : String, // E.g. creationDate
  label : String, // E.g. Creation Date
  hint : String, // E.g. dd.MM.yyyy
  dataType : String, // E.g. Date
  semantic : ValueSemantic, // E.g. Issue hasCreationDate
  options : List[String] = List(),
  mandatory : Boolean = false
)

case class GenericQuery(
  id : QueryId, category : String, description : String,
  supportedInput :
     List[SupportedInput], queryCode : (DataStore,QueryInput) => Option[DataSet]
) extends Query {
    override def executeOn(dataStore: DataStore, input: QueryInput) = {
      queryCode(dataStore, input)
    }
}
```

Listing 4.9: Specification of Query, QueryInput and SupportedInput

4.2.6 Application Configuration

The link between the framework layer and the application layer is created by using of the *ApplicationConfiguration* trait, which is shown in Listing 4.10. The application layer must provide an implementation of this trait. It is possible to access the application specific information via the corresponding class. In example, the definition of STs and SVTs is the responsibility of the application layer. Since the framework is responsible for creating and managing spaces, it needs to access the provided types. The same is true for the different queries and events, that must be accessed when a client event is triggered by an SVP.

Only traits and classes which are provided by the framework are used in the *ApplicationConfiguration* trait. Hence, it is not introducing functional dependencies from framework layer to application layer.

```
trait ApplicationConfiguration {
  def getInitialSpaces : Set[(SpaceType,String,String)]
  def getSpaceTypes() : Set[SpaceType]
  def getSpaceViewTypes() : Set[SpaceViewType]
  def getQueries() : Set[Query]
  def getDataStore() : DataStore
  def getRules() : Set[Rule]
  def getClientEvents() : Set[ClientEvent]
  def getClients() : Set[Client]
}
```

Listing 4.10: The trait ApplicationConfiguration

4.2.7 Data Set Rendering

DS rendering is implemented in both, the backend and the frontend. In the backend, queries must produce DSs that contain data in a specific format and announce the matching way of rendering. In the frontend, a renderer must exist that can handle the corresponding rendering type. The SQAC framework layer provides functions for the rendering of different generic charts. The Javascript object *SqacChartRenderer* encapsulates these functions. The application layer adds more specific renderer that are, e.g., able to display Jira issues (see Section 3.2.3). The corresponding objects are *BasicAppSonarDataRenderer*, *BasicAppJiraDataRenderer* and *BasicAppCommitDataRenderer*.

Each renderer provides a set of rendering functions that are corresponding to specific ways of rendering. The DSVP will provide a container element to this functions. The resulting HTML code will be injected in this container. If the DSVP receives a SM that contains a new DS, it will solely create the outer frame of the rendered DS and the comment area (see Section 3.2.2).

A rendering function will receive two other functions as parameters. A callback function for clicked values and a function for the registration of element semantics. Each renderer is responsible for adding event listener to the HTML elements which are containing DS values. A click will call the provided callback function and pass the value and the semantic of the clicked element. Based on this callback, the VCM described in Section 3.2.2 is created. A renderer will use the registration function to register all the value elements and their semantics at the DSVP. The elements of all DSs are added to an index. It will be used, if the relation between DSs must be highlighted, as described in Section 3.2.2. Via the index, DSs that are containing relevant values can be identified. Since the corresponding HTML elements are registered, they can be highlighted, too.

4.3 Conclusion of Technical Details

4.3.1 Lessons Learned

A central goal of the technical design was the development of a framework on which multiple applications of the same basic concept can be built. Even though, the current design should be reviewed and improved, the goal has be achieved by the framework layer. The design of such a layer is challenging. The responsibilities of the layer must be designed carefully. It must be decided which concepts should be implemented in it and which should only be defined by means of traits. In order to do this, the requirements of the layer clients must be foreseen. It is not easy to find the right level of abstractions and the possibility of over-engineering is given. Based on feedback of the clients, the framework must be improved and restructured regularly.

The used technologies appear to be very suited for this kind of application. Especially Scala's support of first order functions, functional programming patterns, Collection-API and general syntax allows for an efficient implementation of many algorithms. The other side of the coin is the - at least compared to Java - rather mediocre tool support. Even though, Eclipse supports Scala, the corresponding plugins do not work as well as with Java. For example, basic features as performing a renaming refactoring fail regularly, since Eclipse can not handle the Scala companion objects (see [OSV08]). Furthermore, the support by platforms like Sonar and Jenkins is very limited.

Working with the Scala version of the Play 2 framework was very convenient. Especially the support for JSON handling, the general request mapping mechanism, the type safe templates and the web socket support were very useful. The Actor-pattern support of the AKKA library was of major use for the implementation of the event infrastructure. The Play 2 convention to structure the code corresponding to the MVC-Pattern seems not to be optimal for an application like SQAC. Similar to Scala, the tool support for Play 2 is not as good as for other more popular frameworks. Furthermore, all features can only be utilized, if Play 2 is used as full stack framework. Switching from the underlying Jetty[16] to another type of container will constrain the framework.

Compared to a static relational data base schema or object relational mapping, the data import and modelling with ontologies, RDF and Jena turned out to be very fast and flexible. The work with SPARQL was convenient most of the time. However, for some complex queries, which need aggregations and groupings, the semantic model introduced problems that most likely would not appear with a rational data base scheme.

The Javascript libraries used were a big help for the implementation of the frontend functionalities. Although, D3.js and NVD3.js need a certain amount of learning, they are powerful and flexible libraries. The code highlighting via rainbow.js works very well. Nevertheless, certain performance issues are introduced by client side code highlighting as discussed in the next Section.

4.3.2 Limitations

The current version of SQAC is stable and no major bugs are known. Nevertheless, a set of technical limitations exist, that should be addressed in future versions.

Currently, all data is stored in the memory. Although, an export of spaces is possible, persistence should be addressed with future versions of the application. Especially, the EH will lead to heap problems with long running applications with many spaces.

The application is working on the latest versions of all popular browsers, but differences in the rendering of charts are known. This appears to be connected to different interpretation of Scalable Vector Graphics by the browsers. This is not a major issue, but should be reviewed in the future.

Some of the Sonar and Git related queries produce DSs, that are containing source code. This code is highlighted when the DS is rendered. The browser side code highlighting via rainbow.js is basically working well, but it is not scaling appropriately. The rendering of big pieces of code or even full classes needs a certain amount of time. This is due to the fact, that the library will inject a big number of HTML markup which leads to a high amount of string operations. It should be investigated, if the performance can be improved via alternative libraries or via server side injection of the necessary HTML markup.

[16]http://de.wikipedia.org/wiki/Jetty

The framework and application layer are currently represented as packages. In order to use the framework layer, the complete SQAC application must be copied and the application layer package must be replaced. This is inconvenient and is limiting the re-use of the framework layer. Hence, it should be extracted and incorporated into a Play 2 module. Afterwards, a new application could simply include this module and implement the application layer.

Chapter 5

Evaluation

An exploratory study was performed in order to evaluate SQAC in the context of the research question of this thesis. The goal was to gather usage information and user feedback as well as to try to identify strengths and short comings of SQAC.

In order to avoid confusion with the term *issue*, the following convention is used in this chapter: the term CRQ is used for a Jira issue. The term Sonar issue refers to an instance of a broken Sonar rule in a software project. One instance of a code smell, e.g., magic numbers, would be one Sonar issue. The term quality issue refers to one arbitrary problem with the SQ. Hence, a Sonar issue is a quality issue, but not each quality issue must be reflected by a Sonar issue. For example, a decrease of the maintainability could be a quality issue.

5.1 Study Setting

The evaluation was performed with a total of eight participants in two sessions that have be conducted on different days. One study session was performed by four participants at the same time. The participants worked together in groups of two, named group A and group B. The participants were assigned to the groups randomly. Group member were supposed to collaborate. The setting consists of two parts. In the first part, a collaborative SQR was performed by both groups. They received each others results in the second part and reviewed them. In the first part, group A used SQAC and group B Jira, Sonar and Github. In the second part, the tools were switched. The reviews of the results of part one were conducted with the same tools, that were used to create the results. The schedule of the study is shown in Figure 5.1. In the following, the group that is using SQAC in the corresponding study part will be referred to as *group SQAC*. The group using the alternative tools is referred to as *group State-Of-The-Art* (group SOTA). Note that these group names are relative to the current study part.

Data from the open source project Apache Camel[1] (Camel) has been used. An official and open accessible Jira[2] instance for Camel exists. The same is true for a Sonar[3] instance and for the corresponding Github[4] repository.

The two parts of the study are described in detail in the following.

[1]https://camel.apache.org
[2]https://issues.apache.org/jira/browse/CAMEL
[3]https://analysis.apache.org/dashboard/index/37401
[4]https://github.com/apache/camel

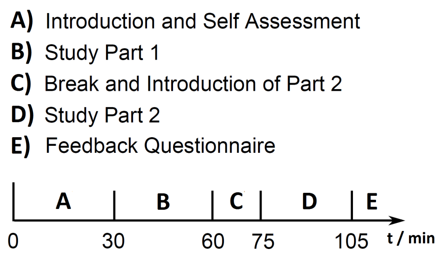

A) Introduction and Self Assessment
B) Study Part 1
C) Break and Introduction of Part 2
D) Study Part 2
E) Feedback Questionnaire
A
B
C
D
E
0
30
60
75
105
t / min

Figure 5.1: **Schedule of the evaluation study.**

5.1.1 Study - Part 1

In the first part of the study, two CRQs were assigned to each group. Each group received different CRQs to prevent a biasing in the second part of the study. A handout was given to the participants, which described their task for part 1 and necessary preparations (see Appendix A).

Both groups received the following task description as part of the study handout: *You and your group colleague are supposed to collaboratively review a number of improvements in the apache camel project. Your goal is to assess the influence of these improvements to the overall software quality of the project (e.g. test coverage, number of quality Issues, maintainability etc.). The results of your assessment will be used by another group to identify refactoring steps to increase the quality of the software.*

The assigned CRQs had the status *Closed*, therefore, they were already implemented. Changes to the source code were made in the context of each CRQ. The corresponding commits included the CRQ key in the commit message. Hence, the commits and the related code changes could be identified. All CRQs have related commits that have been conducted between 28th of March 2014 and the 31th of March 2014. In this time range, a general increase of Sonar issues can be identified. Therefore, it was possible to investigate the relation between the commits and the increase of Sonar issues. All CRQs introduced new or modified unit tests, which could motivate participants to investigate impact on the test quality of the software. CRQs that are as similar as possible with respect to the total amount of changes have been selected for both groups.

Group SQAC. Group SQAC had to use the tool SQAC for the reviewing of the CRQs. One space was provided, which should be used by both group members simultaneously. The group was told, that the final state of the used SQAC space will be their review result. Group SQAC was additionally allowed to communicate by means of SQAC. The CRQs CAMEL-7333 and CAMEL-7327 were assigned to group SQAC. A summary of these CRQs is shown in Table 5.1 and Table 5.2.

Group SOTA. Group SOTA had to use Jira, SonarQube and Github. The necessary URLs were given to both groups. A GoogleDocs text document was provided to the group, that should contain their review result at the end of the first study part. It was also possible to use the provided GoogleDocs document as alternative to Skype. The CRQs CAMEL-7312 and CAMEL-7334 were assigned to group SOTA. A summary of these CRQs is shown in Table 5.3 and Table 5.4.

Group:	SQAC
Key:	CAMEL-7333
Description:	Improvement of monitoring of routing
Resolution:	28.03.2014
Commits:	2

Commit Time	Changed Classes	Changed Test	Added Classes	Added Tests
28.03.2014	11	4	4	1
31.03.2014	2	0	0	0

Table 5.1: **Details of change request CAMEL-7333.**

Group:	SQAC
Key:	CAMEL-7327
Description:	Improvement of fault tolerance of Camel
Resolution:	28.03.2014
Commits:	1

Commit Time	Changed Classes	Changed Test	Added Classes	Added Tests
28.03.2014	2	0	0	1

Table 5.2: **Details of change request CAMEL-7327.**

Group:	SOTA
Key:	CAMEL-7312
Description:	Introduction of new type converter
Resolution:	20.03.2014
Commits:	2

Commit Time	Changed Classes	Changed Test	Added Classes	Added Tests
20.03.2014	1	1	0	0
28.03.2014	1	0	0	0

Table 5.3: **Details of change request CAMEL-7312.**

Group:	SOTA
Key:	CAMEL-7334
Description:	Extension of event notification
Resolution:	30.03.2014
Commits:	1

Commit Time	Changed Classes	Changed Test	Added Classes	Added Tests
30.03.2014	5	4	2	0

Table 5.4: **Details of change request CAMEL-7334.**

For both groups, the style or specific content of the review result was not defined. The participants were told, that they shall add all information that - according to their opinion - is necessary to understand which quality issue they identified. It was stressed, that the other group should be able to comprehend the result and to work with it.

The two members of one group were sitting face to face to each other. They were not able to look at each others screens. Each group was asked only to communicate via Skype chat and to provide the conversation log at the end of the study.

A short introduction to SQAC was given to both groups. If one of the participants did not know one of the alternative tools, it was introduced, too. The latter was only necessary in the second session with Jira. Afterwards, the working time of 30 minutes started. The groups were told, that they could stop earlier if the think they are done.

5.1.2 Study - Part 2

The task of the second study part was to review the results produced by the other group in the first part. The participants should try to identify, understand and rate the documented quality issues. Furthermore, they should suggest possible solutions. The participants received a handout, which described their task and necessary preparations (see Appendix B).

The following task description was given to the participants as part of the study handout: *You and your group colleague are supposed to review the results of a software quality review. You are asked to check the plausibility of issues documented in the original review and suggest possible refactorings to increase the quality of the software.*

The tool support was switched between the groups compared to study part 1. Now, group A used Jira, Github and Sonar, while group B used SQAC.

Group SQAC. The group was granted access to the SQAC study space, that has been used in the last part. No constraints were applied regarding the usage of the space. The participants could remove, add or rearrange DSs as needed.

Group SOTA. Group SOTA was granted access to the GoogleDocs result document that has been produced by the former group SOTA.

A GoogleDocs spreadsheet was provided to each group, that should be filled during this study part. The spreadsheet contained four columns: Issue, Severity, Resolution and Comment. The content of these columns was described as follows:

Issue: One issue you could identify. This includes issues named in the original review you do not agree with.
Severity: The severity of this issue. Minor, Major, Critical, No Issue.
Resolution: What would you do (or not) in order to resolve this issue?
Comment: You can enter additional comments about the issue.

As in the first part of the study, the two members of one group were sitting face to face to each other and were asked to communicate via Skype chat. The corresponding conversation log was obtained at the end of the second study part. Additionally, group SOTA was allowed to communicate via the GoogleDocs document and group SQAC via SQAC.

The maximum working time was 30 minutes. A group could stop, if they were done faster. At the end of the second study part, all participants were asked to answer a feedback questionnaire (see Appendix D).

5.2 Data Collection

5.2.1 Study Population - Experience and Knowledge

All participants have answered a self-assessment questionnaire at the beginning of the study. Via the questionnaire the development experience of the participants as well as their knowledge about the SOTA tools and SQA in general has been assessed. The questionnaire is presented in Appendix C.

The development experience was measured in years. The participants had to state their total and professional experience. The total experience included all development experience. The professional experience addressed development as employee, freelancer, intern or trainee.

The knowledge about Jira, Sonar and Github was measured on a scale from 0 to 3 with 0 being the lowest and 3 the highest degree of knowledge. The value 0 corresponded to no knowledge about the tool. The general SQA knowledge was measured on a scale from 1 to 4. A minimum degree of knowledge - resembling to at least one related lecture - of SQA was mandatory for each participant. The value 1 corresponds to the minimum degree of knowledge. For both, the rating of tool and SQA knowledge, only integer values could be selected.

5.2.2 Collaboration And Coordination

As part of the study it should be investigated, if the use of SQAC influences the amount and type of communication between the participants. The communication logs have been analyzed. As described in Section 5.1, the communication between the participants during the task execution was restricted to written text.

For the analysis all statements made by the participants have been categorized as *Coordination*, *Collaboration* and *Misc*. A statement has been defined as a concluded and sensible series of words.

Coordination. All statements necessary for the establishment of a shared view or for the distribution of work task have been categorized as Coordination. Such communication is necessary multiple times in a collaborative work scenario.

Collaboration. All communication that is part of the actual work has been categorized as Collaboration. For example: discussions about the DSs or technical questions.

Misc. All statements that are neither necessary to fulfill the task or to coordinate the collaboration have been categorized as Misc. For instance: jokes, expression of emotions and statements regarding the study task (meta questions).

To give some examples for the categorization: The question *Where is this data set?* and the answer *top right* are two statements of the category Coordination. The statement *I think this is a big issue* and the reaction *It is for sure* are both statements of the category Collaboration. A statement like *This is exhausting* or the question *Is this sill part of the study task?* are categorized as Misc.

Based on the categorization it will be especially investigated, if SQAC can reduce the amount of Coordination statements in comparison to the SOTA usage. In a distributed collaborative SQR, the lack of a shared view potentially introduces a higher need for Coordination. It should be easier for SQAC users to converge on a specific piece of data, in order to discuss it. Furthermore, the task distribution should be more efficient, since each others results can be observed.

5.2.3 Feedback

User feedback for SQAC was collected by means of a feedback questionnaire that has been provided to each participant. It is presented in Appendix D. The questionnaire was answered at the end of study part 2. Four questions were asked, regarding the usage of SQAC in general and in respect to the tasks of the two study parts.

Question 1. What did you like about the tool SQA-Collaboration? What was useful?

Question 2. What did you NOT like about the tool SQA-Collaboration? What is missing?

Question 3. For a task like the one of Part 1, would you prefer SQA-Collaboration or Sonar, Github and Jira? Why?

Question 4. For a task like the one of Part 2, would you prefer SQA-Collaboration or Sonar, Github and Jira? Why?

The answers to the questionnaire will be analyzed in order to identify strengths and problems of SQAC. Furthermore, it will be investigated, which aspects of SQAC are perceived as beneficial by the participants and which are not. A set of conclusions will be formulated based on the feedback.

5.2.4 SQAC User Behavior

The history function of SQAC will be used to analyze the usage of SQAC in both parts of the study. This analysis will help to evaluate the usefulness of the history function for the comprehension of review processes. It will be investigated, if specific patterns of collaboration can be identified.

5.3 Study Results

5.3.1 Study Population - Experience and Knowledge

Development Experience

Figure 5.2 shows the box-plots of the development experience of all participants, group A participants and group B participants. Each participant had at least five years of development experience. Groups A and B are almost equally well qualified. Means and medians of the development experience are comparable. The groups A have slightly more experience due to one outlier value. The groups A have slightly more professional experience than the groups B, since two members of the groups B had no professional development experience at all.

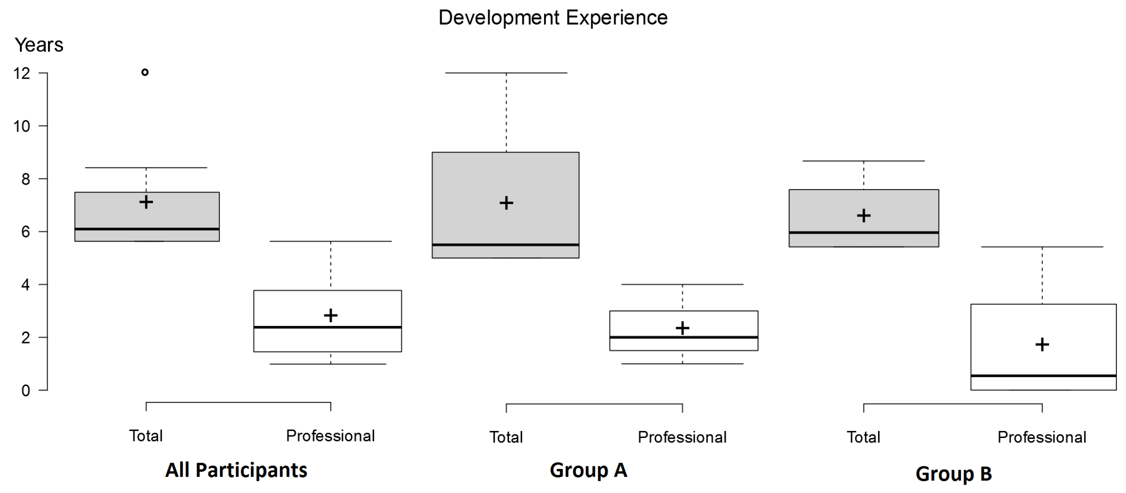


Figure 5.2: **Box-plot of development experience of the participants.**

Tool and SQA Knowledge

The box-plots of the participants knowledge of Jira, Github, Sonar as well as SQA in general are shown in Figure 5.3. The corresponding absolute number of selected values are shown in Table 5.5. The tool and SQA knowledge of the groups A and the groups B are shown in Figure 5.4 and 5.5, respectively. For the box-plots, the SQA scale has been normalized to 0 to 3.

Knowledge Level	Jira	Github	Sonar	SQA
0	1	0	1	-
1	2	2	3	1
2	2	4	3	4
3	3	2	1	2
4	-	-	-	1

Table 5.5: **Absolute number of answers concerning tool and SQA knowledge.** The scale of Jira, Github and Sonar knowledge was 0 to 3. For SQA a scale of 1 to 4 was given.

All but one participant had some or good knowledge of Jira and Sonar. All participants had at least some knowledge with Github. Groups A had a better knowledge of Jira in terms of means and median. Regarding Github and Sonar, both groups had almost the same amount of knowledge, with slightly more Sonar knowledge in groups B. The knowledge of SQA was higher in groups B than in groups A.

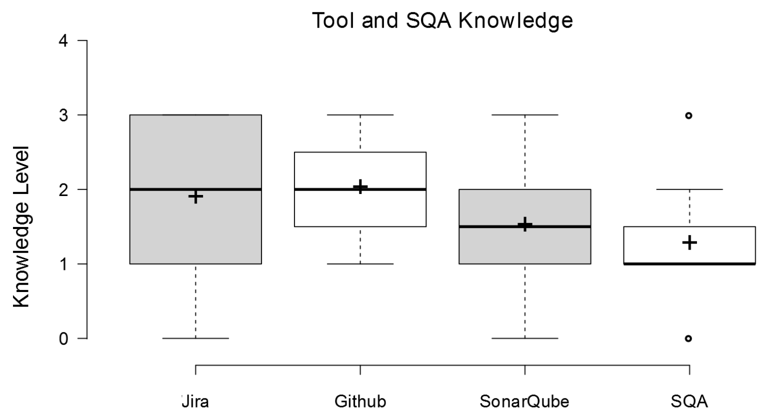


Figure 5.3: **Box-plot of tool and SQA knowledge of all participants.**

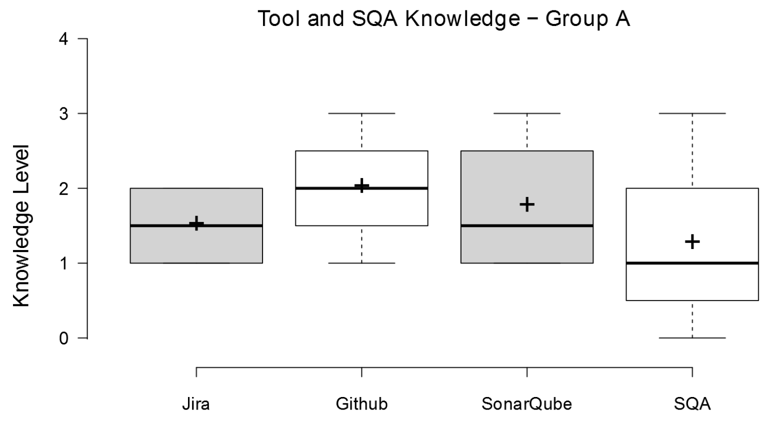


Figure 5.4: **Box-plot of tool and SQA knowledge of group A.**

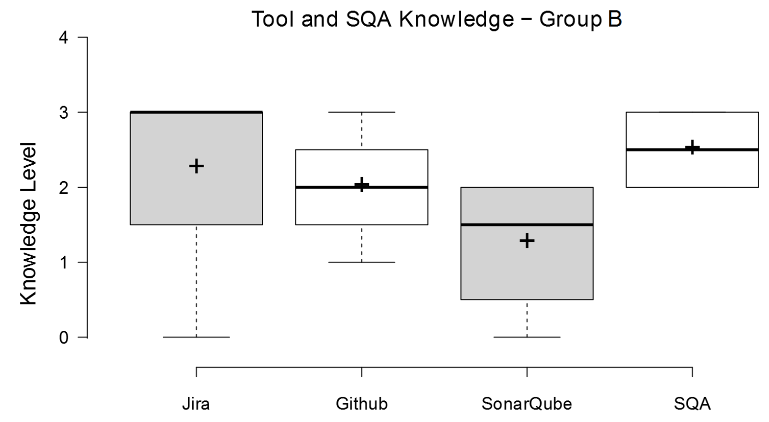


Figure 5.5: **Box-plot of tool and SQA knowledge of group B.**

Discussion

The general development experience as well as the tool and SQA knowledge is fairly equally distributed between the groups A and B. According to the self-assessment, it is not expected that the results will be significantly biased by the qualifications of the participants. However, the general good knowledge of the SOTA tools gives the SOTA groups a certain advantage in both parts of the study. It would be possible, that the benefits of SQAC are somehow levelled by its unfamiliar concept.

5.3.2 Collaboration And Coordination

Table 5.6 shows the analysis of the communication of the SQAC groups. The analysis of the communication of the SOTA groups is shown in Table 5.7. The groups A used SQAC in the first study part and SOTA in the second study part. Groups B used the SOTA tools in the first study part and SQAC in the second study part. A comparison between the communication of the SQAC and SOTA groups is shown in Table 5.8.

In study part 1 the communication between the members of the SOTA groups was higher compared to the SQAC groups. SQAC had a larger relative amount of Coordination than SOTA. In study part 2 the communication between the SQAC groups was significantly larger than the communication of the SOTA groups. Most of the communication was done by group B in session 2. The SQAC groups had a higher proportion of Collaboration statements than the SOTA groups.

Discussion

No benefit of SQAC regarding the Collaboration to Coordination ratio could be identified in part 1. Whereas, in part 2 the Collaboration to Coordination ratio of the SQAC groups was better than the one of the SOTA groups. However, the values are of no significance due to the very low sample rate. In general, the communication between the participants was way lower than expected. Especially, the communication between the SQAC participants in the first part was very low. An exception is group B of session 2 in part 2 (SQAC), which had the highest amount of communication of all groups.

An analysis of the conversation log showed that all but group B of session 2 (GBS2) chose a rather indirect way of collaboration. The participants distributed the two CRQs at the beginning and worked on their own afterwards. GBS2 discussed all the data and coordinated nearly every step, in both, part 1 and part 2. Maybe this behavior was not observed for the other groups, because the task was not framed enough or maybe not enough incentives for collaboration were given (see Section 5.4). In the case of the groups SQAC, the unfamiliar tool might be another factor influencing the results. It is possible that the participants were more occupied with learning the tool, than with the collaboration. The SOTA groups did not have this barrier.

An important factor for part 2 may is its dependency of part 1. Without a sensible result the basis for further work is low. This is assuredly true for session 2 were group SOTA did produce nearly no results at all in part 1.

SQAC Groups

Part 1 (Group A)	Words	Statements	Collaboration	Coordination	Misc
Session 1	44	8	0	8	0
Session 2	72	16	5	8	3
	116	24	5	16	3

Part 2 (Group B)	Words	Statements	Collaboration	Coordination	Misc
Session 1	47	28	15	8	5
Session 2	640	109	66	43	0
	687	137	81	51	5

Table 5.6: **Overview of communication of SQAC groups.** In part 1 the groups A and in part 2 the groups B used SQAC.

SOTA Groups

Part 1 (Group B)	Words	Statements	Collaboration	Coordination	Misc
Session 1	72	30	10	16	4
Session 2	90	18	8	7	3
	162	48	18	23	7

Part 2 (Group A)	Words	Statements	Collaboration	Coordination	Misc
Session 1	69	29	17	12	0
Session 2	42	6	1	2	3
	111	35	18	14	3

Table 5.7: **Overview of communication of SOTA groups.** In part 1 the groups B and in part 2 the groups A used the SOTA tools.

	Part 1		Part 2		Total	
	SQAC	SOTA	SQAC	SOTA	SQAC	SOTA
Words	116	162	687	111	803	273
Statements	24	48	137	35	161	83
Collaboration	5	18	81	18	86	36
Coordination	16	23	51	14	67	37
Misc	3	7	5	3	8	10

Table 5.8: **Comparison of communication during SQAC and SOTA usage.**

5.3.3 Feedback

The feedback given by means of the feedback questionnaire has been analyzed. In the following, seven conclusions that could be entailed are presented. The participants answers can be found in Appendix E. Participants are referenced as follows: *<NUMBER><GROUP><SESSION>*. Hence, *1A1* reads as *Participant with number 1 of group A in session 1*

Conclusion 1. *The data integration aspect of SQAC collaboration is perceived as its main benefit.*

The benefits of the central integration of the necessary data repositories have been addressed by the participants multiple times. Seven out of eight participants (all but 1A1) included this aspect in their answers to question 1. Two of them (4B1,4B2) named only this aspect. Two of eight (3B1,4B2) participants chose SQAC in question 3 and justified this answer with the benefits of the data integration.

Conclusion 2. *The collaboration support of SQAC is not perceived as benefit.*

Only one participant (2A2) named the collaboration support in his answer to question 1. The same is true for the answers to question 3. The same participant would prefer SQAC in question 3, but only in a collaborative scenario.

Conclusion 3. *The application feedback and feature visibility must be improved.*

Four of the eight participants (1A1,2A1,2A2,4B2) addressed related topics in their answers to question 2. Missing feedback during the execution of queries and the visibility of clickable DS values are the main points of complain.

Conclusion 4. *The workspace overview must be improved.*

Four of the eight participants (1A1,2A1,3B1,1A2) addressed this topic in the context of question 2. Main areas of complain are the not re-sizable DSs, the overview on small screens and the overview of the type of data on the space. One participant (1A2) named an improvement of overview in question 3, as criteria for the selection of SQAC. This seems to be primary an issue during the initial data exploration (part 1), since all of these participants selected SQAC in question 4.

Conclusion 5. *The awareness and communication support must be increased.*

All participants but one (2A1) addressed problems due to a lack of awareness and communication support in their answers to question 2. Direct communication is not supported enough, it is to easy to interfere with each other and general information about other users actions are missing.

Conclusion 6. *More guidance and help is needed.*

Two out of the eight participants (2A1,3B1) addressed a lack of guidance and help in their answers to question 2. They especially addressed the high amount of available information which could overexert an user. Furthermore, the lack of a specific workflow was addressed.

Conclusion 7. *The result transfer support of SQAC is perceived as benefit.*

Six of eight participant (all but 4B2,3B1) clearly selected SQAC for question 4. They justified the selection with the comprehensibility of the results. Participant 3B1 would select the tool, but argued that the benefits depend on the right way of usage. For participant 4B2, the comprehension of the thoughts of the reviewers is not supported enough yet.

Discussion

The conclusions 3 to 6 addressed problems that have already been identified in Section 3.3. It is necessary to tackle those problems in future versions of the application. Conclusions 1 and 2 could be due to a general lack of collaboration incentives in the study design, as already addressed in Section 5.3.2. The benefits of collaboration are not perceived, since collaboration was not crucial to solve the tasks. See Section 5.4) for a further discussion of this topic. The conclusion 7 is encouraging since it indicates that the result transfer support of SQAC is perceived as beneficial.

5.3.4 SQAC User Behavior

Part 1, Session 1. Group A of session one (GA1) started with querying the details of one of the assigned CRQs. Both participants queried data related to the CRQ. In comparison to group GA2, they did not use different areas of the working area. Their general goal did not seem to be, to identify specific quality issues, but to document the current quality state of the project and the changed components. They arrange related DSs around the DS which was showing the details of the CRQ. The related DSs were containing project metrics, component metrics, the list of existing Sonar issues as well as the process of Sonar issues. It is not clear, why the group did not look into the other CRQ. Maybe, this is due to time constraints. This team stated, that they spent lots of time to get an overview of the provided functions. This statement is supported by a high number of duplicated executed queries and removed DSs in the first third of their working time. The resulting space is shown in Figure 5.6.

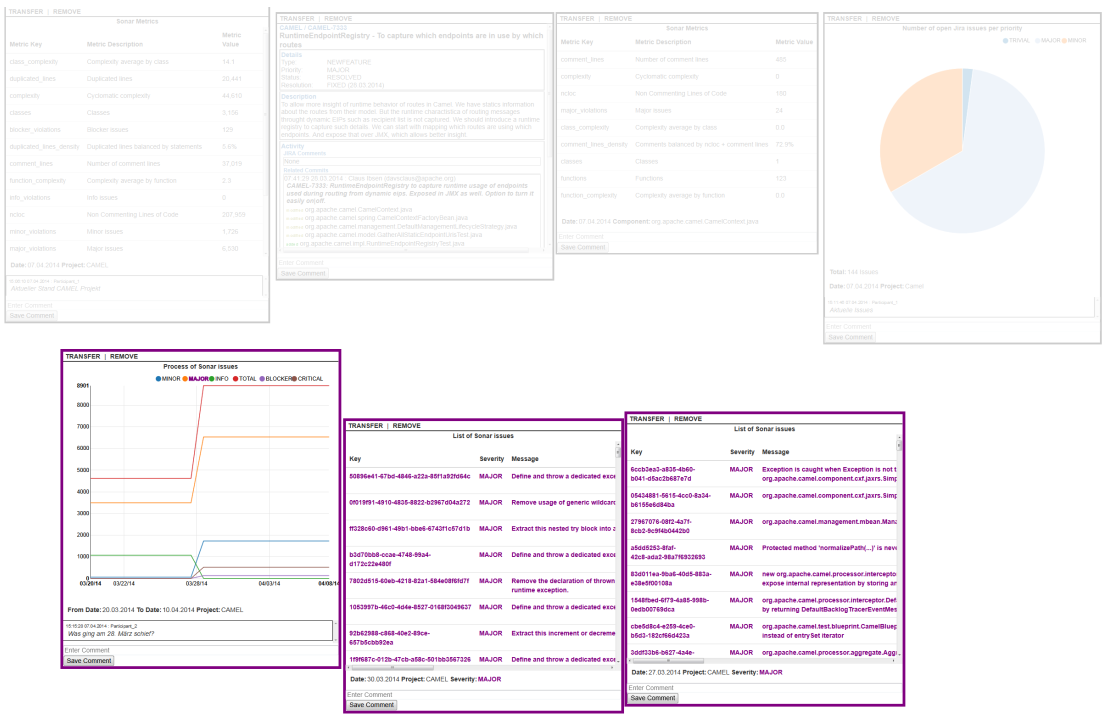
Part 1, Session 2. Group A of session two (GA2) started with querying the Jira details of both CRQs. Participant two was moving one of the corresponding DSs to the right corner of the working area. The time of this action correlates with the decision to split the CRQs, which can be found in the conversation log. Both participants queried the details of the commits that are related to their CRQ. As a next step, both loaded the code of a changed component. One participant started to add comments to the corresponding DS. Both participants queried the metrics of a changed component. The participants kept working for about 20 minutes without any direct communication. The rest of the time the group tried to organize the increased number of DSs on the working area in a sensible way, by means of the space organization and highlighting functions. The state of the resulting space is shown in Figure 5.7.

Part 2, Session 1. Group B of session one (GB1) - which worked with the result of GA1 - did not interact with the space very much. They removed one DS. The reason for this can not be identified. They moved around the DSs on the space occasionally. Possibly, in order to show each other, which DS is currently addressed. A moved DS is highlighted on the other participants screen. It is not possible to identify a direct correlation between their conversation and the interaction with the space.

Part 2, Session 2. Group B of session two (GB2) worked with the result of GB1. They started with the *organize space* operation and discussed all DSs one by one. They moved the DS that should be discussed next in a free area in the middle of the screen. When the discussion was done, a new DS was selected by one of the participants. They deleted DSs that were not needed anymore. During the process, they distributed sub-tasks. One was, e.g., querying the diff of a commit and checking if it introduced a specific Sonar issue.

Discussion

The history function appears to enable the comprehension of reviews, at least to a certain degree. Even with the rather limited data produced by this study, it was possible to identify different approaches of space usages. Nevertheless, without the conversation log, not all actions can be reconstructed. Many of the operations performed on the space - especially in step one - were most likely try and error learning of the application functions. With better communication support and more functions to annotate an DS, the history function should support the comprehension of an review very well.



Sonar Metrics

Metric Key	Metric Description	Metric Value
class_complexity	Complexity average by class	14.1
duplicated_lines	Duplicated lines	20,441
complexity	Cyclomatic complexity	44,610
classes	Classes	3,156
blocker_violations	Blocker issues	129
duplicated_lines_density	Duplicated lines balanced by statements	5.6%
comment_lines	Number of comment lines	37,019
function_complexity	Complexity average by function	2.3
info_violations	Info issues	0
ncloc	Non Commenting Lines of Code	207,959
minor_violations	Minor issues	1,726
major_violations	Major issues	6,530

Date: 07.04.2014 **Project:** CAMEL

15:06:10 07.04.2014 : Participant_1
Aktueller Stand CAMEL Projekt

CAMEL / CAMEL-7333
RuntimeEndpointRegistry - To capture which endpoints are in use by which routes

Details

Type:	NEWFEATURE
Priority:	MAJOR
Status:	RESOLVED
Resolution:	FIXED (28.03.2014)

Description
To allow more insight of runtime behavior of routes in Camel. We have statics information about the routes from their model. But the runtime charactistica of routing messages throught dynamic EIPs such as recipient list is not captured. We should introduce a runtime registry to capture such details. We can start with mapping which routes are using which endpoints. And expose that over JMX, which allows better insight.

Activity
JIRA Comments
None
Related Commits
07:41:29 28.03.2014 : Claus Ibsen (davsclaus@apache.org)
CAMEL-7333: RuntimeEndpointRegistry to capture runtime usage of endpoints used during routing from dynamic eips. Exposed in JMX as well. Option to turn it easily on|off.
modified org.apache.camel.CamelContext.java
modified org.apache.camel.spring.CamelContextFactoryBean.java
modified org.apache.camel.management.DefaultManagementLifecycleStrategy.java
modified org.apache.camel.model.GatherAllStaticEndpointUrisTest.java
added org.apache.camel.impl.RuntimeEndpointRegistryTest.java

Sonar Metrics

Metric Key	Metric Description	Metric Value
comment_lines	Number of comment lines	485
complexity	Cyclomatic complexity	0
ncloc	Non Commenting Lines of Code	180
major_violations	Major issues	24
class_complexity	Complexity average by class	0.0
comment_lines_density	Comments balanced by ncloc + comment lines	72.9%
classes	Classes	1
functions	Functions	123
function_complexity	Complexity average by function	0.0

Date: 07.04.2014 **Component:** org.apache.camel.CamelContext.java

Total: 144 Issues
Date: 07.04.2014 **Project:** Camel

15:11:46 07.04.2014 : Participant_1
Aktuelle Issues

From Date: 20.03.2014 **To Date:** 10.04.2014 **Project:** CAMEL

15:15:20 07.04.2014 : Participant_2
Was ging am 28. März schief?

List of Sonar issues

Key	Severity	Message
50896e41-67bd-4846-a22a-85f1a92fd64c	MAJOR	Define and throw a dedicated exce
0f019f91-4910-4835-8822-b2967d04a272	MAJOR	Remove usage of generic wildcard
ff328c60-d961-49b1-bbe6-6743f1c57d1b	MAJOR	Extract this nested try block into a
b3d70bb8-ccae-4748-99a4-d172c22e480f	MAJOR	Define and throw a dedicated exce
7802d515-60eb-4218-82a1-584e08f6fd7f	MAJOR	Remove the declaration of thrown runtime exception.
1053997b-46c0-4d4e-8527-0168f3049637	MAJOR	Define and throw a dedicated exce
92b62988-c868-40e2-89ce-657b5cbb92ea	MAJOR	Extract this increment or decreme
1f9f687c-012b-47cb-a58c-501bb3567326	MAJOR	Define and throw a dedicated exce

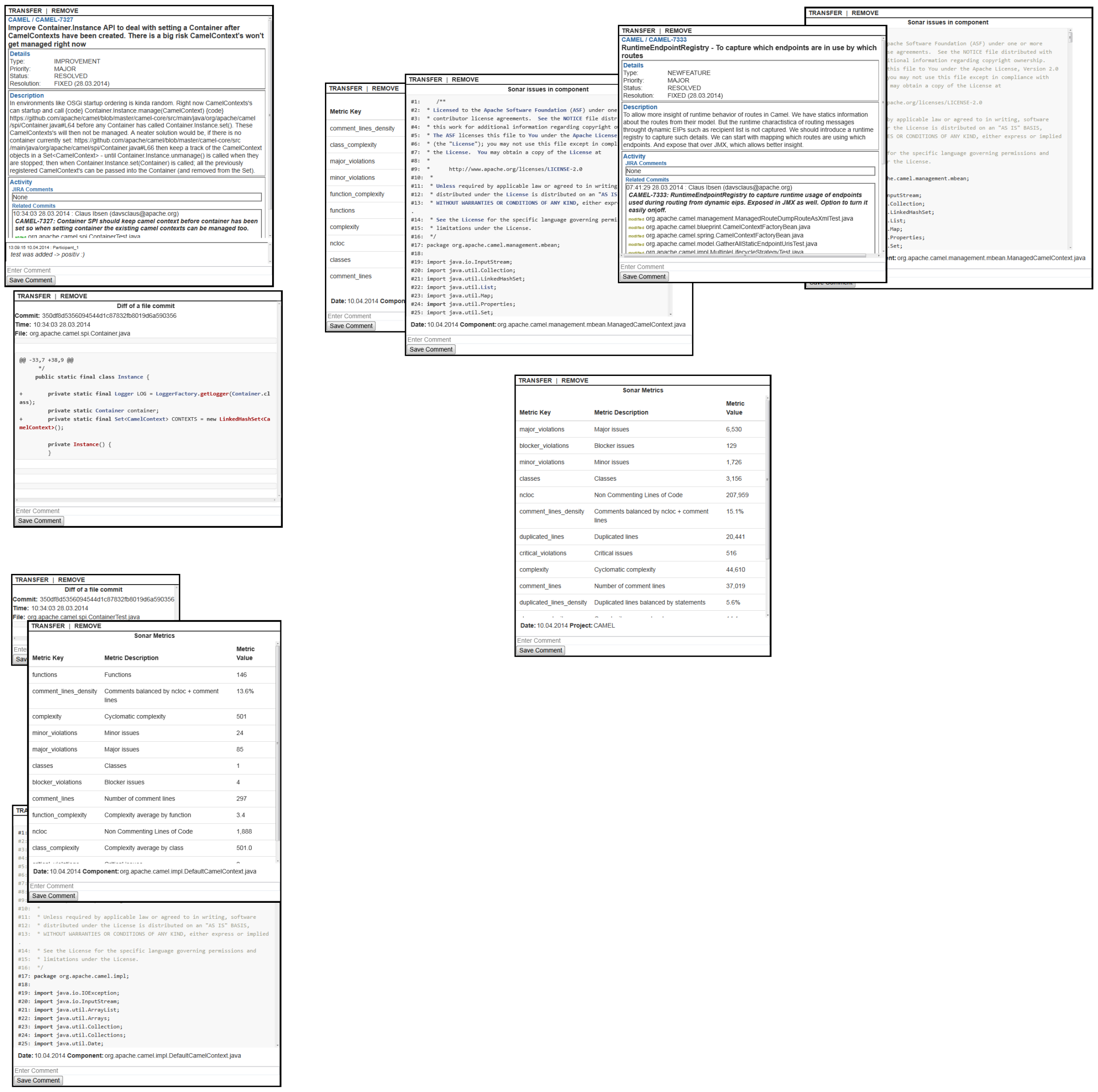
Date: 30.03.2014 **Project:** CAMEL **Severity:** MAJOR

List of Sonar issues

Key	Severity	Message
6ccb3ea3-a835-4b60-b041-d5ac2b687e7d	MAJOR	Exception is caught when Exception is not t org.apache.camel.component.cxf.jaxrs.Simp
05434881-5615-4cc0-8a34-b6155e6d84ba	MAJOR	org.apache.camel.component.cxf.jaxrs.Simp
27967076-08f2-4a7f-8cb2-9c9f4b0442b0	MAJOR	org.apache.camel.management.mbean.Mana
a5dd5253-8faf-42c8-ada2-98a7f6932693	MAJOR	Protected method 'normalizePath(...)' is neve
83d011ea-9ba6-40d5-883a-e38e5f00108a	MAJOR	new org.apache.camel.processor.intercepto expose internal representation by storing an
1548fbed-6f79-4a85-998b-0edb00769dca	MAJOR	org.apache.camel.processor.interceptor.Def by returning DefaultBacklogTracerEventMes
cbe5d8c4-e259-4ce0-b5d3-182cf66d423a	MAJOR	org.apache.camel.test.blueprint.CamelBluep instead of entrySet iterator
3ddf33b6-b627-4a4e-	MAJOR	org.apache.camel.processor.aggregate.Aggi

Date: 27.03.2014 **Project:** CAMEL **Severity:** MAJOR

Figure 5.6: **Result of group A in session one, part one.**



TRANSFER | REMOVE

CAMEL / CAMEL-7327

Improve Container.Instance API to deal with setting a Container after CamelContexts have been created. There is a big risk CamelContext's won't get managed right now

Details

Type:	IMPROVEMENT
Priority:	MAJOR
Status:	RESOLVED
Resolution:	FIXED (28.03.2014)

Description

In environments like OSGi startup ordering is kinda random. Right now CamelContexts's can startup and call {code} Container.Instance.manage(CamelContext) {code} https://github.com/apache/camel/blob/master/camel-core/src/main/java/org/apache/camel/spi/Container.java#L64 before any Container has called Container.Instance.set(). These CamelContexts's will then not be managed. A neater solution would be, if there is no container currently set: https://github.com/apache/camel/blob/master/camel-core/src/main/java/org/apache/camel/spi/Container.java#L66 then keep a track of the CamelContext objects in a Set<CamelContext> - until Container.Instance.unmanage() is called when they are stopped; then when Container.Instance.set(Container) is called; all the previously registered CamelContext's can be passed into the Container (and removed from the Set).

Activity

JIRA Comments

None

Related Commits

10:34:03 28.03.2014 : Claus Ibsen (davsclaus@apache.org)

CAMEL-7327: Container SPI should keep camel context before container has been set so when setting container the existing camel contexts can be managed too.

added org.apache.camel.spi.ContainerTest.java

13:09:15 10.04.2014 : Participant_1

test was added -> positiv :)

Enter Comment

Save Comment

TRANSFER | REMOVE

Diff of a file commit

Commit: 350df8d5356094544d1c87832fb8019d6a590356

Time: 10:34:03 28.03.2014

File: org.apache.camel.spi.Container.java

```
@@ -33,7 +38,9 @@
      */
     public static final class Instance {

+        private static final Logger LOG = LoggerFactory.getLogger(Container.cl
ass);
         private static Container container;
+        private static final Set<CamelContext> CONTEXTS = new LinkedHashSet<Ca
melContext>();

         private Instance() {
         }
```

Enter Comment

Save Comment

TRANSFER | REMOVE

Metric Key

comment_lines_density

class_complexity

major_violations

minor_violations

function_complexity

functions

complexity

ncloc

classes

comment_lines

Date: 10.04.2014 **Compon**

Enter Comment

Save Comment

TRANSFER | REMOVE

Sonar issues in component

```
   /**
 * Licensed to the Apache Software Foundation (ASF) under one
 * contributor license agreements.  See the NOTICE file distr
 * this work for additional information regarding copyright o
 * The ASF licenses this file to You under the Apache License
 * (the "License"); you may not use this file except in compl
 * the License.  You may obtain a copy of the License at
 *
 *      http://www.apache.org/licenses/LICENSE-2.0
 *
 * Unless required by applicable law or agreed to in writing
 * distributed under the License is distributed on an "AS IS
 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either expr
.
 * See the License for the specific language governing permi
 * limitations under the License.
 */
package org.apache.camel.management.mbean;

import java.io.InputStream;
import java.util.Collection;
import java.util.LinkedHashSet;
import java.util.List;
import java.util.Map;
import java.util.Properties;
import java.util.Set;
```

Date: 10.04.2014 **Component:** org.apache.camel.management.mbean.ManagedCamelContext.java

Enter Comment

Save Comment

TRANSFER | REMOVE

CAMEL / CAMEL-7333

RuntimeEndpointRegistry - To capture which endpoints are in use by which routes

Details

Type:	NEWFEATURE
Priority:	MAJOR
Status:	RESOLVED
Resolution:	FIXED (28.03.2014)

Description

To allow more insight of runtime behavior of routes in Camel. We have statics information about the routes from their model. But the runtime charactistica of routing messages throught dynamic EIPs such as recipient list is not captured. We should introduce a runtime registry to capture such details. We can start with mapping which routes are using which endpoints. And expose that over JMX, which allows better insight.

Activity

JIRA Comments

None

Related Commits

07:41:29 28.03.2014 : Claus Ibsen (davsclaus@apache.org)

CAMEL-7333: RuntimeEndpointRegistry to capture runtime usage of endpoints used during routing from dynamic eips. Exposed in JMX as well. Option to turn it easily on|off.

modified org.apache.camel.management.ManagedRouteDumpRouteAsXmlTest.java

modified org.apache.camel.blueprint.CamelContextFactoryBean.java

modified org.apache.camel.spring.CamelContextFactoryBean.java

modified org.apache.camel.model.GatherAllStaticEndpointUrisTest.java

modified org.apache.camel.impl.MultipleLifecycleStrategyTest.java

Enter Comment

Save Comment

TRANSFER | REMOVE

Sonar issues in component

```
pache Software Foundation (ASF) under one or more
se agreements.  See the NOTICE file distributed with
itional information regarding copyright ownership.
this file to You under the Apache License, Version 2.0
you may not use this file except in compliance with
 may obtain a copy of the License at

pache.org/licenses/LICENSE-2.0

by applicable law or agreed to in writing, software
r the License is distributed on an "AS IS" BASIS,
ES OR CONDITIONS OF ANY KIND, either express or implied

for the specific language governing permissions and
r the License.

he.camel.management.mbean;

nputStream;
.Collection;
.LinkedHashSet;
.List;
.Map;
.Properties;
.Set;
```

nt: org.apache.camel.management.mbean.ManagedCamelContext.java

TRANSFER | REMOVE

Sonar Metrics

Metric Key	Metric Description	Metric Value
major_violations	Major issues	6,530
blocker_violations	Blocker issues	129
minor_violations	Minor issues	1,726
classes	Classes	3,156
ncloc	Non Commenting Lines of Code	207,959
comment_lines_density	Comments balanced by ncloc + comment lines	15.1%
duplicated_lines	Duplicated lines	20,441
critical_violations	Critical issues	516
complexity	Cyclomatic complexity	44,610
comment_lines	Number of comment lines	37,019
duplicated_lines_density	Duplicated lines balanced by statements	5.6%

Date: 10.04.2014 **Project:** CAMEL

Enter Comment

Save Comment

TRANSFER | REMOVE

Diff of a file commit

Commit: 350df8d5356094544d1c87832fb8019d6a590356

Time: 10:34:03 28.03.2014

File: org.apache.camel.spi.ContainerTest.java

TRANSFER | REMOVE

Sonar Metrics

Metric Key	Metric Description	Metric Value
functions	Functions	146
comment_lines_density	Comments balanced by ncloc + comment lines	13.6%
complexity	Cyclomatic complexity	501
minor_violations	Minor issues	24
major_violations	Major issues	85
classes	Classes	1
blocker_violations	Blocker issues	4
comment_lines	Number of comment lines	297
function_complexity	Complexity average by function	3.4
ncloc	Non Commenting Lines of Code	1,888
class_complexity	Complexity average by class	501.0

Date: 10.04.2014 **Component:** org.apache.camel.impl.DefaultCamelContext.java

Enter Comment

Save Comment

```
 *
 * Unless required by applicable law or agreed to in writing, software
 * distributed under the License is distributed on an "AS IS" BASIS,
 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied
.
 * See the License for the specific language governing permissions and
 * limitations under the License.
 */
package org.apache.camel.impl;

import java.io.IOException;
import java.io.InputStream;
import java.util.ArrayList;
import java.util.Arrays;
import java.util.Collection;
import java.util.Collections;
import java.util.Date;
```

Date: 10.04.2014 **Component:** org.apache.camel.impl.DefaultCamelContext.java

Enter Comment

Save Comment

Figure 5.7: Result of group A in session two, part one.

5.4 Final Discussion

The data so far does not allow for statements regarding the benefits of the approach of SQAC in a collaborative SQR. This is due to the limited number of participants and the very open design of the study.

Nevertheless, valuable user feedback has been collected. By the questionnaires as well as by analyzing the general SQAC usage. Based on the gathered data, significant improvements of SQAC should be possible.

Furthermore, some important implications for future studies could be identified. A noticeable fact is the rather low level of collaboration of all groups, but one. This can be influenced by two factors. First, certain problems in the collaboration support have been identified by the feedback. It is possible, that this decreased the motivation to collaborate in the tasks. However, this should only be a factor of SQAC, which is not the case. Furthermore, according to the conversation log, the groups defined their approach in the first minutes. A second factor could be a general lack of incentives for collaboration. The collaboration scenarios introduced in Section 1.1 included clear incentives for collaboration. Either, the knowledge of other participants is needed or specific responsibilities enforce the collaboration. For the participants of the study, neither was true. In terms of productivity, the distribution of the tasks was maybe the most economic decision.

The study was focused on the review process and not the review results. The tasks were primary a vehicle to frame the evaluation by a sensible and motivating work scenario. The results of the groups SQAC and SOTA in part 1 are not comparable, due to the general lack of constraints and the fundamentally different format. The results of part 2 are more structured, but the very open task description allows not for a clear specification of measurable variables.

It is possible, that the chosen scenario would be too complicated for a study that is focusing on comparable review results. An SQR itself is a complicated task, since different kinds of data must be considered. The creation of a sensible and well structured result document is challenging, too. In the study, both aspects were combined with distributed real-time collaboration and hence the complexity was increased even more.

Further studies should decrease the complexity of the scenario and add more structure to the general work process. More constraints are needed in order to produce results that can be compared. Different features of SQAC should be examined separately. After the identified improvements have been implemented, a first study could address the general usability of the tool in a single user scenario. The result transfer and the optimal structure of a result space could be studied in such a scenario without the interference of collaboration problems. Afterwards, shared space usage could be added in order to evaluate the collaboration support. It should be combined with real collaboration incentives. The creation of such could be challenging if no participants are available that worked together at a suited project. Incentives could be created by giving handouts with specific knowledge to the participants during the introduction. Such a handout could contain information about commits which have no CRQ key in their messages. Without the help of the informed participant, the rationals of the commits could not be identified by the other participants. To measure the amount of collaboration and knowledge transfer, one could examine if all participants are at the same knowledge level at the end of study. This could be done by asking them questions about information only explicitly given to other participants.

Chapter 6

Final Remarks

6.1 Conclusion

The goal of this thesis was to support different aspects of distributed collaborative SQRs. Review participants should be provided with a shared view on the necessary data. They should be enabled to explore the data as well collaboratively as individually. The creation and the transfer of the review results should be eased. The comprehension of the results and the analysis of the review process should be supported. The web application SQAC has been developed to support SQRs in the described way.

SQAC is an SQR groupware for collaboration-oriented synchronous work. It provides the users with multiple shared spaces with a working area comparable to a whiteboard. Queries can be executed by the users to analyze the underlying data. The repositories Jira, Github and Sonar are supported and their data can be imported in a central DST. An ontology based semantic model is used to store and integrate the repository data. Queries create DSs which will be displayed on the working area. An example for a DS is a pie chart showing the number of Jira issues per priority. Users can arranged DSs on the working area and comment on them. Each DS contains the semantic information of its values. This information is used to highlight related DSs and recommend queries. It is possible to transfer DSs between different spaces. Each operation of a user is mapped to a specific event. All events are stored in a history. By means of this history, users can step through the previous states of the space similar to a slideshow of a presentation.

The shared spaces provide users with a shared view. Collaborative and individual data exploration is supported. Multiple users can execute queries on the same space. Furthermore, users can transfer DSs to different spaces in order to analyze them individually. The final state of a space can be regarded as the review result. Hence, the result document is created by arranging and commenting the DSs. DSs can be organized and relations between them can be highlighted. Additionally, the event history allows to reconstruct the operations that have been conducted on the space. This supports the comprehension of the review results.

The project was evaluated by an exploratory study. Valuable user feedback was generated and areas of necessary improvements identified. Further studies should address the different supported aspects of distributed collaborative SQRs and investigate if SQAC introduces benefits in comparison to alternative approaches.

6.2 Future Work

6.2.1 Support of Further Repositories

The currently included repositories primary support the review of technical aspects of SQ. As discussed in Section 2.1, SQ has multiple aspects and can be analyzed from different points of view. Future versions of SQAC should support further repositories in order to allow for a comprehensive SQR. For example, parts of specifications like user stories or requirements could be imported. The DS concept is flexible enough to represent very different kinds of data. Additionally, more of the data provided by Jira should be utilized. Depending on the Jira configuration and usage, it can contain information as time schedules and story points. This information could be used to investigated the state of the project and the development process.

6.2.2 Review Process Support

As already addressed in Section 3.3, no structured quality review process is currently provided by SQAC. With its spaces and the DSVP, the application offers a potential useful tool, but no guidance how to use it. It could be a topic of future research, how to use the basic concept of the application in order to implement structured review processes. A possible way of doing this is based on different STs. Spaces of different types could be used for different steps in a review process. These spaces would allow different operations depending on the goal of the review step. Furthermore, DSs could be rendered differently, depending on the current process step. The following example should clarify the idea:

Three spaces are used. The first space is used for the gathering of data. The second space is used to prioritize the data. The third space is used to discuss the data in an order that is corresponding to its priority. The first space allows for the execution of queries and commenting of DSs, just like the DSVP. Users can push interesting DSs to the second space. The second space does not allow for the execution of queries and modification of DSs. It provides a read only view on the DSs and the possibility to rate the relevance of DSs. All participants must rate all DSs. When all DSs are rated, the results are pushed to a third space. It will present the DSs in the order of their rating. The third space could automatically query related data based on the priority, semantics and relations of the DSs. The described process is similar to the one implemented by [JT93], which has been described in Section 2.1. Such a process could be built with the tools provided by SQAC.

6.2.3 Role Based Views

The same space can be represented in different ways using different SVTs. This possibility could be utilized to build role specific SVTs. It could be a topic of future research, if such SVTs can help to increase the ease of reviews and the quality of the results. A simple example for a role based SVT is the following: A space supports one SV for testers and one for developers. Both SVs render the basic DS in the same way, but show additional role specific information. Both roles could see a DS containing the source code of a component. The developer would additionally see information about the code complexity. Whereas the tester would see information about the testing results of this component. Furthermore, SV could not be bound to specific user roles, but represent ways of looking at specific characteristics of the data. Users could switch between different SVs if required.

6.2.4 Mobile Device Support

Since most modern smart phones provide a web browser, SQAC could be used via such devices. Due to the screen size limitations of smart phones, the application would not be very useful yet. The development of an SVT specific for mobile device could be a part of future work. Such support could increase the usefulness of the tool significantly. Smart phone support could be especially useful for stakeholders who missed a review session and want to take a look on the results on a train. This use case would probably introduce the need for an off-line mode, due to the likely connection losses during train travel. Such a mode would introduce a lot of technical interesting problems like the synchronization between the off-line version and the possibly different on-line version of a space.

6.2.5 Extended Workspace Organization

As addressed in Section 3.3, the current possibilities of space organization have limitations. The semantic highlighting is limited to on kind of value at a time. The overview will not increase much, if very much DSs are on a space. Hence, it should be investigated how the existing semantic information could be utilized in order to provide better and more powerful ways of space organization. For example, one could think of some kind of semantic overview map. It could give a high level view of the DS semantics and the relation between the DSs. Such a map could omit the actual DSs and focus only on the semantic information and the relations.

6.2.6 Extended Query Recommendations

The current query recommendation mechanism is suggesting queries for the values of a DS. This is based on the semantic of the values. In future work it could be investigated, how the existing semantic information can be used to create more powerful recommender. Such a recommendation service could take the complete space data into account and not just on DS. It could be attempted to put the DSs in a semantic relation to each other and suggest queries that are sensible in the context of the current space state. Of course, what is sensible for which kind of DSs would be a question to ask first. The research field of information-needs could help to answer this question. The behavior of other users could be analyzed and used for the recommendations (see also Section 6.2.7). An extended and more powerful query recommendation service could help users with little experience in SQA.

6.2.7 Interaction Pattern Identification

An interesting topic for future research would be the analysis of user behavior. This topic is related to the third use case that has been introduced in Section 1.1. It could be possible to identify specific patterns in the interaction between users and a space. It would be interesting to investigate, if a best practice of analysis steps can be identified. Such knowledge could be used to extend the query recommendation or provide general recommendations for operations on a space. One could think about a space, that is performing a pattern of operations on its own. This could happen during the initialization to ease the review start. Alternatively, the automatic execution of specific operation patterns could be suggested when the space is in a matching state. For example, if a Jira issue is displayed, it could be recommended to load the Sonar metrics for all files that have been changed in the context of this issue. It is yet uncertain if it is possible to identify interaction patterns and to utilize them. Even though, the event based design and the already implemented event history should provide a solid basis for further work in this area.

Appendix A

Handout Study Part 1

Part 1

1. Introduction

You and your group colleague are supposed to collaboratively review a number of improvements in the apache camel project. Your goal is to assess the influence of these improvements to the overall software quality of the project (e.g. test coverage, number of quality Issues, maintainability etc.). The results of your assessment will be used by another group to identify refactoring steps to increase the quality of the software.

You will have 30 minutes for the quality review. Please read the following instructions.

2. Instructions and Preparations

You will need Skype for the survey. Please install Skype and add your group colleague to your contact list. **You are asked to provide your Skype conversation protocol to us at the end of the survey. Please keep this in mind during the survey and avoid the discussion of private topics.**

2.1 Group specific instructions and preparations

Group A

You are asked to use Firefox and the tool SQA-Collaboration for this survey.

- Open http://seal-students.ifi.uzh.ch:40000/
- Login as "Participant_<YOUR_PARTICIPANT_NUMBER>"

There is a workspace named "Study_Space"

Please join this workspace and use it for the quality review. When you think you are done or when the 30 minutes are over, please just leave the space by closing the corresponding browser tab. The end state of the workspace is the review result. In the second part of the study, another group will join the workspace in order to check your results and work with them.

You and your colleague are allowed to communicate only via Skype and the SQA-Collaboration application.

Group B

You and your colleague are supposed to use Jira, Github and SonarQube for the quality review.

- https://github.com/apache/camel/commits/master
- https://issues.apache.org/jira/browse/CAMEL
- http://nemo.sonarqube.org/dashboard/index/146590

You have been granted access to a GoogleDocs document named "Part1_Result". This document should contain the results of your quality review at the end of your review. In the second part of this study, another group will get access to the document in order to check your results and work with them.

You and your colleague are supposed to communicate only via Skype and the GoogleDocs document.

2.2 Common instructions and preparations

At the end of this part of the survey please provide your Skype protocol to us. Only one log per group is necessary. In order to do this, please perform the following steps:

- Open a new document in a text editor and copy your Skype protocol in the document
  - If available, please use an editor which is able to save plain text documents, e.g. Notepad++, Vi, Vim or Emacs
- Replace your user name with "Participant_<YOUR_PARTICIPANT_NUMBER>"
- Replace your colleagues user name with "Participant_<COLLEAGUE_PARTICIPANT_NUMBER>"
- Save the document as "Skype_Group_<YOUR_GROUP>_Part1.txt"
- Copy this file on the provided USB stick.

Appendix B

Handout Study Part 2

Part 2

1. Introduction

You and your group colleague are supposed to review the results of a software quality review. You are asked to check the plausibility of issues documented in the original review and suggest possible refactorings to increase the quality of the software.

2. Instructions and Preparations

You will need Skype for the survey. Please install Skype and add your group colleague to your contact list. **You are asked to provide your Skype conversation protocol to us at the end of the survey. Please keep this in mind during the survey and avoid the discussion of private topics.**

2.1 Group specific instructions and preparations

Group B

You are asked to use Firefox and the tool SQA-Collaboration for this survey.

- Open http://seal-students.ifi.uzh.ch:40000/
- Login as "Participant_<YOUR_PARTICIPANT_NUMBER>"

There is a workspace named "Study_Space".

Please join this workspace and use it for the review. The space contains the results of the quality review you are supposed to review. When you think you are done or when the 30 minutes are over, please just leave the space by closing the corresponding browser tab.

You and your colleague are allowed to communicate only via Skype and the SQA-Collaboration application.

Group A

You and your colleague are supposed to use Jira, Github and SonarQube for the quality review.

- https://github.com/apache/camel/commits/master
- https://issues.apache.org/jira/browse/CAMEL
- http://nemo.sonarqube.org/dashboard/index/146590

You have been granted access to a GoogleDocs document named "Part1_Result". This document contains the results of the quality review you are supposed to review.
You and your colleague are supposed to communicate via Skype and the GoogleDocs document only.

2.2 Common instructions and preparations

You have been granted access to a GoogleDocs document named "Group_<YOUR_GROUP>_Results_Part2". Please fill the rows. This document will be the result of the study. The rows should contain the following information:

Issue: One issue you could identify. This includes issues named in the original review you don't agree with.
Severity: The severity of this issue. Minor, Major, Critical, No Issue
Resolution: What would you do (or not) in order to resolve this issue?
Comment: You can enter additional comments about the issue.

At the end of the survey please provide your Skype protocol to us. Only one log per group is necessary. In order to do this, please perform the following steps:

- Open a new document in a text editor and copy your Skype protocol in the document
    - If available, please use an editor which is able to save plain text documents, f.e. Notepad++, Vi, Vim or Emacs
- Replace your user name with "Participant_<YOUR_PARTICIPANT_NUMBER>"
- Replace your colleagues user name with "Participant_<COLLEAGUE_PARTICIPANT_NUMBER>"
- Save the document as "Skype_Group_<YOUR_GROUP>_Part2.txt"
- Copy this file on the provided USB stick.

Appendix C

Self-Assessment Questionnaire

You are in Group	____
Your participant number is	____
Your colleagues participant number is	____
Assigned Jira Improvements for Part 1	____

Please answer the following questions.

How many years of software development experience (any language) do you have?

______ Years

How many years of professional (Internship, Employment, Self-Employment etc.) development experience do you have?

______ Years

How do you rate your knowledge of Jira on a scale from 0 to 3? Where 0 means "no experience whatsoever" and 3 "intensive usage in a software project".

[0] [1] [2] [3]

How do you rate your knowledge of Github on a scale from 0 to 3? Where 0 means "no experience whatsoever" and 3 "intensive usage in a software project".

[0] [1] [2] [3]

How do you rate your knowledge of SonarQube on a scale from 0 to 3? Where 0 means "no experience whatsoever" and 3 "intensive usage in a software project".

[0] [1] [2] [3]

How do you rate your knowledge about the analysis of software quality on a scale from 1 to 4 ? Where 1 means "Attended one related lecture" and 4 "Extended theoretical knowledge and practical experience".

[1] [2] [3] [4]

Please enter the following information.

Start time Part 1 __________	Stop time Part 1 ____________
Start time Part 2 __________	Stop time Part 2 ____________

Appendix D

Feedback Questionnaire

Please answer this question after you attended to both parts of the study.

What did you like about the tool SQA-Collaboration? What was useful?

What did you NOT like about the tool SQA-Collaboration? What is missing?

For a task like the one of Part 1, would you prefer SQA-Collaboration or Sonar, Github and Jira? Why?

For a task like the one of Part 2, would you prefer SQA-Collaboration or Sonar, Github and Jira? Why?

Appendix E

Feedback Questionnaire Answers

Participants are referenced as follows: *<PARTICIPANTNUMBER><GROUP><SESSION>*. Hence, *1A1* reads as *Participant with number 1 of group A in session 1*

Question 1

What did you like about the tool SQA-Collaboration? What was useful?

1A1: Navigation between tools is easy; Commenting on data sets: comments are exactly where they are relevant; Outlining and highlighting of related data sets;

2A1: Shared working area; Shared work artifacts; Possibility to tidy the working area;

3B1: Nice concept; Data available in one place; Nice possibilities to browse through linked data;

4B1: Have one tool instead of multiple different tools;

1A2: The linkage and the navigation between the data;

2A2: Easy to organize different data sets by placing them next to each other; Placement can be used to mark categories or what belongs together; Information from different tools together to allow easy reasoning and sharing; Easy to extract additional info from one view;

3B2: It is very handy, to have all existing data in the same view;

4B2: To see the information of different tools by touching a button; Go deeper into the details;

Question 2

What did you NOT like about the tool SQA-Collaboration? What is missing?

1A1: It is difficult to see where operations / links are available; No feedback on some actions, e.g. loading data in progress; Small space on screen; Moving / creating / deleting data sets is confusing for the other collaborator; Talking about 'the window in the bottom left corner' is difficult, maybe a pointer would be nice;

2A1: Sometimes, feedback from the user interface is missing; It is not possible to re-size the data sets; To much information accessible for developer without experience with Sonar, Jira and Jenkins; Sometimes it is not obvious, what data is needed for a query; No time-line view;

3B1: Re-sizing of data sets not possible; Include a workflow and guide the usage somehow; Better overview of what data is available; Include a chat, would make sense to me;

4B1: Perhaps, to really see what the other one is doing, in order to prevent removing something the other is viewing;

1A2: Overview of the project and the available data is missing; It is possible that necessary information are hidden by the highlighting function, if multiple persons working with the same workspace;

2A2: Multiple users at the same time interfere with each other and sometimes delete other users' work; Feedback about the success or failure of the creation of new data sets is not visible without scrolling;

3B2: Focus switching between SQAC and Skype is difficult; A possibility to mark a data set during a discussion is missing; It is difficult, to comprehend the trains of thought of group A;

4B2: It is very difficult to coordinate the communication if the screen is full of data sets; It is not clear, where a data set is coming from and which query did create it;

Question 3

For a task like the one of Part 1, would you prefer SQA-Collaboration or Sonar, Github and Jira? Why?

1A1: No opinion, not enough experience with either;

2A1: Sonar, Github, Jira: The time-line view is very important for such task. It is missing in SQAC.

3B1: SQAC, since it was really hard fitting all the pieces together, especially with Sonar.

4B1: I think Sonar because you can analyze down to the code file. Even though we did not do it because we noticed to late that it would be useful.

1A2: SQAC would be more comfortable, if the navigation at the beginning would be easier. Hence, for example by means of a overview graphic of the complete project. At the moment, the overview is missing.

2A2: Depends on the information seeked. Sonar is maybe easier if only one thing for myself is seeked, otherwise indifferent. If sharing / collaboration is needed, definitive SQAC. Also, if knowledge from multiple persons is needed.

3B2: SQAC;

4B2: SQAC, because all of the data is available in the same view and no tool switching is necessary.

Question 4

For a task like the one of Part 2, would you prefer SQA-Collaboration or Sonar, Github and Jira? Why?

1A1: SQAC, since it is easier to establish the line of thinking / context if you can take over the workspace;

2A1: SQAC, because it allows to restore the used artifact / the used view and the identified issues can be comprehended;

3B1: This depends on how it was done; If it is nicely documented, either tool would work; However, I can imagine SQAC to be faster;

4B2: SQAC because you see better what the other persons really did / what data they used;

1A2: SQAC, since the data is provided in a collected manner;

2A2: SQAC, because it gives an easy overview and allows to easily follow and understand the thoughts of another user, through the comments, the arrangement etc.;

3B2: SQAC;

4B2: SQAC, if a better way is possible to follow the train of thoughts of group A;

Appendix F

Contents of the CD-ROM

Zusfsg.txt	German version of the abstract of this thesis
Abstract.txt	English version of the abstract of this thesis
Masterarbeit.pdf	Copy of this thesis

Bibliography

[Bak97] Richard A. Baker, Jr. Code reviews enhance software quality. In *Proceedings of the 19th International Conference on Software Engineering*, ICSE '97, pages 570–571, New York, NY, USA, 1997. ACM.

[BC12] Amiangshu Bosu and Jeffrey C. Carver. Peer code review in open source communitiesusing reviewboard. In *Proceedings of the ACM 4th Annual Workshop on Evaluation and Usability of Programming Languages and Tools*, PLATEAU '12, pages 17–24, New York, NY, USA, 2012. ACM.

[BL13] Michael Bevilacqua-Linn. *Functional Programming Patterns in Scala and Clojure: Write Lean Programs for the JVM.* O'Reilly, USA, 2013.

[BLFM98] T. Berners-Lee, R. Fielding, and L. Masinter. Uniform resource identifiers (uri): Generic syntax, 1998.

[Che10] Chaomei Chen. Information visualization. *Wiley Interdisciplinary Reviews: Computational Statistics*, 2(4):387–403, 2010.

[DFAB93] Alan Dix, Janet Finlay, Gregory Abowd, and Russell Beale. *Human-Computer Interaction.* 1993.

[Gar84] David Garvin. What does product quality really mean? *Sloan Management Review*, 26:25–45, 1984.

[GHJV94] Erich Gamma, Richard Helm, Ralph Johnson, and John M. Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software.* Addison-Wesley Professional, 1994.

[Gos11] Debasish Gosh. *DSLs in Action.* Manning Publications Co., Stamford, USA, 2011.

[Gru93] Thomas R. Gruber. A translation approach to portable ontology specifications. *Knowl. Acquis.*, 5(2):199–220, June 1993.

[Gru94] Jonathan Grudin. Computer-supported cooperative work: History and focus. *IEEE Computer*, 27(5):19–26, 1994.

[HBC14] Peter Hilton, Erik Bakker, and Francisco Canedo. *Play for Scala: Covers Play 2.* Manning Publications Co., New York, NY, USA, 2014.

[Joh88] Robert Johansen. *Groupware: Computer Support for Business Teams.* The Free Press, New York, 1988.

[JT93] Philip M. Johnson and Danu Tjahjono. Improving software quality through computer supported collaborative review. In *Proceedings of the Third Conference on European Conference on Computer-Supported Cooperative Work*, ECSCW'93, pages 61–76, Norwell, MA, USA, 1993. Kluwer Academic Publishers.

[KC04] Graham Klyne and Jeremy J. Carroll. Resource description framework (RDF): Concepts and abstract syntax. World Wide Web Consortium, Recommendation REC-rdf-concepts-20040210, February 2004.

[KP96] B. Kitchenham and S.L. Pfleeger. Software quality: the elusive target [special issues section]. *Software, IEEE*, 13(1):12–21, Jan 1996.

[MF13] S.C. Muller and T. Fritz. Stakeholders' information needs for artifacts and their dependencies in a real world context. In *Software Maintenance (ICSM), 2013 29th IEEE International Conference on*, pages 290–299, Sept 2013.

[MWFG12] Sebastian Müller, Michael Würsch, Thomas Fritz, and Harald Gall. An approach for collaborative code reviews using multi-touch technology. In *5th International Workshop on Cooperative and Human Aspects of Software Engineering (CHASE 2012)*, JUN 2012.

[NNKR09] Mehrdad Nurolahzade, Seyed Mehdi Nasehi, Shahedul Huq Khandkar, and Shreya Rawal. The role of patch review in software evolution: An analysis of the mozilla firefox. In *Proceedings of the Joint International and Annual ERCIM Workshops on Principles of Software Evolution (IWPSE) and Software Evolution (Evol) Workshops*, IWPSE-Evol '09, pages 9–18, New York, NY, USA, 2009. ACM.

[Nor02] Donald A. Norman. *The Design of Everyday Things*. Basic Books, New York, reprint paperback edition, 2002.

[OSV08] Martin Odersky, Lex Spoon, and Bill Venners. Programming in scala: A comprehensive step-by-step guide, 2008.

[Ras00] Jef Raskin. *The Humane Interface: New Directions for Designing Interactive Systems*. ACM Press/Addison-Wesley Publishing Co., New York, NY, USA, 2000.

[Rem05] J. Remillard. Source code review systems. *Software, IEEE*, 22(1):74–77, Jan 2005.

[RG06] Peter C. Rigby and Daniel M. German. A preliminary examination of code review processes in open source projects. Technical Report DCS-305-IR, University of Victoria, January 2006.

[RL02] I. Rus and M. Lindvall. Knowledge management in software engineering. *Software, IEEE*, 19(3):26–38, May 2002.

[Shn96] B. Shneiderman. The eyes have it: a task by data type taxonomy for information visualizations. In *Visual Languages, 1996. Proceedings., IEEE Symposium on*, pages 336–343, Sep 1996.

[SSEB10] F. Salger, S. Sauer, G. Engels, and A. Baumann. Knowledge transfer in global software development - leveraging ontologies, tools and assessments. In *Global Software Engineering (ICGSE), 2010 5th IEEE International Conference on*, pages 336–341, Aug 2010.

[SSU01] Gerhard Schwabe, Norbert Streitz, and Rainer Unland. *CSCW-Kompendium: Lehr- und Handbuch zum computerunterstützen kooperativen Arbeiten*. 2001.

[TSMB95] Stefanie Teufel, Christian Sauter, Thomas Mühlherr, and Kurt Bauknecht. *Computerunterstützung für die Gruppenarbeit*. 1995.

[Tuf86] Edward R. Tufte. *The Visual Display of Quantitative Information*. Graphics Press, Cheshire, CT, USA, 1986.

[War04] Colin Ware. *Information Visualization: Perception for Design*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2004.

[WGG13] Michael Würsch, Emanuel Giger, and Harald Gall. Evaluating a query framework for software evolution data. *ACM Transactions on Software Engineering and Methodology*, 22(4):38–38, 2013.

[WGH$^+$12] Michael Würsch, Giacomo Ghezzi, Matthias Hert, Gerald Reif, and Harald Gall. Seon: A pyramid of ontologies for software evolution and its applications. *Computing*, 94(11):857–885, 2012.

[WGRG10] Michael Würsch, Giacomo Ghezzi, Gerald Reif, and Harald Gall. Supporting developers with natural language queries. In *32nd ACM/IEEE International Conference on Software Engineering*, pages 165–174, MAY 2010.

[WRDG10] Michael Würsch, Gerald Reif, Serge Demeyer, and Harald Gall. Fostering synergies - how semantic web technology could influence software repositories. In *2nd International Workshop on Search-driven Development: Users, Infrastructure, Tools and Evaluation*, pages 45–48, MAY 2010.