Towards API Usability Engineering as a Software Engineering Paradigm

Christian Klauß

Software Technology Group
Technische Universität Darmstadt
Hochschulstr. 10
64289 Darmstadt
klauss@st.informatik.tu-darmstadt.de

Abstract: APIs and their production and consumption are part of the core of most software development today. From the perspective of software developers APIs can be interpreted as a type of user interface used to solve programming tasks. In this context a small number of publications has investigated the analysis, evaluation and improvement of the usability of APIs under the term of API usability in the past. We propose to investigate if and to what extent a new paradigm of API usability engineering can be a useful complement to established paradigms of software engineering. To this end we present the first draft of a framework that aims to capture the dimensions of API usability as a domain for research and application and might serve as a basis for scoping the proposed field of API usability engineering.

1 Introduction

APIs play a central role in almost every software development project today. Creating »good« APIs thus becomes an important goal for software development [Blo06]. The quality and success of APIs can be understood by several properties. Aside from the basic requirement of allowing developers to solve the functional requirements of a given task/domain, APIs can vary in properties like availability (e.g. platform, license, cost), productivity, market penetration (which is of interest in the context of marketable public APIs) and properties corresponding to non-functional requirements of software development (e.g. security, maintainability) and more. We assume that API usability is one of they key attributes of a high-quality, successful API. That means investing into making APIs more usable has a measurable and significant positive impact on quality and success of an API (that outweighs the costs of doing so) in most contexts. The potential of positive impact of improving API usability is illustrated in works like [SGB+08].

To our knowledge there is no widely agreed on definition of either API or API usability. We assume the term API to include a relatively wide array of software component artefacts that define operations and underlying types such as programming libraries, Web services, frameworks or toolkits. In a previous work API usability has been defined on the basis of ISO 9241-11 as »the extent to which an API in its role as a user interface for software developers during production and maintenance of software structures can be used by given software developers to solve problems effectively, efficiently and satisfyingly« [Kla11].

2 Towards API Usability Engineering as a Software Engineering Paradigm

The aim of this paper is twofold. First of all we would like to spread awareness for API usability as a cross-cutting concern of software engineering research and application. Secondly we propose to start a process of aggregating and formalizing the results of API usability research that might ultimately lead to an established paradigm[1] of API usability engineering. To this end we will introduce the field of API usability very briefly and describe some of the challenges that lie ahead. Finally we present the first draft of our API Usability Domain Description Framework as a possible basis for scoping the proposed field of API usability engineering.

2.1 Extent of Published Research on API Usability and Selected Results

A paper by McLellan et al. from 1998 is widely cited as the first instance of the concept of usability being applied to APIs [MRTS98]. Starting in the middle of the 2000s, a growing but small number of works was published by a small research community, that can be considered being responsible for establishing the the field of API usability research[2]: an overview can be found in a report on a special interest group at CHI from 2009 [DFMS09]. In 2012 a semi-systematic literature review classified 28 papers as belonging to the field [BFHM12]. Since then a small number of works referring to the concept of API usability has been published each year.

Some of the notable results include an observed negative impact of design patterns (factory pattern) [ESM07] and coding practices (requiring parameters in object constructors) [SC07] on API usability. A large portion of papers focuses on methods for evaluating API usability, for example: [CB03, FWZ10]. Further sample research results include tool improvements [PH09], analysis of APIs to produce generalizable insights and/or heuristics [RD10] and a positive impact of static typing on API usability [EHRS14].

[1]In the context of software engineering we understand the term paradigm as a set of theories, methods, tools, processes/activities and tasks/problems. Usually a paradigm emphasizes one or more elements of this set. For example security engineering emphasizes the security aspects of a software system, while component-based software engineering focuses on reuse and separation of concerns.

[2]The API Usability project of the Carnegie Mellon University can be considered the most prominent part of this community. For more information see `https://www.cs.cmu.edu/~NatProg/apiusability.html`.

2.2 Challenges and Agenda

As shown there is a small but solid body of work on API usability. We consider this to be the first stepping-stone for establishing a paradigm of API usability engineering. There are several challenges that need to be tackled.

First of all there is a lack of a agreed upon terminology and concepts. Not only does this lead to communication problems, but also makes non-consideration of related research results more probable. Establishing a common terminology would belong to the first tasks on the proposed agenda that needs to be solved. This is tightly connected to the need for a well-defined scope of the field. The concept of API usability, as we understand it, has, not only but also due to its human factor, many intersections and interactions with other fields of research and application in software engineering. This ranges from technical issues (like language design) to human-centered issues (that lend themselves to interdisciplinary approaches, for example integrating results from classic usability engineering) and those in between. To highlight this issue, consider for example tools that act as an intermediary between developers and APIs like code recommendations in modern IDEs. Such tools can shape how developers explore or interact with an API and thus can have an impact on API usability. Tools and publications that improve code recommendation that make no direct reference to API usability (for example [BMM09]) might still need to be considered.

Thirdly, we believe that API usability engineering would need a strong empirical foundation. We believe that qualifying, substantiating or disproving claims of the positive effects of improving API usability (also in a more formalized manner) should occur early in the agenda. This aligns well with a positive trend towards a more wide-spread use of empirical experiments in software engineering (though there is a noticeable lack of experiments in industry settings, which might produce more reliable results [DJM13]).

2.3 API Usability Domain Description Framework

The API Usability Domain Description Framework aims to serve two purposes. For one, it tries to establish a basis for describing and scoping API usability engineering as a field. Furthermore it tries to help structuring thoughts and communication for practical and research purposes like analyzing problems and generating and describing research questions.

The framework defines 11 dimensions and 4 aspects. Dimensions structure properties of problems, questions, results and solutions (called »elements« hereafter) of the API usability domain. Aspects, which represent cross-cutting, abstract categories, can be used to modify and/or link dimensions.

Two results/problems can serve as an example for what constitutes elements of this domain: (i) There is some evidence that static typing systems (and API documentation) have a positive impact on API usability [EHRS14]. This constitutes a trade-off in contexts where usage of dynamically typed languages might be considered preferable. (ii) The literature suggests that there might be a negative influence of established design patterns and coding practices on API usability [ESM07, SC07]. This constitutes a trade-off between achieving optimal API usability and other goals.

Dimensions

- Cognition & Mental Models: Captures in what way an element is concerned with cognitive processes and states, that are relevant to using APIs. This concerns exploring, comprehending and learning APIs in particular. Also includes analogous machine processes (e.g. program comprehension).
- Development Tools & Automation: Describes to what extent an element is related with the tools used by developers, particularly those that can be considered as an intermediary between developer and API like IDEs and their recommendation tools.
- Evaluation: Captures in what way an element is related to the development, validation and improvement of methods for evaluating API usability and the concrete use of such methods. Development can also mean adapting methods from other fields such as usability engineering.
- Information Resources: Describes to what extent an element is concerned with external information resources for given APIs. This includes documentation, Q&A sites (such as Stack Overflow) and tutorials, for example.
- Life Cycle: Describes in what way an element is related to the life cycle of an API. The life cycle can be described in multiple ways. An intuitive example are the classic stages of the waterfall process.
- Management & Business: Captures in what way an element is concerned with software management and business considerations that impact handling of APIs. This includes classical topics of software management like project management or quality management and other topics like API marketing and the API economy.
- Product Properties & Domain: Captures to what extent an element is related with the properties of a software product that is being developed by using APIs and the domain that the respective product is being developed for. Properties include non-functional quality characteristics of the product like performance and security.
- Programming & Activities: Describes in what way an element is concerned with the different activities of programming (e.g. writing, reading, testing, refactoring) and further activities during the software development process.
- Roles & Actors: Captures to what an extent an element is related to the different roles and actors that are found in the software development process. Typical roles/actors are developers, managers, end-user-programmers, for example.
- Surface & Programming Languages: Captures in what way an element is concerned with the surface of an API, in the sense it being an user interface for developers, and properties of programming languages. The surface consists of API artefacts that developers interact with, like class names or method signatures.

- Structure & Instantiation: Describes to what extent an element is related to the concrete API structure, as opposed to the surface, and its instantiation. This pertains to topics like framework instantiation and scaffolding, platform stacks or availability of the source code of an API implementation.

Aspects

- Interaction & Relation: Captures the aspect of interactions and relations between the dimensions.
- Impact: Describes the aspect of the impact on API usability and its quantification. It expresses what impact, be it negative or positive, an element has on API usability. Its value thus determines to what an extent an element can be considered being part of the API usability domain.
- Patterns & Heuristics: Captures the aspect of patterns and their analysis in relation to the dimensions.
- Perspective & Method: Describes the aspect of the different possible perspectives and methods in relation to the dimensions. Possible perspectives include usage of APIs compared to API design or API (usability) research. Method especially pertains to methodology in research.

For a very brief example how elements relate to dimensions we can revisit problem (i) mentioned above. The relation to the dimensions Surface & Programming Languages (influence of static type system) and Information Resources (influence of API documentation) is probably the most obvious. But one could also consider how usage of code recommendation tools, captured by the dimension Development Tools & Automation, influences these phenomena, for example.

3 Summary

Based on the assumptions about a need for high-quality APIs and the major role API usability plays in this regard, we propose inquiry into establishing API usability engineering as a paradigm in software engineering. For this purpose we present the first draft of a framework that structures elements of API usability as a domain for research and application and might serve as a basis for scoping the proposed field of API usability engineering.

References

[BFHM12] Chris Burns, Jennifer Ferreira, Theodore D. Hellmann, and Frank Maurer. Usable results from the field of API usability: A systematic mapping and further analysis. *2012*

IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC), pages 179–182, September 2012.

[Blo06] Joshua Bloch. How to design a good API and why it matters. In *Companion to the 21st ACM SIGPLAN conference on Object-oriented programming systems, languages, and applications*, pages 506–507, New York, New York, USA, 2006. ACM Press.

[BMM09] Marcel Bruch, Martin Monperrus, and Mira Mezini. Learning from examples to improve code completion systems. In *Proceedings of the the 7th joint meeting of the European software engineering conference and the ACM SIGSOFT symposium on The foundations of software*, pages 213–222, New York, New York, USA, 2009. ACM Press.

[CB03] Steven Clarke and Curtis Becker. Using the cognitive dimensions framework to evaluate the usability of a class library. In *Proceedings of the First Joint Conference of EASE PPIG (PPIG 15)*, number April, pages 359–366, 2003.

[DFMS09] John M. Daughtry, Umer Farooq, Brad a. Myers, and Jeffrey Stylos. API Usability: Report on Special Interest Group at CHI. *ACM SIGSOFT Software Engineering Notes*, 34(4):27–29, July 2009.

[DJM13] Oscar Dieste, Natalia Juristo, and Mauro Danilo Martinc. Software industry experiments: A systematic literature review. In *Proceedings of the 1st International Workshop on Conducting Empirical Studies in Industry*, pages 2–8, 2013.

[EHRS14] Stefan Endrikat, Stefan Hanenberg, R Robbes, and A Stefik. How do API documentation and static typing affect API usability? In *Proceedings of the 36th International Conference on Software Engineering*, pages 632–642, 2014.

[ESM07] Brian Ellis, Jeffrey Stylos, and Brad Myers. The Factory Pattern in API Design: A Usability Evaluation. In *29th International Conference on Software Engineering (ICSE'07)*, pages 302–312. IEEE, May 2007.

[FWZ10] Umer Farooq, Leon Welicki, and Dieter Zirkler. API usability peer reviews: a method for evaluating the usability of application programming interfaces. In *Proceedings of the 28th international conference on Human factors in computing systems*, pages 2327–2336. ACM, 2010.

[Kla11] Christian Klauß. *Annotations- und werkzeuggestützte Verbesserung von API-Usability*. Diploma thesis, Technische Universität Dresden, 2011.

[MRTS98] S.G. McLellan, A.W. Roesler, J.T. Tempest, and C.I. Spinuzzi. Building more usable APIs. *IEEE Software*, 15(3):78–86, 1998.

[PH09] David M. Pletcher and Daqing Hou. BCC: Enhancing code completion for better API usability. *2009 IEEE International Conference on Software Maintenance*, pages 393–394, September 2009.

[RD10] Martin P. Robillard and Robert DeLine. A field study of API learning obstacles. *Empirical Software Engineering*, 16(6):703–732, December 2010.

[SC07] Jeffrey Stylos and Steven Clarke. Usability Implications of Requiring Parameters in Objects' Constructors. In *29th International Conference on Software Engineering ICSE07*, pages 529–539, Minneapolis, MN, 2007. IEEE.

[SGB$^+$08] Jeffrey Stylos, Benjamin Graf, Daniela K. Busse, Carsten Ziegler, Ralf Ehret, and Jan Karstens. A case study of API redesign for improved usability. In *IEEE Symposium on Visual Languages and Human-Centric Computing*, pages 189–192, Herrsching am Ammersee, September 2008. IEEE.