First publ. in: Proceedings of the 2011 annual conference extended abstracts on Human factors in computing systems / the 29th Annual CHI Conference on Human Factors in Computing Systems : CHI 2011, Vancouver, BC, May 7 - 12, 2011 / general chair: Desney Tan. - New York, NY : ACM, 2011. - S. 3373-3382. - ISBN 978-1-450-30268-5

The Concept Maps Method as a Tool to Evaluate the Usability of APIs

Jens Gerken, Hans-Christian Jetter, Michael Zöllner, Martin Mader, Harald Reiterer
University of Konstanz, HCI Group
Konstanz, Germany
{firstname.lastname}@uni-konstanz.de

ABSTRACT
Application programming interfaces (APIs) are the interfaces to existing code structures, such as widgets, frameworks, or toolkits. Therefore, they very much do have an impact on the quality of the resulting system. So, ensuring that developers can make the most out of them is an important challenge. However standard usability evaluation methods as known from HCI have limitations in grasping the interaction between developer and API as most IDEs (essentially the GUI) capture only part of it. In this paper we present the Concept Map method to study the usability of an API over time. This allows us to elicit the mental model of a programmer when using an API and thereby identify usability issues and learning barriers and their development over time.

Author Keywords
API usability, evaluation method, longitudinal, concept maps.

ACM Classification Keywords
H5.2. [Information Interfaces and Presentation]: User Interfaces. Evaluation/methodology.

General Terms
Measurement.

INTRODUCTION

In today's software development it has become a rare occurrence that everything has to be programmed from scratch. This is not only true for subsequent releases but also for "new" products. Instead, developers often rely on existing widgets, frameworks, libraries, or software development toolkits that provide existing code structure for reuse. To access these, application programming interfaces are provided (APIs) and while there may be many different kinds of APIs they all serve the same purpose, as Daughtry et al. [10] described it: "they each provide a programmatic user-interface to a module of code". As with any kind of interface, some of them are more usable than others and this can have a tremendous impact on the final product as well as the efficiency of the development process. Advocates for API usability, such as Joshua Bloch from Google have stressed that

"good APIs increase the pleasure and productivity of the developers […] the quality of the software they produce, and ultimately, the corporate bottom line. Conversely, poorly written APIs […] have been known to harm the bottom line to the point of bankruptcy" [4].

A number of researchers have started to investigate the usability of APIs more in detail in recent years, with McLellan et al. [25] often being cited as having conducted the first formal usability study of an API. Since then, there have been quite a few studies on different design aspects, such as the use of different patterns (e.g. [13]) or API documentation. Besides, several books and papers providing API design guidelines have been published [9] [29]. At the CHI 2009 conference a special interest group (SIG) took place on API Usability [11] to discuss the challenges of designing a usable API. As one outcome, the organizers have created a web resource with a collection of useful resources and links to papers about the topic.

An area within this field, that one can find only little research about, are the data gathering methods used to actually assess the usability of an API. Essentially, most methods have been adaptations of existing HCI usability evaluation techniques such as usability tests and inspection methods. Since an API is fundamentally different from a graphical user interface, for which these methods have been designed for, we think that there is a huge potential for evaluation methods that have been specifically designed to address the particularities of an API. Since the GUI, which allows researchers to directly observe the interaction with an interface, is missing, direct observation methods are more vulnerable to subjective interpretation. Inspection methods require a high level of knowledge about the API and API programming in general by the analyst. Besides, writing a piece of code is often a tedious process over days if not weeks, so in case of the observation approaches and depending on the complexity of the API it can be difficult to define ecologically valid tasks that fit in a 1-2 hours observation session. Furthermore, using an API is a constant learning process, as developers seldom read documentation in advance but rather search for examples or documentation on the fly. Thereby, a research method for API usability should be able to grasp this learning process

over time and allow the researcher to identify learning barriers.

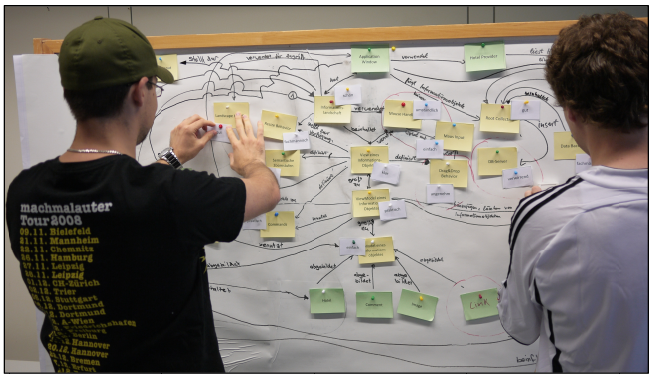

Figure 1: A concept map of the ZOIL API

In this paper we will address and contribute to this issue by presenting an API evaluation method that is based on the concept mapping technique (see fig. 1) known from learning theories [27]. It allows the researcher to elicit the developer's mental model when working with an API by making the interaction visible. Furthermore, it is especially useful in a longitudinal design as our method is designed to allow the track of changes within the data over time - a common and difficult to address challenge in longitudinal research [8]. Thereby it can be used to assess the learning barriers developers come across when working with an unfamiliar API as well as their evolution over time. Besides, our method is easy to apply in practice as it uses hands-on materials and may include a wide variety of possible metrics. In the following sections we will first review existing literature of API evaluation methods to discuss the challenges that a method should address. We will then present the method in detail, outlining the materials, the design rationales, and the data-gathering process. Eventually, we will discuss the application and analysis possibilities of the method by presenting a case study of an API evaluation with university students who were given the task to create a software prototype with the help of an unfamiliar API.

CHALLENGES FOR THE EVALUATION OF AN API

When reviewing the literature about API evaluation methods, only few papers focus specifically on the data-gathering method (e.g. [15] [7] [25]). However, there are quite some papers that present, discuss, and evaluate certain design choices, such as specific patterns of an API. In principle, we can identify three different purposes for studying the usability of an API. The first is to support the development process of an API, following the user-centered tradition of usability engineering lifecycles. In this case, studying the usability has the goal to obtain answers to questions such as how easy it is to learn the API, how efficiently it can be used for specified tasks, or which areas are difficult to use and lead to misconceptions in the programmers' understanding. The second purpose is to derive design principles and a theoretical foundation for the design of new APIs as discussed in the introduction – thereby, studying and analyzing existing APIs serves the purpose of understanding how people actually use these which can then help us to design better APIs in the future. The third purpose might be to conduct comparative studies of APIs. This is especially important when companies have to decide which of several competing APIs they should introduce in their software development process but also for marketing purposes when launching a new API.

Data-gathering

A tremendous challenge for the evaluation of an API is that using and interacting with an API is much more subtle than using a standard software application and therefore more difficult to observe and analyze. The reason is that the IDE as the interface does not necessarily capture the whole interaction with an API. Accordingly, it is not straight forward to define wrong doings or errors during the observation of users since there are many ways to reach a goal.

Nevertheless, the most common approaches to study the usability of an API have been lab based usability tests in combination with the thinking aloud protocol. In the already cited study by McLellan et al. [25] four programmers from an API target group were given the task to analyze and understand a code example that used calls from the API. They were asked to think aloud while trying to understand the code and express what information about the API they would need to reproduce such a code sample. They were also asked what further features they would expect from the API from what they have seen, allowing the researchers to assess how users of this API might perceive its ceiling [26]. As the participants were allowed to ask questions to an API expert, one could describe this approach as some kind of co-design for API development. In Klemmer et al [22], the authors conducted a more traditional usability test with seven participants using the *Papier-Mâché* toolkit for developing tangible user interfaces. Participants were first introduced to the toolkit and then were asked to complete three typical programming tasks by using it. Thinking aloud as well as participants' Java code was then used to analyze the usability of the toolkit. In a similar way, Heer et al. [19] analyzed the usability of their *prefuse* toolkit. An interesting alteration of this approach was proposed by Beaton et al. [3]. In their approach, participants first would have to write in pseudo code what they would expect in the API for a certain task and then perform the real task using the API. Thereby, the authors suggest, one can better assess the mapping between the user's mental model and its matching with the real world. All these approaches had the primary goal of finding usability flaws within a specific API rather than generate knowledge for a theoretical basis for API design. Contrary, de Souza et al. [12] performed an extensive field study to understand how APIs are used in practice, which roles they serve, and whether their use has only beneficial purposes or also drawbacks. The authors spent 11 weeks on site of a

software company, conducting non-participant observations and semi-structured interviews, as well as being able to gain access to documents about the processes and to discussion databases. In a grounded theory approach, the data was analyzed and continuously enriched with new observations and interviews. The nature of such a study obviously makes it inappropriate for analyzing the usability of an API during the development process, nevertheless more focused, short-term field observations can help in defining e.g. requirements for an upcoming new version of an existing API.

Next to these methods with direct involvement of end users (programmers), there has also been some research regarding analytical inspection type methods, comparable to usability inspection methods such as cognitive walkthroughs or heuristic evaluation. The main advantage here obviously is that no real users are needed which may facilitate testing as the target group of an API often is spread around the world and not as easy to get into a lab as the potential iPod user. Farooq and Zirkler [15] presented a method called API Peer reviews, which is based on cognitive walkthroughs and adapted to APIs. The approach has been used within Microsoft in addition to usability tests. It is a group-based usability inspection where different members of the API development team serve different roles, e.g. the feature owner is the one whose part of the API is under review and some of the team members serve as reviewers. During a 1.5 hours meeting the goal is to walk through a specific part of an API while trying to resemble a typical scenario of use. The reviewers comment on this by trying to put themselves in the role of users. The method proved to be highly scalable and to have a very good benefit-to-cost ratio. Nevertheless, the authors see it as an addition to usability testing rather than a replacement.

Metrics

Regarding the metrics used in the studies cited above to assess the usability, there have been both qualitative and quantitative approaches. Purely quantitative measurements include task-completion times [1] [13], sometimes lines of codes [22], or number of iteration steps needed [1]. While these can help in comparing different APIs [13] they can only indicate usability issues in a rather broad sense. More detailed qualitative analysis of the think-aloud protocol and video observation data helps in identifying more deep usability issues. Here, the work of Clarke [7] has been rather influential. He used the cognitive dimensions framework [18] and adapted it to fit the needs of API usability evaluation. By using this framework, researchers can cluster findings in the different categories, e.g. API Viscosity or Consistency and by that get help in identifying which higher level concept of the API might be problematic. Farooq and Zirkler also relied on this framework to cluster the findings of their API Peer Review approach [15].

Ko et al. [23] on the other hand identified six learning barriers of an API, such as selection barriers or information barriers, in a large field study which can be again used to cluster qualitative data. Identifying such learning barriers can be one step to assess the threshold of an API, which basically means how difficult it is to achieve certain outcomes with it.

Myers et al. introduced the *threshold* and *ceiling* concept as quality criteria. “The threshold is how difficult it is to learn how to use the system and the ceiling is how much can be done using the system” [26]. In most of the studies cited so far, the goal was to identify the threshold or barriers within the API that seem to increase the threshold. The ceiling on the other hand defines what is achievable with an API. So instead of looking at the process, one can look at the artifacts that can be created by using a specific API and thereby determine its value and quality. Common approaches here are case studies that show a wide range of possible systems [19] [22].

In summary, the most common data-gathering approaches are usability tests, thinking aloud, inspection methods, and in some cases field observations. From an analysis perspective, the metrics include straight forward aspects such as task-completion time and lines of codes as well as more theoretical grounded analysis frameworks such as the cognitive dimensions.

We think that these current approaches seem to be insufficient to address two major aspects: 1) in case of observation or inspection approaches, most studies are limited to one or maybe a few hours. Thereby, tasks are rather simple and most of the time “pre-defined” with given code samples. More complex or even real tasks, where developers can use the API for real projects are seldom and difficult to integrate in such study designs, although such tasks would provide very valuable input regarding the usability of an API in real world situations. 2) It is difficult to assess eventual changes in learning barriers or the threshold of an API during a single session. One can assume that barriers shift during longer usage times and thresholds may be perceived differently after some time. Both of these aspects can be addressed by using a longitudinal study design, which basically gathers data at more than one point in time [28]. What is still needed is an appropriate data-gathering method which then makes it possible to integrate more complex tasks and observe these changes. Besides, most approaches rely on direct observation or inspection. However, given the task of coding a piece of software, we can see a value of retrospective approaches that might allow users to better reflect on the pros and cons of a certain API. Simple retrospective interviews seem insufficient to do so, as they would lack a proper artifact to trigger the discussion with the participant. In the following section we will present the Concept Map method, which incorporates a longitudinal

field study design and a visual representation of the API usage and therefore directly addresses these issues.

THE CONCEPT MAP METHOD

Novak [27] introduced concept mapping in the late 1960s and early 1970s as a research method during a longitudinal 12-year research project that assessed how children's understanding of science concepts changed over time. Concept maps can be described as visual knowledge representations with nodes and edges. Each node represents a concept and is linked with one or several other nodes via edges. The edges are typically directed and labeled to describe the nature of the connection between the two nodes. Originally, it has been defined as a top-down diagram to decompose hierarchical relationships within a main concept. However, it has since been applied in a number of variations, including non-hierarchical but flat structures. While Novak has originally introduced it to improve teaching biology, it has since then shown to have great value on student learning for a variety of topics and teaching situations [14], both as a learning strategy and as an instructional strategy. It has even been applied as a means to assess the students understanding of science concepts [24]. In HCI, concept maps have been applied as creativity and structuring tools, similar to mind maps, e.g. during the requirements phase in a usability engineering process [2].

Given the nature of an API, we propose concept maps as an evaluation and assessment method to elicit the programmer's mental model of an API. Thereby, we are able to identify misconceptions and problematic areas and assess how these are changing over time.

Main Idea

An API, by definition, is always an interface between two distinct pieces of software code. One of them being the application that is under development and the other being a more general framework or SDK to which the API provides the interface. Our concept mapping approach asks participants to visualize this relationship between their own piece of code (which can be a given task or a real application) and the API. This happens during a 30-60min observation session, which is video-taped and includes a thinking aloud protocol. For each participant, this session is repeated (e.g. once a week over a five week period) depending on the complexity of the API and the application. During these repetitions, the users don't start from scratch but are handed their concept map from the last session and asked to change everything which they do not longer perceive as being a correct representation of their mental model. This is an important aspect, as we do not ask them how their understanding of the API has changed (which would be much more difficult to answer) but ask them to update their own artifact. How their understanding has changed is then implicitly reflected in the changes they make to the map. By analyzing these maps together with API experts, it is now possible to understand misconceptions of or simply usability problems with the API. Given the graph-based structure of such a map, we are furthermore able to (digitally) compare it with a "master" map created by the API developers or API experts.

Design rationale & Materials

The method is designed with hands-on materials, making it easy to apply in any environment. In the following, we will present the materials needed and discuss the design rationale and possible design choices behind them. We have explored different alternatives in two case studies, of which we will present the second one in detail later on.

A modified pin-board/whiteboard: We have applied the method both on a table and on a vertical pin-board. While the table allows more people to position themselves around the map, the vertical board has the advantage that it allows the user to step back and gain an overview, which we consider as an essential advantage of that setting

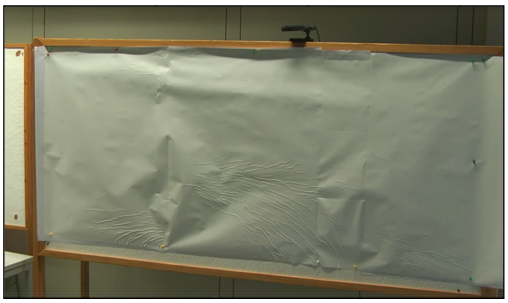

Figure 2: A "modified" vertical pin board

As we want to allow participants to easily place concepts on the map as well as change the placement and any links they have created, a huge whiteboard would be the best solution. A hands-on alternative, which we used during our second study, is a modified pinboard with painter foil pinned upon it (see fig. 2). This allows participants to pin concepts as on a pinboard and draw and remove connections as on a whiteboard.

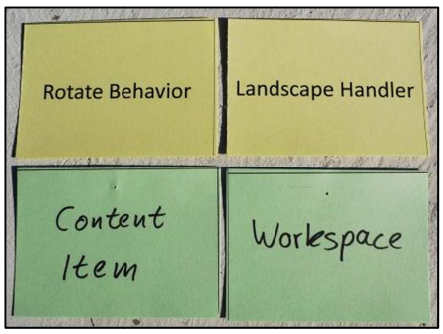

Figure 3: yellow API concepts and green prototype concepts

The concepts: In our studies, we used cards of the size 7.5x10.5cm for each concept (see fig. 3). Depending on the goal of the study, it is possible to either pre-define concepts or let participants define these by themselves. A more explorative study would prefer the latter while a more controlled setting, with specific parts of an API under investigation, should pre-define concepts. This allows

easier comparison of concept maps between users or with a master map, enabling quantitative data analysis. What is a concept? The granularity of a concept can be adapted to the research goal as well. A concept can be a certain method, a class name or a higher level construct that includes multiple classes. It can also be detached from the actual code by using an abstract or a user-centered perspective. If the API for example is responsible for handling the input modalities, one concept could be "Input modality" or it could be decomposed into "mouse input", "touch input", "voice input", etc. By using different levels of granularity for different parts of the API, the researcher can define which aspect is under close investigation (the detailed part) and still assess the overall understanding of the whole API.

We further distinguish between API concepts and what we call "prototype concepts" which include the concepts for the piece of software the participant is writing. The task for the participant during the concept mapping session is to connect the prototype concepts with the API concepts by drawing a line and add a label to it that further explains the connection. Basically, we thereby ask the users to visualize the processes between the software and the API.

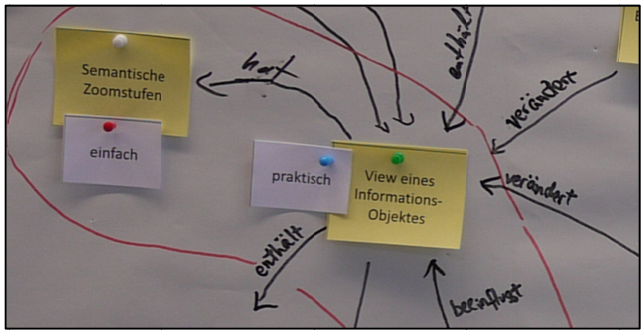

Figure 4: Adjectives attached (easy, practical) to Concepts (semantic zoom level, view of info. object) and a problem area

Rating concepts and indicating problem areas: The method includes two further tools to help understand potential usability issues (see fig. 4). First, the participants are asked to assign one of several pre-defined adjectives, which are also written on individual cards, to each concept at the end of a session. These adjectives are presented as contrasting pairs of adjectives as in a semantic differential. We have used a set of eight pairs, including the likes as convenient – inconvenient, easy – complicated, beautiful – ugly. Participants are only allowed to assign one adjective per concept - the one which best expresses their feeling. The main idea here is to quickly identify the concepts which trigger a positive and which trigger a negative feeling. Using the adjectives allowed us to use the same approach as with the concept map cards, nevertheless, other emotion or mood measures can be applied here as well. The second tool is asking participants to indicate problem areas by drawing a red line around those concepts that they have had the most trouble with when they were using the API. In that way, individual concepts can be marked as well as a whole group of concepts. We have found that asking participants to indicate these areas quickly triggers responses explaining the problems and thereby providing tremendous help in understanding any usability issues. Again we think that having the artifact helps participants to talk about such issues more easily, as they can visualize and assign the problem to a concrete object.

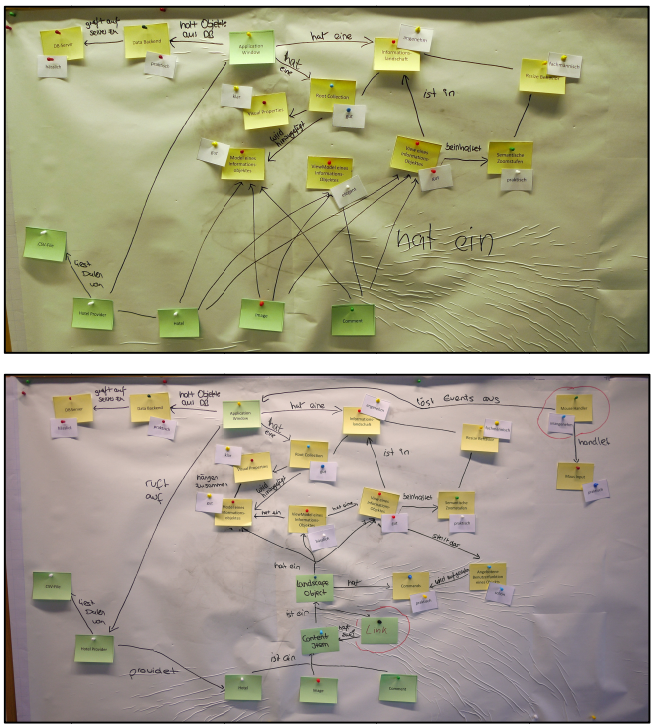

Figure 5: Concept Map session 1 and 2 from group 2

Extending and modifying maps over time: As discussed in the previous section, one main goal of the method is to make changes over time visually graspable. This, of course means, that the method can be most effective if there is time for such a longitudinal data gathering design which includes multiple sessions that build on top of each other. The idea here then is that participants continue to work and refine their concept map during each session, given that they have continued to use the API either with predefined tasks or for their real work (see fig. 5). First, they are asked to review all the concepts and connections and encouraged to think about currently unlabeled links and the map structure. Therefore, it is important to provide a flexible map background such as a whiteboard or the modified pin-board. Otherwise, changes are tedious and participants will be reluctant to do these. Changes always hint towards a changed or extended understanding of the API and thereby indicating potential problem areas as well as false positives one may come across in a usability test – some aspects of an API might just need some time to learn. This "change and update" procedure is also used for the adjective ratings and the problem areas. Regarding the former, changes are done by placing a new adjective on top of the old one, making it easy in the end to recapture the process. For the

analysis, the most interesting parts are when participants change from a negative to a positive adjective or the other way around, indicating a clear change of perception of this specific concept. Problem areas can be removed or just reduced in size as well as enlarged. Users just have to erase the drawing and change it accordingly. This gives researchers an understanding of the complexity of a problem which is furthermore supported by the thinking aloud. Again, being asked to do such changes often triggers users to explain these. The number of repeated sessions needed strongly depends on the complexity of the API, the nature of the task and the experience of the users. In our studies, we used at least five iterations to be able to grasp changes as well as a level of stabilization. The time duration mostly depends on the amount of time participants spend with the task in-between concept map sessions.

Besides these clear advantages for the longitudinal design, the method can already provide valuable input in cross-sectional designs as an addition for example to a usability test. Thereby, one could for example assess the knowledge about an API prior and after the test. Having such an externalization of the users' mental model furthermore can also enhance interviews with experienced developers – not to test their understanding but to understand their knowledge.

CASE STUDY

The Concept Map method has been developed in an iterative process which included two case studies. These were used to test out different variations of the method (e.g. table or vertical board, pre-defined or user-defined concepts).We used a framework for building zoomable user interfaces, which has been under development in our group, as a testbed during the studies. In this section, we present our second case study in detail. The idea of this section is to present a subset of our study results as empirical evidence about the usefulness of the method as well as more specifics about the possibilities during data analysis.

The ZOIL API

The **Z**oomable **O**bject-Oriented **I**nformation **L**andscape (ZOIL) API provides access to the ZOIL framework, which is deployed as a software framework written in C#/XAML for .NET & Windows Presentation Foundation (WPF). It provides programmers with an extensible collection of classes covering a wide range of functionality, e.g. ZUIs, client-server persistency, and input device abstraction. Basically, it serves as a toolkit for developing zoomable user interfaces in the context of reality based interaction and Surface Computing [21]. For the study, both the framework and the API were still under development and not "finished" products.

Study Design & Procedure

We conducted this study within a course about visual information seeking systems. The computer science students were given the task to create a prototype of such a system by using the ZOIL framework, which they had never used or seen before. However, they were familiar with the C# language. Eleven students participated and were split into five groups of two users (in one case three). This allowed us to apply a "discussing aloud" as a variation of thinking aloud during the concept map sessions for a better understanding of the users. We applied a longitudinal design over five weeks with five sessions (one session each week) of which the first was an introduction session. During the other four the participants were asked to create and modify their individual concept map. Each session lasted about 30 minutes. The overall programming task was split up into four milestones and after each session, the milestone for the next week was handed to the students. Thereby, we could resemble a realistic setting in which the task would require users to gain a deeper understanding of the API as time goes by.

Concepts: We created a master map of the ZOIL API prior to the study, which took two API developers about three hours. Based on this master map, we pre-selected 24 concepts. These focused on three aspects of the API/framework. The input handling, the MVVM (Model-View-ViewModel) pattern which is required to create objects in the zoomable canvas (the application window), and the attached behavior pattern, which allows users of the API to easily attach functionality to any object without having the object to implement it in its class hierarchy. Participants were not allowed to add concepts, as we wanted to control this variable for comparison between groups and the master map. We also provided "prototype" concepts which users were allowed to extend during the sessions in order to reflect their specific implementation of the given task. All API concepts were handed to the participants in the first session, and they were advised to use those concepts in the map to which they could refer to in any way. As students were learning the API and the framework during the task, we expected their understanding to change over time, which would then be reflected in their use of concepts on the map.

Procedure: The first session was used to present the programming task and explain the concept map approach. We did so by asking users to build a concept map of the "driver-car" interaction with the car representing the API and the driver representing the prototype. In the second session, users had worked with the API for one week and were asked to create a first concept map. We presented the materials, including the modified pin-board, different markers, the API concepts, the prototype concepts, and the adjectives. Usually, all participants started by flipping through the available concepts and using a table to get an overview. They then started to pin the known concepts onto the board and connect them through links. They were asked to discuss their decisions with their team mate but were advised that the researchers would not interfere with their task. After about 20 minutes, participants indicated that they had finished their map. They were asked to once again review the map and check any connections and labels.

Eventually, we asked them to assign the adjectives to the API concepts and mark any problem areas by drawing a red circle around the concepts before presenting them the next milestone for their programming task. In the following sessions, participants were first asked to review their existing map and change anything that they would now consider as a wrong reproduction of their mental map. The next step required them to extend the map, reflecting their programming work done during the week and add any additional concepts they had come across. Eventually, they again revisited the adjectives and the problem areas and made changes, accordingly. Every session was videotaped and stills were shot from each concept map at the end of a session.

Data analysis

In our understanding, a useful evaluation method has to be both flexible in terms of how it can be applied and in terms of possible measures, that can be derived from it. The Concept Maps method provides a large variety of possibilities for data analysis. In this section, we will illustrate the different steps needed and exemplify these with results from our case study.

Step 1 – digitizing the map: The method is designed in a way that the resulting maps can be represented in the GraphML standard (http://graphml.graphdrawing.org/) by using a graph editor such as yEd (http://www.yworks.com), with concepts and adjectives being represented as nodes and problem areas as groupings. In our case study, we reproduced one map from each session (the "final" map, example in fig. 6), thereby totaling 4 maps per group and 20 maps in total (duration for this step: 4h). It can also be interesting to include intermediate maps from within the sessions, if one is interested in this level of detail.

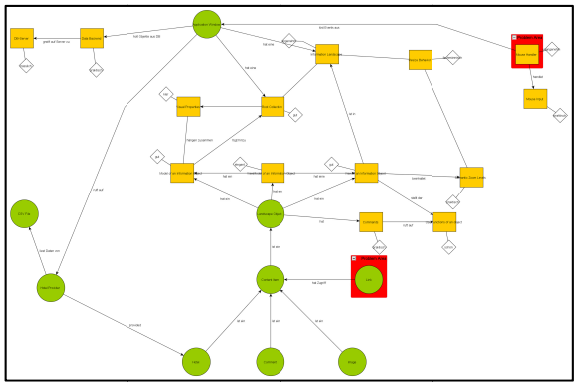

Figure 6: digitized map of group 2, session 2 (compare to Fig.5 for the still image)

Having this digital representation helps the analyst to identify interesting parts as the visual noise of a still photograph is cleared and aspects such as the problem areas stand out more clearly. Furthermore, we can use additional tools for graph analysis, which we will show in step 3.

Step 2 – General analysis of concepts, adjectives, and problem areas: Before analyzing individual maps in detail, a more quantitative and general approach can be helpful in identifying potential usability problems and misconceptions. At first, we can check which concepts have been added to the map during which session. We can cross-check this with the milestone for each session. If a concept has been added to the map although the part of the API it represents was not used until this point this could indicate that users were able to anticipate parts of the API which they had not used before. However, if a concept has been missing although the milestone clearly asked the participants to make use of this specific API part, this could indicate that they did not use or understand a necessary part of the API. We can also compare the use of concepts across participant groups and for example, identify, how similar groups are to each other and whether there are similarities regarding the use or disuse of concepts. In the case study, the concept "Landscape Handler" is easily identified as a problematic candidate. This concept refers to the part of the API that captures input events from different input modalities and forwards them to the zoomable canvas (which acts as a view). By comparing the groups we can see that only two of our five groups integrated this concept into their map, both during the second session. This is correct as the milestone for this second session was to integrate mouse input into the prototype. However, all other groups are missing this concept. When looking at the maps of those two groups who made use of the Landscape Handler, we can furthermore see that only one group used it correctly. In fact, the other groups connected the Mouse Handler concept directly to the view. While this understanding still resulted in a working prototype (most probably by copying code) the concept maps reveal that these users did not understand the abstraction layer this landscape handler introduces. The integration of further input modalities would therefore cause problems and require more time. So we can clearly state that this part of the API lacks some clarity and should be either refined or better documented.

A major benefit of the Concept Maps method compared to existing approaches is the ability to capture the dynamics of use, which also refers to the learning of the API and helps in avoiding "false positives". For example, looking at the adjective ratings from such a perspective can be very helpful. We can use a simple excel table to visualize which adjectives have been assigned to which concept at what point in time and whether this has changed at some point. Table 1 illustrates this for one of our groups in the case study (group 1). We can quickly see that the *View* and the *ViewModel* of the MVVM pattern were assigned with a negative adjective during the first session which was later on changed to a positive adjective (and corrected links between concepts), indicating the overcoming of a learning barrier. Other groups resemble this behavior, however in some cases, the negative adjective stays. In such cases, the knowledge of an API designer can then be very helpful to resolve conflicts between functional and non-functional requirements (the utility of the MVVM pattern vs. the

learning issues). In this table, we also visualized whether a concept was part of a problem area or not (the red frame around adjectives). The *DB Server* concept was assigned with the adjective "complicated" during the first three sessions and "confusing" during the last session. It was furthermore marked as being part of a problem area during the second and third session, but not in the fourth. We interpret the choice of adjectives and the problem area here in a way that the users found some way to get the *DB Server* to work, but even in the end were not quite sure how they managed it. So a negative adjective stayed, but the problem area disappeared. In this example, analyzing the final (working) code could lead to the wrong impression that the API was well understood ("false negative"). So we think that the concept maps allow for a more objective measure of understanding by looking at the dynamics of the learning process.

Group 1				
Concepts/Session	G1S1	G1S2	G1S3	G1S4
Semantic_Zoom_Levels	elegant	elegant	elegant	elegant
View_Information_Object	confusing	precise	precise	precise
Resize_Behvior	*empty*	competent	competent	competent
ViewModel	inconvenient	convenient	convenient	convenient
Model	easy	easy	easy	easy
Drag_Drop	pleasant	pleasant	pleasant	pleasant
InformationLandscape	beautiful	beautiful	beautiful	beautiful
SurfaceHandler	*empty*			
DBServer	complicated	complicated	complicated	confusing
RootCollection	good	good	good	good
LandscapeHandler		good	good	good
UserFunctions		beautiful	beautiful	beautiful
Commands		convenient	convenient	convenient
VisualProperties		*empty*	precise	precise
MouseInput		inconvenient	easy	easy
MouseHandler		inconvenient	inconvenient	inconvenient
SurfaceInput			easy	easy
DataBackend			*empty*	competent
RotateBehavior				

Table 1 – Adjectives assigned to concepts over time. Each column represents one session and each row one concept. Black = concept not yet added to the map, empty: concept added, but no adjective assigned.

We can also confirm here the already discussed issues with the input handler concepts, such as the Landscape Handler or the Mouse Handler. Only one group did not assign a negative adjective with either one of the two at some point. The others also frequently assigned problem areas to this part of the API (as in table 1), again indicating some clear misconceptions and usability issues.

Step 3: Visualizing changes over time: While the above analysis is in principal also possible by looking at the original maps, this part of the analysis requires the graphML based digital representations. This allows us to use graph analysis software to further decompose and analyze the links between nodes. As we are especially interested in changes over time, we find animations to be particularly useful [20]. The graph analysis research project *visone* (http://www.visone.info), which can be downloaded and used for free for non-commercial use, provides the necessary functionality. It easily allows displaying an animation between two or several graphs and highlights any changes. For example, nodes are animated on their way to a new position, new nodes are smoothly faded in, disappearing links are marked red before fading out and new links are marked green before becoming permanent. When analyzing one group in detail, this is already very helpful. We recommend using the results from step 2 as a focus point for the eye; then, play back and forth between the maps several times to identify the details. To obtain even more comprehensible animations to compare two groups with each other or the groups with the master map, there is another useful operation available, namely automatic dynamic graph layout. This is helpful, as each group as well as the master map, while maybe being semantically similar, may have very different spatial layouts that can make visual comparisons difficult. *visone* employs a framework for offline dynamic graph drawing, meaning that all states of a graph are known before a layout is to be computed, as is the case here. The underlying layout algorithm used is the energy-based technique *stress minimization* [17], which generally produces better results than comparable energy-based techniques and also scales very well [5]. In dynamic graph layout, the objective is to preserve the mental map of a viewer, i.e. parts of a layout, where the graph does not change much, should not alter over the course of time, therefore producing coherent layouts and facilitating easy comparison between successive states. However, layout quality in terms of faithful representation of structural features in the graph and maintaining dynamic stability are naturally opposed objectives in most cases. The algorithm employed in *visone* explicitly models this trade-off with an *anchoring*-approach [6] [16], penalizing point-wise deviations of a nodes' position from a reference position during layout calculation. A stability parameter $0<=\alpha<=1$ allows control between quality and stability. Using $\alpha=0$ corresponds to regular stress minimization for each individual layout, whereas $\alpha=1$ will result in the reference layout for each state.

Regarding the reference layout, there are three options available. We can use either one of the input graphs as reference, which is a sensible choice for comparisons with the master map or to compare to different groups at one point in time; take the previous state as reference for the current one; or compute an aggregated layout of the whole sequence as reference, which worked best for comparing a series of graphs of one group.

Figure 7 shows the original and rearranged maps for group 5 as well as the master map. While it is very difficult to visually grasp any differences between the original and the master map, the layout algorithm makes this a much easier task. We can easily see several differences but also similarities. The lower part of the graph stays more or less completely stable (the prototype concepts are missing in the master map). The upper part looks similar as well but the

animation reveals some differences. The *commands* concept is missing and the thereby connected *usefunctions of an object* concept is wrongly connected directly to *view* concept. This indicates that the *commands* were treated as a black box and usage could be enhanced and simplified with templates or code snippets. Besides, several *attached behavior* concepts are missing in group5. As the functionality existed in the prototype, they probably took advantage of these by copying existing code without understanding the underlying conceptual model. This can cause problems when new behaviors have to be designed that are not provided by the framework.

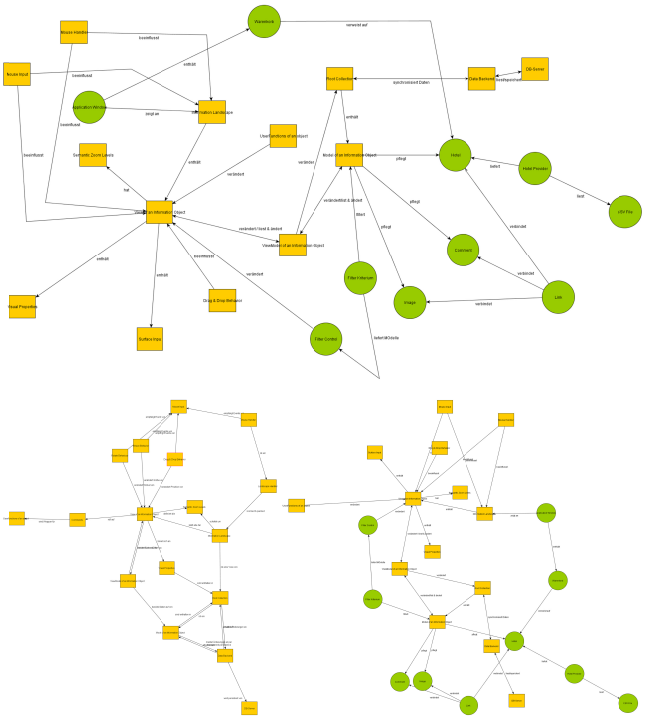

Figure 7: Top: group 5 orig. map, bottom left: master map, bottom right: group 5 map based on the stress minimization layout and the master map as reference (α = 75%)

Step 4/0: Video analysis: We intentionally did not discuss video analysis at that point, as this step is not really method specific. Nevertheless, we think that analyzing the video-taped session can reveal insights that are difficult to identify from a result based analysis as shown here. Participants often discuss the position and the linking of concepts in detail; sometimes argue about it, which obviously should be considered when analyzing the data. If time constraints don't allow detailed video analysis, note taking during the sessions can also help identify the important situations. In any case, the video data should be used to verify any claims.

Case Study Conclusion

We could identify three main issues with the help of the Concept Maps method within the ZOIL API. First, people have difficulties understanding the concept of different input handlers. Video analysis revealed that they misused the concepts as they expected a different functionality based on their prior experiences. Second, the MVVM pattern, while causing less trouble than expected, still led to some misconceptions and was widely rated with negative adjectives. In some cases, concepts that should connect to the View or the Model were connected to the ViewModel, indicating that users had problems to clearly separate these from each other. Third, we observed several group specific issues that did not cause problems on a general level with individual concepts. The insight gained here will nevertheless help to create a more usable interface. The duration of five sessions also revealed to be appropriate, as the concept maps had mostly converged up to the fourth session.

CONCLUSION

In this paper we have presented the Concept Maps method as a longitudinal approach to evaluate the usability of an API. The method is based on the idea that concept maps can be used to elicit and assess the knowledge users have of complex and abstract domains – for example science in the original use of concept maps or an API as in our case. We showed that the high-level view above code-level, which the Concept Maps demand and provide, makes it easier to recognize misconceptions and usability issues before they will lead to serious problems after deployment. The method provides a variety of means for data gathering and analysis, such as the possibility to rate concepts or indicate problem areas. The graph based structure of the maps allows the creation of digital representations of the maps which facilitates the use of graph analysis tools, such as *visone*. Using the Concept Maps method in a cross-sectional design can already greatly increase the benefit, e.g. of an API usability test. By allowing participants to create such a personal map and extend and modify it over time, changes in understanding are becoming visible to the researcher as well and learning barriers can be observed. Using graph analysis tools such as *visone* could also allow the application of similarity algorithms, thereby providing means to measure the level of agreement between participants and the API developers. Furthermore, the method can be applied in a more realistic task setting than what is possible in a usability test. While we have focused on the creation of the maps, they can also serve as helpful prompt during interviews, allowing participants to spatially locate problems, which was greatly appreciated in our studies. Last but not least, the Concept Map method can help participants in gaining a better understanding of the API as it asks them to reflect on their usage and as concept maps have proven to be useful learning aids in the past. While this certainly influences the method itself (as with many evaluation approaches), we see this as being of specific benefit as a training opportunity for participants that are from within an organization that develops an API for internal use and are meant to be end users as well. And finally, asking the API developers to create a master map can also help in identifying potential issues upfront.

In the future, it will be interesting to investigate how the method can also be combined with theoretical frameworks, such as Clarke's approach of using the cognitive dimensions. It might also be interesting to investigate the effect of using pre-defined vs. user-defined concepts in detail. While we have comprehensively discussed how to use the Concept Maps method and the possibilities during the analysis of the data, we think that one significant benefit of the method is its flexibility in terms of materials and data gathering techniques that are included. Eventually, it opens up a huge design space for future research on how to elicit knowledge and understanding of an API which can be beneficial for analyzing the usability of one specific API as well as for the design of future APIs.

REFERENCES

1. Ballagas, R., Memon, F., Reiners, R., and Borchers, Jan. iStuff mobile: rapidly prototyping new mobilephone interfaces for ubiquitous computing. In *Proc. CHI '07* (2007), ACM Press, 1107-1116.
2. Barksdale, J. and McCrickard, D.S. Concept Mapping in Agile Usablity: A Case Study. In *Proc. CHI 2010 EA* (2010), 4691-4694.
3. Beaton, J.K., Myers, B.A., Stylos, J., Jeong, S., and Xie, Y. Usabiltiy evaluation for enterprise SOA APIs. In *Proc. of SDSOA '08* (2008), ACM Press, 29-34.
4. Bloch, J. How to write a good API and why it matters. In *LCSD workshop at OOPSLA* (2005). Keynote, online http://lcsd05.cs.tamu.edu/#keynote.
5. Brandes, U., Pich, C. An experimental study on distance-based graph drawing. In *Proc. 16th Int. Symp. on Graph Drawing* (2008), Springer, 218-229.
6. Brandes, U. and Wagner, D. A Bayesian paradigm for dynamic graph layout. In *Proc. 5th Int. Symp. on Graph Drawing* (1997), Springer, 236-247.
7. Clarke, S. Measuring API Usability. *Dr. Dobbs Journal* (May 2004), 6-9.
8. Courage, C., Jain, J., and Rosenbaum, S. Best practices in longitudinal research. In *Proc. CHI '09 EA* (2009).
9. Cwalina, K. and Abrams, B. *Framework design guidelines*. 2005.
10. Daughtry, J.M, Farooq, U., Myers, B.A., and Stylos, J. API Usability: Report on Special Interest Group at CHI. *Software Engineering Notes* (July 2009).
11. Daughtry, J.M., Stylos, J., Farooq, U., and Myers, B.A. API Usability: CHI'2009 Special Interest Group Meeting. In *Proc. CHI'09 EA* (2009), ACM Press.
12. de Souza, C.R.B., Redmiles, D., Cheng, L., Millen, D., and Patterson, J. Sometimes You Need to See Through Walls - A Field Study of Application Programming Interfaces. In *Proc. CSCW* (2004), 63-71.
13. Ellis, B., Stylos, J., and Myers, B.A. The Factory Pattern in API Design: A Usability Evaluation. In *Proc. ICSE '07* (2007), ACM Press, 302-312.
14. Eppler, M.J. A comparison between concept maps, mind maps, conceptual diagrams, and visual metaphors as complementary tools for knowledge construction and sharing. *Information Visualization*, 5, (2006), 202-210.
15. Farooq, U. and Zirkler, D. API Peer Reviews: A Method for Evaluating Usability of Application Programming Interfaces. In *Proc. CHI 2010* (2010).
16. Frishman, Y. and Tal, A. Online dynamic graph drawing. *IEEE Trans. on Visualiz. and Comp. Graphics*, 14, 4 (2008), 727-740.
17. Gansner, E., Koren, Y., and North, S. Graph drawing by stress majorization. In *Proc. 12th Int. Symp. on Graph Drawing* (2004), Springer, 239-250.
18. Green, T.R.G and Petre, M. Usability Analysis of Visual Programming Environments: A Cognitive Dimensions Framework. *Journal of Visual Languages and Computing*, 7, 2 (1996), 131-174.
19. Heer, J., Card, S.K., and Landay, J.A. prefuse: a toolkit for interactive information visualization. In *Proc. CHI '05* (2005), ACM Press.
20. Heer, J. and Robertson, G. Animated Transitions in Statistical Data Graphics. *IEEE Trans. Visualization & Comp. Graphics*, 13, 6 (2007), 1240-1247.
21. Jetter, H.-C., Gerken, J., Zöllner, M., and Reiterer, H. Model-based Design and Prototyping of Interactive Spaces for Information Interaction. In *Proc. of Human-Centred Software Engineering (HCSE)* (2010).
22. Klemmer, S.R., Lie, J., Lin, J., and Landay, J.A. Papier-Maché: toolkit support for tangible input. In *Proc. CHI ' 04* (2004), ACM Press.
23. Ko, A.J., Myers, B.A., and Aung, H.H. Six learning barriers in end-user programming systems. In *Proc. IEEE Symp. on Visual Languages and Human-Centric Computing* (2004), IEEE, 199-206.
24. McClure, J.R., Sonak, B., and Suen, H.K. Concept Map Assessment of Classroom Learning: Reliability, Validity, and Logistical Practicality. *Journal of Research in Science Teaching*, 36, 4 (1999), 475-492.
25. McLellan, S.G., Roesler, A.W., Tempest, J.T., and Spinuzzi, C.I. Building more usable APIs. *IEEE Software*, 15, 3 (1998), 78-86.
26. Myers, B., Hudson, S.E., and Pausch, R. Past, present, and future of user interface software tools. *ACM Trans. Computer-Human Interaction*, 7, 1 (2000), 3-28.
27. Novak, J.D. and Gwon, D.B. *Learning How to Learn*. Cambridge, UK, 1984.
28. Taris, T.W. *A primer in longitudinal data analysis*. SAGE Publications, London, 2000.
29. Tulach, J. *Practical API Design: Confessions of a Java Framework Architect*. 2008.