Methods towards API Usability: A Structural Analysis of Usability Problem Categories

Thomas Grill, Ondrej Polacek, Manfred Tscheligi

Methods Towards API Usability: A Structural Analysis of Usability Problem Categories

Thomas Grill, Ondrej Polacek, Manfred Tscheligi

ICT&S Center, University of Salzburg
Salzburg, Austria
[firstname.secondname]@sbg.ac.at

Abstract. The usability of Application Programming Interfaces (APIs) is one of the main factors defining the success of a software based framework. Research in the area of human computer interaction (HCI) currently mainly focuses on end-user usability and only little research has been done regarding the usability of APIs. In this paper, we present a methodology on how to use and combine HCI methods with the goal to evaluate the usability of APIs. The methodology consist of three phases – a heuristic evaluation, a developer workshop and interviews. We setup a case-study according to the methodology, in which we are evaluating the usability of a service-oriented framework API. The goal was to explore different HCI methods and compare the applicability of such methods to find usability problems in an API. The case-study combined qualitative and quantitative methods in order to investigate the usability and intuitiveness of the API itself. It allowed us to identify relevant problem areas for usability related issues that could be mapped to specific types of HCI methods. Examples for this are e.g. structural problems, which are identified mainly in inspection methods, while problems regarding errors and exception handling are mainly identified during the hands-on example part of the developer workshops conducted. The resulting problem areas allow us to develop a first classification of API related usability problems that are making the relevancy of usability issues for APIs more explicit and applicable.

Keywords: API, Usability, Contextual Interaction Framework, HCI.

1 Introduction

Research in Human Computer Interaction (HCI) focuses mainly on the usability and user experience of hardware and software user interfaces. Little research has been done in methods and tools for software developers, especially when it comes to complex frameworks based on a service-oriented architecture (SOA). The developers use Application Programming Interfaces (APIs) to build rich applications with various functionalities. API is a set of reusable components such as objects, routines, or variables that provides a specific functionality. When it comes to measuring usability of an API existing methods from cognitive sciences

are usually applied. Such methods are typical user tests where an end-user has to conduct a specific task or inspection methods, where experts evaluate systems based on given heuristics. Regarding the usability of APIs a number of guidelines and heuristics exist that can be used for heuristic evaluations. Such guidelines are for example elaborated by Joshua Bloch, a software engineer at Google. In his paper "How to Design a Good API and Why it Matters" he discusses insights into requirements of a good and usable API [1]. Other guidelines are given by Scaffidi [2] where he discusses a number of challenges regarding the usability of APIs. All these guidelines have been used and applied in evaluation methods stemming from the area of human computer interaction (HCI). The existing guidelines can be mapped to specific usability attributes. For example, learnability can be linked to documentation, intuitiveness to structure, or user satisfaction to future reuse. This leads to the fact that not only heuristics but also concrete measurable evaluation criteria can be extracted. We address this in the form of a workshop that combines a tutorial or course-like setup with an evaluation approach. This evaluation approach uses methods like questionnaires, think-aloud protocol, video-taking, and interviews. This results in a combined methodology providing us with deeper insight about the usability problems of an API while investigating the applicability of our approach to a SOA-based API. The findings obtained during the study allowed us to do a categorization of the problems based on the problem area targeted by the finding. It also allowed us to map the resulting outputs to particular usability methods applied in our case study in order to highlight appropriate method for different problem areas and to show how the case study covers these areas.

2 Related Work

In the last decade a huge effort has been made to distill common usability flaws within an API. Conducting expert evaluation and user studies have led to development of several design guidelines as well as methodologies for API evaluation. The current methodologies for API evaluation can be divided into two groups according to involvement of users: user studies and expert evaluations.

Empirical studies with real users provide a deep understanding of usability flaws in an API. However, they are hardly suitable for testing large APIs with hundreds of elements because of their high costs. According to Farooq et al. [3] other limitations are difficult recruitment of the users with specific domain knowledge, time demands, and rather slow feedback. One of the first of API user studies was conducted by McLellan et al. [4]. In this study an application example (approx. 2.300 lines) containing the studied API calls was shown to the participants, who were asked to go through it and understand it. The participants were allowed to ask questions. After the test the API was redesigned based on these questions. A quantitative approach was used in the work by Robillard [5]. A questionnaire-based survey with 83 developers identified 11 categories of API learning obstacles

that can be divided into five groups: *Resources*, *Structure*, *Background*, *Technical environment*, and *Process*.

The three studies described in the following paragraph focus on general patterns in API design. A study aiming on use of constructor parameters by Stylos et al [6] showed that developers are more effective when no constructor parameters are required. In the next study Stylos et al [7] found a large usability impact on a method placement. Ellis et al. [8] argued that the factory pattern [9] is confusing, difficult to use and should be avoided. A case study of improving an API is described by Stylos et al. [10]. They used the aforementioned general patterns as well as input from interviews with developers.

Expert evaluation methods, sometimes referred to as discount usability methods, are not as time demanding as studies with the users. In the methods an expert or evaluator analyzes the API and checks whether it is compliant with predefined set of recommendations or guidelines. The first design principles for a programming language design called "Cognitive Dimensions of Notations" was published by Green and Petre [11] in 1989. The cognitive dimensions have been successfully used for evaluating visual programming [12] as well as API design [13]. Bore and Bore [14] proposed a set of simplified API profile dimensions, namely *Specificity*, *Simplicity*, and *Clarity*. De Souza and Bentolila [15] focused only on complexity of APIs by automatic counting primitive elements exposed to the users. They also proposed a visualization of complexity of different parts of an API. Watson [16] proposed a heuristic to check consistency in naming conventions in a large API consisting of hundreds of elements.

An effective method for API usability testing is called *API Usability Peer Reviews* published by Farooq et al. [3]. They define four roles in the process: *Feature owner*, who determine the goals of API peer review, *Feature area manager*, who fill any knowledge gaps and record feedback, *Usability engineer*, who is responsible for evaluating the usability, and *Reviewers*, who are not fully familiar with the API but have some knowledge the domain. The method was compared to traditional usability test and it was found that API peer reviews are faster and less expensive, but identify lesser usability flaws and are less sensitive.

The API usability studies and personal experience with API design led to development of many API design guidelines. For example, Bloch [1] lists 39 detailed design guidelines for an API design. Scaffidi [2], on the other hand, gives 4 general challenges of API usability and discusses strategies that designers and users developed to overcome these challenges. Henning [17] gives an detailed example of poorly designed API and discusses costs of dealing with it. He also describes and discusses 8 guidelines for a good API design. Zibran [18] gives a summary of existing design guidelines and results from studies with users. In [19] Zibran et al. identify relative significance of the design guidelines by studying usability-related bug posts across five different bug repositories.

Currently the research on usability of service-oriented architecture (SOA) focuses exclusively on usability of web services. Beaton et al. [20] identify usability

challenges for large SOA APIs. They also list HCI methods that can be used for evaluating SOA APIs – think-aloud protocol, expert evaluation using cognitive dimensions, and cognitive walkthrough. In Beaton et al. [21] a qualitative user study with six participants was conducted in which eight errors often made by the participants were observed. Jeong et al. [22] focused on improving documentation for an API of multiple web services. They provide 18 design guidelines for documentation of the API.

3 Methodology

Our methodological approach for evaluating API usability combines several usability evaluation methods. Each method has different strengths and weaknesses. It combines several methods in order to take full advantage of the strengths of each method. This combination provides an opportunity to get a big picture of the usability of the evaluated API. These may be not only problems and flaws in the code, but also conceptual and run-time problems as well as findings related to user experience. The methodology is the first step towards defining a structured process to achieve these goals.

3.1 Roles

Following roles are present in our methodological approach: Expert, Developer, and Evaluator.

(1) Experts are persons knowledgeable in the application domain with experience in the programming language of the evaluated API. They should not be fully familiar with the API. The experts are used in the first phase of the evaluation process – the heuristic evaluation.

(2) Developers are persons with experience in software development. They should be familiar with the programming language of the evaluated API, but they do not need to be familiar with the application domain. Developers are recruited to participate in the developer workshop (2nd phase of the evaluation process) and semi-structured interviews (3rd phase). The number of developers depends on the expected outcome of the evaluation. With low amount of developers mainly qualitative data can be obtained. With higher numbers we can gain quantitative data as well.

(3) Evaluators are persons who collect and analyze findings in each phase of the evaluation. They actively participate in the developer workshop and they also conduct the semi-structured interviews. The number of evaluators can vary between one to three depending on the number of developers and amount of work to be responsible for.

3.2 Process Overview

Figure 1 depicts the process of our evaluation approach addressing the elaboration of API distinct usability problems. Further it allows us to elaborate on an API-centric classification of usability problems as described in Section 4. The process consists of five phases: (1) planning, (2) heuristic evaluation, (3) workshop with developers, where the questions, problems, and potentials are evaluated, (4) interviews with developers and (5) the final analysis. The three different evaluation methods are conducted independently and the interviews with the developers are done right after the workshop to avoid the repetition of an introduction part to the evaluated API.

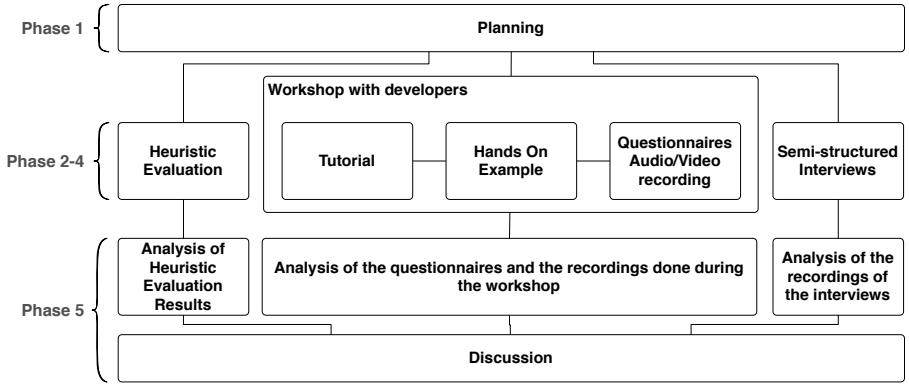


Fig. 1. Methodological Approach

Phase 1: Planning. In the planning stage the evaluators define a number of experts and developers involved in the evaluation process and start recruiting them. Evaluators also decide the objectives and identify relevant parts of the API to be evaluated. They also decide logistics of the latter stages – they plan time slot for heuristic evaluation, date and place of the developer workshop and time slots for the subsequent interviews.

Phase 2: The heuristic evaluation takes place approximately 2-4 days before the user workshop is conducted as it can reveal potentially problematic parts of the API. The developer workshop should then focus on these parts to clarify the problems. *Experts* are recruited and briefed about the API to evaluate by *evaluators*. Each expert receives a list of heuristics used to find, analyze, and categorize problems identified with the API. The outcome of the heuristic evaluation consists of a list of problems identified with the API while the problems are categorized according to the heuristics defined before. After the evaluation the findings of all experts are collected and analyzed according to the occurrence, the severity, and the addressed usability problem.

Phase 3: The developer workshop consists of three parts – introduction, tutorials, and hands-on example. In the introduction, the application domain of the API is presented as well as the API itself. A simple "Hello world" example is shown in tutorials. After that the developers are asked to implement a component defined by evaluators using the API. When the scope of the API is too vast to be covered within one task, we suggest to provide different tasks for each developer in order to cover as much of the API as possible. The developers are also asked to note problems they struggle with in a prepared questionnaire. Besides the developers, evaluators are also present during the hands-on example. Their role is to observe developers, log usability problems, and provide help when they struggle with a problem for too long. Video and audio is recorded during the workshop for future analysis. Demographic questionnaire, informed consent, and non-disclosure agreement forms have to be signed by the developers at the beginning of the workshop.

Phase 4: Post-workshop interviews with developers are held right after the workshop to debrief individual experience of workshop participants. An interview is not longer than half an hour. The total length of this phase depends on the number of participants. Using multiple evaluators can accelerate this process as they can work in parallel. For example, having eight participants and two evaluators, interviews can last only two hours in total. The last interview should be taken no more than two days after the workshop as the developers may forget some important facts. An interview is recorded to allow future analysis.

Phase 5: Discussion. After collecting all materials, evaluators analyze notes, logs, questionnaires, audio, and video recordings. They identify and summarize usability problems and rank them according to severity. The output of the methodology is a list of recommendations for improving the evaluated API.

4 Case Study – The API of the Contextual Interaction Framework

The Contextual Interaction Framework (CIF) is an OSGi[1] based framework developed at the ICT&S Center, University of Salzburg . The framework was built with the goal to support the engineer during the development of contextual study setups. The characteristics of such setups are twofold. Contextual study setups are setups that use information from the environment. Such information is obtained through external sources like sensors and actuators, which provide information about the study environment, the users, and the tasks users are doing during a contextual study. The second approach is to regard to contextual studies simply by defining the environment the study shall take place. The CIF provides functionality to support rapid prototyping by providing the developer a set of functionalities tailored for the rapid prototyping of contextual

[1] `http://www.osgi.org`

study setups. The second functionality is to not only simplify the development of such study setups for programmers, but provide tools that allow usability engineers to configure and conduct simple contextual study setups using existing functionality.

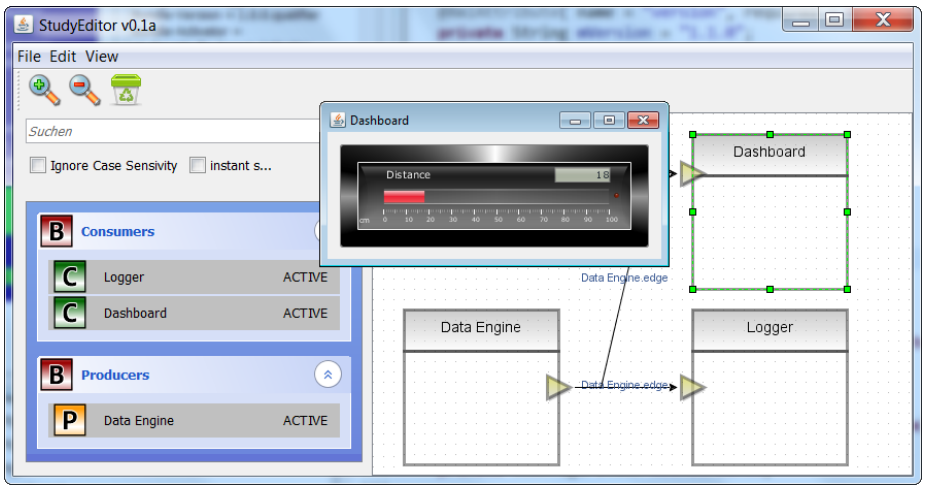


Fig. 2. Graphical user interface of the CIF in which bundles providing various functionality can be connected by creating directed wires.

Figure 2 depicts the main graphical interface of the CIF for usability engineers to setup studies and for developers to test their functionality to be developed. The CIF implements a plug-in architecture allowing the engineer to apply and extend its functionality. Such functionality encapsulated in *bundles* may be e.g. retrieving data from a sensor, logging to a database, etc. Bundles are depicted as rectangles with a title in the GUI. A bundle contains services that are capable of producing or consuming data. The services are displayed as triangles (pins) attached to the left side of the bundle (consuming data), or to the right side (producing data). The services can be linked together by wires to ensure the data exchange. The GUI provides graphical means to dynamically load, unload, start, and stop bundles and the services within bundles. In the example shown in the Figure 2, the *Data Engine* bundle is used to produce random data which is then displayed on a configurable *Dashboard* and at the same time logged to a database using the *Logger* bundle.

The setup defined by the CIF represents a data-centric approach where the engineer defines the data-flow between the available bundles, similar to a flow-diagram used during the specification phase of a system. The advantage is that the developer can immediately try out and thus verify the specific data-flow occurring in the system.

In order to address usability problems as well as the user experience of developers with the developed application programming interface (API), we setup a study that focuses in comparison to existing work (see Section 2) on the combination of multiple evaluation methods as described in Section 3.

4.1 Study Setup

Following the methodology described in Section 3 we setup a study to obtain results stemming from the different evaluation methods. Three different types of methods have been applied. We conducted a heuristic evaluation (see Section 3 – *Phase 2*) to obtain findings identified by experts. A one-day workshop for developer (see Section 3 – *Phase 3*) has been conducted with the intention to get in-depth information about the actual usability of the API. In addition we interviewed each workshop participant individually (see Section 3 – *Phase 4*) to be able to identify qualitative data about the usability of the CIF API.

The heuristic evaluation was conducted before the workshop for developers with the goal to identify usability problems of the API of the Contextual Interaction Framework. We selected 16 heuristics based on API design guidelines identified by Zibran [18] and used them to categorize the problem areas occurring when evaluating an API. These heuristics were selected as they summarize existing guidelines available in related work. Table 1 gives a short description of each heuristic based on [18]. The matter of analysis addressed by the experts has been the framework together with the documentation of the CIF, which we made available to the experts through a browser-based interface. The heuristic evaluation was conducted with four experts that were asked to analyze the API of the CIF itself together with the accompanying documentation. The experts were selected according to their experience in software engineering and API design. The number of experts is based on the studies of Nielsen and Molich [23] where 3-5 experts find 60–75% of existing usability problems of an interface. For usability of API this was also approved by Farooq et al. [3], who found that approximately 60% of API usability problems are identified with three to five experts in an API usability expert review. A higher number of experts is resulting in more precise results (see Cockton and Woolrych [24]) and shall be addressed in future studies. The findings were collected by evaluators. The severity of findings was rated in agreement between the evaluators on a scale from one to five where five is related to highest severity.

Table 1: Selected heuristics for the CIF evaluation

Name	**Description**
Complexity	An API should not be too complex. Complexity and flexibility should be balanced. Use abstraction.
Naming	Names should be self-documenting and used consistently.
Caller's perspective	Make the code readable, e.g. makeTV(Color) is better than makeTV(true).

Documentation	Provide documentation and examples.
Consistency and Conventions	Design consistent APIs (order of parameters, call semantics) and obey conventions (get/set mehods).
Conceptual correctness	Help programmers to use an API properly by using correct elements.
Method parameters and return type	Do not use many parameters. Return values should indicate result of the method. Use exceptions when exceptional processing is demanded.
Parametrized constructor	Always provide default constructor and setters rather than constructor with multiple parameters.
Factory pattern	Use factory pattern only when inevitable.
Data types	Choose correct data types. Do not force users to use casting. Avoid using strings if better type exists.
Concurrency	Anticipate concurrent access in mind.
Error handling and Exceptions	Define class members as public only when necessary. Exceptions should be handled near from where it occurred. Error message should convey sufficient information.
Leftovers for client code	Make the user type as few code as possible.
Multiple ways to do one	Do not provide multiple ways to achieve one thing.
Long chain of References	Do not use long complex inheritance hierarchies.
Implementation vs. Interface	Interface dependencies should be preferred as they are more flexible.

The developer workshop had the primary goal to tutor the developers on the CIF. The workshop started with a *setup session* that had the purpose to show the developers how the CIF can be setup and explain them the particular requirements. In the course of this session, they obtained all the resources necessary to develop with the CIF and also got an introduction to the documentation of the CIF, which should help them to setup the framework. In addition they were given a demographic questionnaire that focuses on the programming experience of the developers as well as a questionnaire in the form of a categorized notebook. The categorizations have been elaborated based on the guidelines for the semi-structured interviews so that we could use the notes that the developers took during the whole workshop as a basis for adapting the interview questions based on the experiences of the particular developers.

The setup session was followed by a presentation about SOA and OSGi and an in-depth presentation providing the developers with insights about the functional parts of the CIF as well as the API that the CIF provides to be able to extend its available functionality. This was done in the form of a slideshow presentation. After the theoretical introduction the developers were given examples that show how to use the framework. Also the different available bundles have been discussed and the developers obtained an introduction to the templates provided by the framework, which act as a starting point to develop an own CIF bundle. The examples were first presented in a slideshow and then explained hands-on using the Eclipse development environment and the CIF. After

this the developers were asked to start developing their own *hands-on example bundle*. They were provided with ideas of useful bundles that they could easily implement. Optionally, they also had the possibility to bring in their own ideas, but none of them did so. The developers were asked to use the documentation and templates available and to note all the problems, misunderstandings, potential bugs, and errors they identified. These notes were collected and provided us with information regarding the semi-structured interviews conducted after the workshop. In addition the workshop was recorded using audio and video which provided us with additional means of clarifying problems that occurred during the workshop.

The workshop was held at the ICT&S Center, University of Salzburg where 8 developers (all men, mean age 29, st. dev. 3.1) participated in a one-day workshop. Pre-conditions for the developers participating have been defined through required knowledge in programming Java and the Eclipse programming environment. Seven of them had a university degree, one was a university student. Five of them had at least four years of developing experience, while the experience of the rest varied from one to two years. Their Java experience varied from half a year to nine years.

Post-workshop interviews have been done right after the developer workshop on the same day and on the day after. Semi-structured interview guidelines were defined to obtain comparable qualitative results. The guidelines covered API usability issues based on Table 1 as well as usability factors of the API that have been defined based on Bloch's guidelines [1]. The factors cover the parameters of learnability, rememberability, efficiency, misconceptions, errors, perception of the API, and self-documenting structure and naming. Further parameters of the documentation in terms of completeness, understandability, and helpfulness have been addressed during the interviews. The notes that have been taken by the developers during the workshop act as a basis for the interviews as they are structured the same way as the guidelines.

4.2 Analysis

In this section we present the results of the in-depth analysis of the three different parts of the evaluation of the API of the CIF.

Heuristic Evaluation. Four experts participated in the heuristic evaluation of the API. All four experts were familiar with the application domain and they had already known some parts of the API. They inspected total of 110 classes and 30 interfaces in 29 packages. The length of one inspection varied between three to six hours.

The findings were analyzed and clustered after collecting from experts. A total of 127 unique usability problems were identified by the experts with mean severity of 3.1 (1 – the least severe problem, 5 – the most severe problem) and standard deviation of 1.06. The distribution of the usability problems over heuristics is

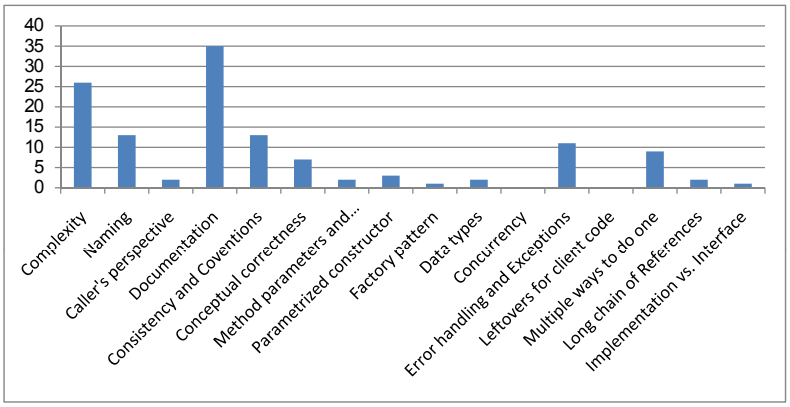
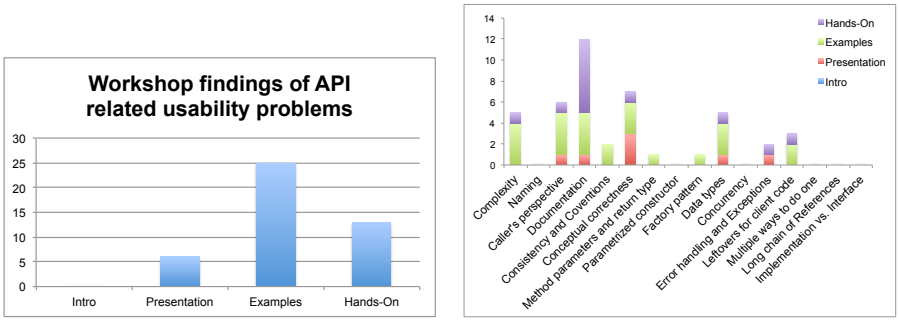


Fig. 3. Distribution of findings in heuristic evaluation

depicted in Figure 3. Most problems were in missing, incorrect, or incomplete documentation (35), unnecessary complexity (26), e.g. some classes or methods should be removed or redesigned to decrease complexity of the API. Many problems were also in obscure or wrong naming (13), and consistency and conventions (13), e.g. a setter method was missing for its getter counterpart. Heuristic evaluation explore all parts of API equally. Due to inspection-like methodology, the experts could only evaluate definition of classes, methods, interfaces etc., but were not able to analyze run-time behavior.

Developer Workshop. During the developer workshop different types of materials have been collected. The evaluators took notes, the workshop sessions have been recorded on video for post-analysis, and notes have been taken by the participants of the workshop. The collected material provided us with insights into understanding problems of the presentation and the tutorial as well as usability problems of the API and the documentation. During the whole workshop 44 usability relevant issues with the API have been identified. We first separated the findings according to the different parts of the workshop – (a) introduction, (b) presentation, (c) examples, which included a tutorial, and (d) hands-on. Figure 4(b) describes this distribution where the findings are categorized based on the defined heuristics and on the part of the workshop. During the introduction and the setup phase (Intro), where the whole system was setup based on a tutorial provided in the documentation, no API relevant usability issues could be detected. During the presentation phase, 15.91% of API usability relevant issues were detected. During the presentation and elaboration of the examples the biggest part of the usability problems was found (54.55%) while during the hands-on examples phase of the workshop, where the developers had to use the API and develop an own bundle 29.55% of the Usability problems have been identified (see Figure 4(a)).



(a) Distribution of findings between the applied methods

(b) API relevant usability findings

Fig. 4. Workshop findings

The issues identified in the workshop have been split based on the categorization of the heuristics (see Table 1). Figure 4(b) shows this distribution. The most of the problems are related to the documentation issues (27,27%), to the concept in terms of conceptual correctness (15,91%), and to the caller's perspective problems (13,64%). An interesting outcome is that during the tutorial when the examples have been elaborated and tutored only 33.3% of the documentation problems were found but 58.3% of these problems could be identified during the hands-on part where developers worked directly with the API. Concluding the API is used in more depth during the real application than during the phase of learning the concept. This conclusion is also supported by the fact that 60% of the identified problems regarding the complexity of the system occurred during the example phase, while only 20% of the complexity related problems emerged during the hands-on phase.

Post-Workshop Interviews. The interviews with developers were conducted within the next two days after the workshop. A total of eight developers, who participated in the workshop, took part in the interviews. The interviews identified 15 unique usability problems, which developers struggled with during the workshop. Classification of these problems according to heuristics is depicted in Figure 5. The mean severity of the problems was 3.2 (sd = 1.14) on a five-point Likert scale. Again, the largest number of problems was related to documentation (5). Participants also mentioned three naming problems and three conceptual correctness problems. Interviews allowed us to gain valuable insights into the biggest problems that the developers struggled with. These problems are the most crucial to be corrected as they represent obstacles, which the developer has to overcome during the very first use. Such obstacles can then easily discourage a novice CIF developer from using our framework.

Besides usability problems, interviews revealed the subjective attitude of the developers to CIF. Three participants liked the structure of the API, four partici-

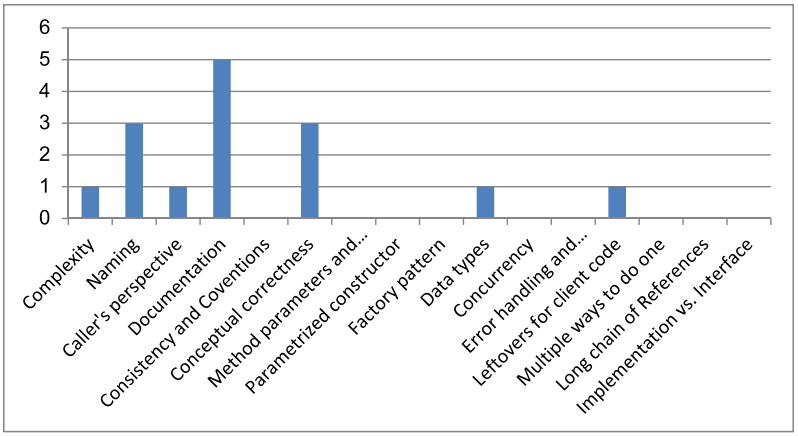


Fig. 5. Distribution of findings from interviews with developers

pants positively commented learnability and five rememberability. Even though most usability flaws were found in the documentation, three participants stated that the documentation is complete and very helpful. Five participants found naming of methods, classes and interfaces self documenting. The participants were also asked to comment on the idea of CIF. Five participants liked the idea, but three of them stated that it is important to have a library of bundles. In the current state, the library of bundles is rather limited, which caused two neutral attitudes. One participant did not see any added value in the CIF as he is not developing relevant applications.

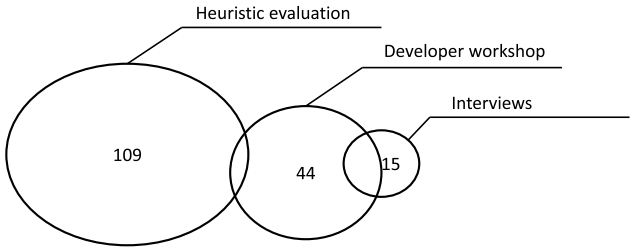


Fig. 6. Distribution of findings

4.3 Discussion

The evaluation revealed a total of 168 usability problems in our API, 157 of them were unique. 109 problems were found in heuristic evaluation, 44 in developer workshop and 15 in the interview. 9 problems were found in both, developer

workshop and interviews. In the interviews these problems were addressed in more detail. Only two overlapping problems regarding missing documentation were found in the heuristic evaluation and developer workshop. The distribution of the problems is depicted in Figure 6. Surprisingly, no other overlapping problem was found.

The findings and their structure were different from each part of the methodology. Altogether, they compose a transparent picture of the whole API with focus on its most important parts. An excerpt of the findings is shown in Table 2. While findings from heuristic evaluation are formal and detailed descriptions of problems in the API, findings from the workshop focus on usability problems regarding concept and structure of the API. During the workshop we could reveal more run-time problems that are not obvious and thus cannot be found in the heuristic evaluation. Interviews not only provided deeper understanding of these problems, but also revealed their attitude towards the CIF.

Table 2. Findings example, HE = Heuristic evaluation, W = workshop, I = interviews

Finding	**Phase**
...	
class: ServiceMetaData *finding*: no default constructor, constructors have 4-8 parameters *heuristics*: parametrized constructor	HE
class: ServiceMetaData *finding*: no setter counterpart for getDate() method *heuristics*: consistency and coventions	HE
Data types produced and consumed are stored in a configuration. The configuration is created when the bundle is run for the first time. But if the types are changed after the configuration was saved, it always loads the types from the first saved configuration. The configuration has to be deleted manually and that confused four workshop participants.	W
I was not able to run my bundle for a long time, because the system allowed to instantiate only bundles with specific prefix and I was not aware of it.	I
I am not happy with names consumer, producer. From the point of view of the input device is it actually vice-versa.	I
...	

The data collected during the workshop resembled usability testing. The reason why the workshop was used instead of standard usability testing was that the participants had to acquire some basic knowledge about SOA and about the idea of the CIF before taking part in the evaluation. The workshop allowed us to make a presentation of CIF and examples for all participants together in

order to save time. Moreover, the presentation during the workshop initiated discussions, from which we could gain valuable insights and comments. During the workshop the developers got familiar with concept of CIF and its API and thus can be subjects for future studies without necessity of the time consuming introduction.

The severity of problems identified during the evaluation was also taken into account. The problems were rated on a five-point Likert scale where five is the most severe problem and one is only a minor problem. Table 3 shows the distribution of the degree of severity among each phase of evaluation and among the different methods applied during the workshop. In order to ensure validity and consistency of the rating, the same group of people (evaluators) assigned the severity for the findings in each phase.

Table 3. Severity of API usability problems for each phase

Phase		Mean	SD
Heuristic evaluation		3.1	1.06
Workshop	Presentation	2.0	0.89
	Examples	2.8	1.30
	Hands-On	3.7	1.38
Interviews		3.2	1.14

The highest mean severity of the usability problems found is for the hands-on part of the workshop. This indicates that the severity of usability problems of an API increases the more the developers are involved in actual using the API. A more abstract engagement in the API (like in the case of the presentation and tutorial) results in the identification of less severe problems.

Results collected from all phases of the evaluation are divided into 16 fine-grained heuristics. As the heuristics do not cover especially user experience factors we developed a classification that provides another categorization of the findings based on the characteristics of the findings rather than on the applied methodology. The classification consists of the following four categories:

1. *Documentation.* This category contains all findings related to documentation – source code documentation, online documentation, tutorials, FAQs, etc. It is mainly covered by the heuristic "Documentation" already used in the evaluation.

2. *Runtime.* Findings that can be only revealed when working with the API during at runtime. An example of such finding can be a misuse of the API that results to a runtime exception.

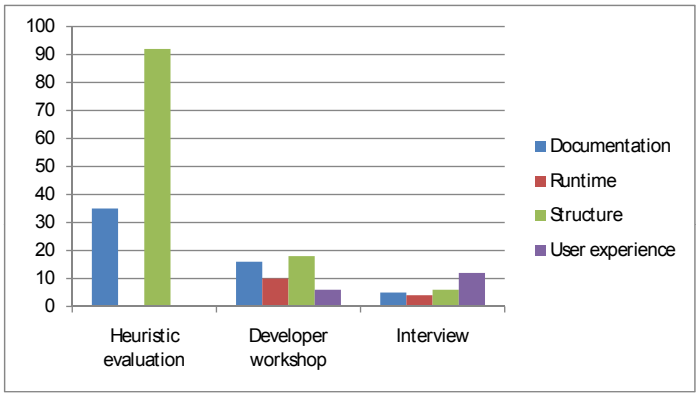


Fig. 7. Classification of the findings.

3. *Structure*. Findings related to structure include low-level problems such as naming or complexity of the API, but also higher-level problems, e.g. problems related to API concept and actual perception of the API.

4. *User experience*. This category contains subjective findings related to all aspects of the experience when working with the API. This includes all usability parameters such as ease of use or efficiency, etc., as well as typical user experience attributes like attractiveness, emotions, perceived utility, perceived effeciency, etc.

The classification has been applied to the findings and the result is depicted in Figure 7. The findings from heuristic evaluation are classified either as documentation or structure problems. Neither runtime nor user experience findings were revealed in this phase as no developer was involved. This shows that heuristic evaluations are mainly applicable in order to identify problems related to the structure and also the documentation of the API. The problems identified are of formal nature and do not cover usability aspects. During the developer workshop we could identify in addition to documentation and structural findings also runtime related findings and a low number of user experience issues. During the interviews a lower number of runtime related problems, structural, or documentation issues were found. The strength of this method lies in finding user experience problems. This results in a categorization of identifiable problems regarding their characteristics.

Each applied method has its strengths and weaknesses and focuses on different areas of problems. A combination of the selected methods resulted in a more complete view about the overall usability covering not only static but also dynamic aspects appearing through the hands-on workshop. Combining such specific methods thus allows to simultaneously address related usability problems with an API in an appropriate way. Anyway, further combinations of methods

and variations of the methodologies still need to be compared to identify good combinations of methods to evaluate the usability of APIs.

5 Conclusion and Future Work

To study the appropriateness of HCI methods for evaluating the usability of the API of the Contextual Interaction Framework a methodology based on a mixed methodical approach focusing on the mutual pollination of the contributing methods has been developed. Mixing the HCI methods is important as it allows us to get a big picture of problems covering all areas of API usability. We applied this method in a case-study and were able to identify the applicability of different methods for different types of results. Based on the results we could elaborate a classification that provides a mapping between usability findings and HCI methods to be applied. The identified categories reflect structural, run-time related, documentation related, and user experience related usability findings of APIs. The applied methodology generated insights based on an inspection method, a user test, and interviews. They represent a first step towards the classification of the applicability of HCI methods to identify specific usability issues in a software development process. Current limitations of our work mostly exist regarding the generalizability of the approach. In future studies, we will address this by comparing combinations of other usability methods like cognitive walkthroughs, focus groups, API reviews, etc. By doing this, we expect to obtain a more fine-grain classification of the potential methods reflecting relevant areas of usability findings in APIs. Additionally we will evaluate the applicability of user experience methodologies like questionnaires targeting user experience factors like acceptance, attitude, or emotions with respect to the usability and user experience of APIs.

6 Acknowledgements

The financial support by the Federal Ministry of Economy, Family and Youth and the National Foundation for Research, Technology and Development is gratefully acknowledged. The work described in this paper was supported by the Christian Doppler Laboratory for “Contextual Interfaces” and the COMET K-Project “AIR – Advanced Interface Research”. Further we want to thank our colleague Michael Humer who supported us throughout the conception of the work and the user study described in this paper.

References

1. Bloch, J.: How to design a good API and why it matters. In: Companion to the 21st ACM SIGPLAN symposium on Object-oriented programming systems, languages, and applications, ACM (2006) 506–507

2. Scaffidi, C.: Why are APIs difficult to learn and use? Crossroads **12**(4) (2006) 4–4
3. Farooq, U., Zirkler, D.: API peer reviews: a method for evaluating usability of application programming interfaces. In: Proc. of CSCW '10, New York, NY, USA, ACM (2010) 207–210
4. McLellan, S.G., Roesler, A.W., Tempest, J.T., Spinuzzi, C.I.: Building More Usable APIs. IEEE Softw. **15**(3) (1998) 78–86
5. Robillard, M.P.: What Makes APIs Hard to Learn? Answers from Developers. IEEE Software **26**(6) (2009) 27–34
6. Stylos, J.: Informing API Design through Usability Studies of API Design Choices: A Research Abstract. In: Proc. of IEEE Symp. VL/HCC '06. (2006) 246–247
7. Stylos, J., Myers, B.A.: The implications of method placement on API learnability. In: Proc. of ACM SIGSOFT '08/FSE-16, New York, USA, ACM (2008) 105–112
8. Ellis, B., Stylos, J., Myers, B.: The Factory Pattern in API Design: A Usability Evaluation. In: Proc. of ICSE '07, IEEE Computer Society (2007) 302–312
9. Gamma, E., Helm, R., Johnson, R., Vlissides, J.: The Abstract Factory Pattern. In: Design Patterns: Elements of Reusable Object-Oriented Software. Addison-Wesley (1995)
10. Stylos, J., Graf, B., Busse, D.K., Ziegler, C., Ehret, R., Karstens, J.: A case study of API redesign for improved usability. In: Proc. of IEEE Symp. VL/HCC '08, Washington, DC, USA, IEEE Computer Society (2008) 189–192
11. Green, T.R.G.: Cognitive dimensions of notations. In: Proc. of the 5th Conf. of the British Computer Society, HCI Specialist Group on People and Computers V, New York, NY, USA, Cambridge University Press (1989) 443–460
12. Green, T.R.G., Petre, M.: Usability Analysis of Visual Programming Environments: a 'cognitive dimensions' framework. Journal of Visual languages and computing **7**(2) (1996) 131–174
13. Clarke, S.: Measuring API usability. Doctor Dobbs Journal **29**(5) (2004) 1–5
14. Bore, C., Bore, S.: Profiling software API usability for consumer electronics. In: Proc. of ICCE '05. (2005) 155–156
15. de Souza, C., Bentolila, D.: Automatic evaluation of API usability using complexity metrics and visualizations. In: Proc. of ICSE-Companion '09. (2009) 299–302
16. Watson, R.: Improving software API usability through text analysis: A case study. In: Proc. of IEEE Conf. IPCC '09. (2009) 1–7
17. Henning, M.: API Design Matters. Queue **5**(4) (2007) 24–36
18. Zibran, M.: What Makes APIs Difficult to Use? IJCSNS International Journal of Computer Science and Network Security **8**(4) (2008) 255
19. Zibran, M., Eishita, F., Roy, C.: Useful, But Usable? Factors Affecting the Usability of APIs. In: Proc. of WCRE '11. (2011) 151–155
20. Beaton, J.K., Myers, B.A., Stylos, J., Jeong, S.Y.S., Xie, Y.C.: Usability evaluation for enterprise SOA APIs. In: Proc. of SDSOA '08, New York, NY, USA, ACM (2008) 29–34
21. Beaton, J., Jeong, S.Y., Xie, Y., Stylos, J., Myers, B.A.: Usability challenges for enterprise service-oriented architecture apis. In: Proc. of IEE Symp. VLHCC '08, Washington, DC, USA, IEEE Computer Society (2008) 193–196
22. Jeong, S.Y., Xie, Y., Beaton, J., Myers, B.A., Stylos, J., Ehret, R., Karstens, J., Efeoglu, A., Busse, D.K.: Improving Documentation for eSOA APIs through User Studies. In: Proc. of IS-EUD '09, Berlin, Heidelberg, Springer-Verlag (2009) 86–105
23. Nielsen, J., Molich, R.: Heuristic evaluation of user interfaces. In: Proc. of ACM SIGCHI '90, New York, NY, USA, ACM (1990) 249–256
24. Cockton, G., Woolrych, A.: Sale must end: should discount methods be cleared off HCI's shelves? interactions **9**(5) (2002) 13–18