Developing Usable APIs with XP and Cognitive Dimensions

Rahul Kamal Bhaskar, Craig Anslow, John Brosz, Frank Maurer
Department of Computer Science
University of Calgary
Calgary, Canada
{rbhaskar, canslow, jbrosz, fmaurer}@ucalgary.ca

***Abstract*— Developing a usable Application Programming Interface (API) is a complex and expensive task. Two major factors play important roles on the usability of an API: the design and resources (e.g. documentation, tutorials). API Developers typically evaluate the usability of an API after implementation that results in refactoring tasks if an API lacks usability after development. This refactoring could be avoided if evaluation were continuously conducted while development. This paper explores a new combined process for building usable APIs that combines concepts from a usability evaluation method (Cognitive Dimensions Framework) and an Agile development methodology (eXtreme Programming). We explored the effectiveness of this combined process by implementing a web-based API and conducting a user study. The findings from our evaluation indicated that the new process helped in designing and building a usable API, but ignored some concerns related to resources.**

Keywords*— *APIs, Cognitive Dimension Framework, Usability, XP.

I. Introduction

The process of software development has evolved over the time instead of implementing code from scratch, developers reuse code or functionalities available through sets of routines, and protocols called Application Programming Interfaces (APIs) [1–3]. To reduce costs and effort to develop software, developers often use APIs. Benefits of using an API are mostly seen if an API make it easy for a developer to understand and reuse its functions that indicates usability of an API [2]. An aspect that plays a major role in the decision of whether an API will be used is the usability of an API [4]. APIs can have various usability issues that can be generated due to design, resources, and technical constraints [2]. Designers can evaluate the usability of an API through different methods such as API peer review [5], API profile dimensions [6], Cognitive Dimensions (CDs) Framework [7, 8], and text analysis [9]. Building an API gets more complex when requirements continually evolve and there are time constraints [10]. Agile methodologies accommodate changing requirements to build software under time pressure with the help of effective planning [10]. There are different Agile methodologies such as eXtreme programming (XP) [11], [12] and Scrum [13].

Literature suggests that the design decision and development phase during software development is the most appropriate phase to consider the usability of an API [14, 15]. This paper explores a development process devised by combining a development process with an API usability evaluation method which can help in building a usable API. As a case study this development process was used to implement an API to explore the impact on the development process. We conducted a user study on the resulting API to examine its usability and to understand contribution of the development process in making the case study API usable.

II. Related Work

An API should be the result of a good development process, and every step in the process should offer the opportunity for improvement [16]. The design and development phase is the most appropriate phase to consider the usability of an API [14, 15]. Zibran et al. pointed out that "*API designer and developers need a good understanding on API usability and apply those usability concepts during design and development phases, so that they can minimize the maintenance difficulties caused by the usability issues associated with such APIs*" [17]. Design and evaluation are tightly coupled with all stages of software design [15, 18].

Different research has pointed out that the usability evaluation when made part of the development process can lead to usable software [19–21]. CRUISER a development process proposed by Memmel et al. [19], emphasizes an increased involvement of stakeholders and developers by using prototypes and scenarios. Mclnerney and Maurer showed that User Centric Design and Agile methods can coexist and can result in better UI design [22]. Singh proposed the U-SCRUM methodology as a variant of SCRUM that can be used to improve usability. U-SRUM suggests using two product owners in order to improve usability (one for responsibility for function implementation and other for usability). Ahmad et al. proposed a process called Agile Usability software Development Life Cycle to make interfaces more usable [21].

Our work is inspired from past research accomplished in designing usable software by combing usability evaluation aspects with the development process. We designed a development process that combines usability methods with the development process to design a usable API. In order to find an appropriate development process and usability method which when combined can result in a usable API, we explored different issues that developers face during development of an API. On exploring the literature we found the following issues that were faced by developers: frequently changing

978-1-5090-0252-8/16/$31.00 ©2016

requirements, issues related to source code such as easy to learn, readable code, hard to misuse, easy to extend, naming convention, design patterns, abstraction level [1, 16, 17, 23, 24], method placement (i.e. on which class or classes methods are placed) and method calls [25]. These issues were analyzed and used for searching a usability evaluation method and development process, which can address these issues.

When searching for the development process, we explored different Agile methodologies and found that XP is appropriate when requirements changes frequently [11, 12]. XP also emphasize more on the development practices (e.g. refactoring, unit test, test driven development, continuous integration) which helps in organizing development work [11].

When searching for an evaluation process, which can address a wide variety of issues, we found that the Cognitive Dimensions (CDs) framework is appropriate. Previous research shows that CDs has been successfully used in the past to evaluate different software [7] and APIs [4, 5, 8, 26]. The CDs has multiple dimensions that can be used to evaluate different usability issues of an API [27]. The dimensions can also be used as shared vocabulary to generalize results of the usability evaluation, so that other developers can use findings from other studies to develop their own API [28]. This paper describes a development process that combines a subset of XP practices with the CDs framework to build a usable API. In the remainder of this paper, "API-designer" represents the programmer who developed the API, "API-user" for the developers who use the API and "end user" for the person who uses the application based on the API.

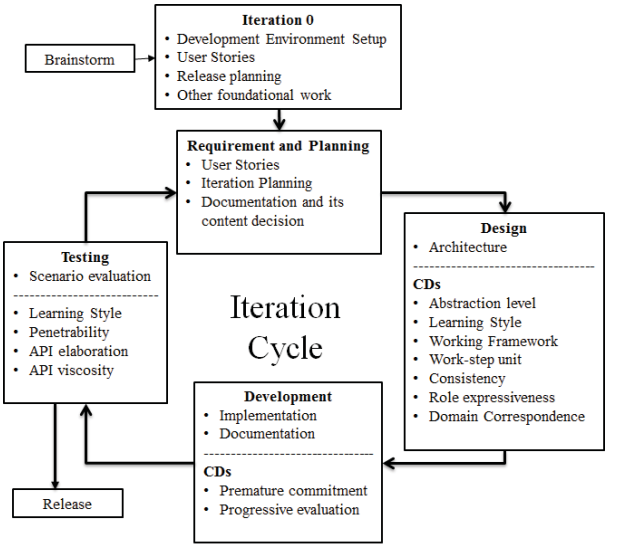
III. Development Process: XP + CDs

We designed a development process (**Figure 1**) by combining a subset of XP practices (e.g. incremental planning, release and iteration planning, user stories, short iteration, refactoring and unit tests) with the CDs framework.

In XP + CDs, all dimensions of the CDs framework are applied to the output of each phase of the development process to evaluate usability in the different phases for each iteration (design, implementation, testing, and evaluation). There is no check of usability during requirements phase, as there are no decisions regarding the design of an API architecture nor is any coding taken place during this phase. Usability evaluation in every phase of development ensures the API-designer that the chosen design decisions thus far will have limited number of usability issues. This approach will help in addressing usability issues when it is generated and avoid future unwanted consequences due to usability issues. This will likely result in a more usable API at the time of release. The CDs framework does not specify which dimensions can be used for the evaluation for certain outcomes (e.g. function name, signature or abstraction level) of the development process. Therefore, we decided to apply all the dimensions during all the phases of development so that usability issues can be discovered and find which dimensions are appropriate for which phase.

XP + CDs is made up of seven phases. The first phase, **Brainstorming**, where the API stakeholders make decisions regarding the API ideas and the features. In **Iteration 0** the API-designer creates user stories, performs release planning,

Figure 1 XP + CDs – Combined development process, combining the XP and CDs frameworks.

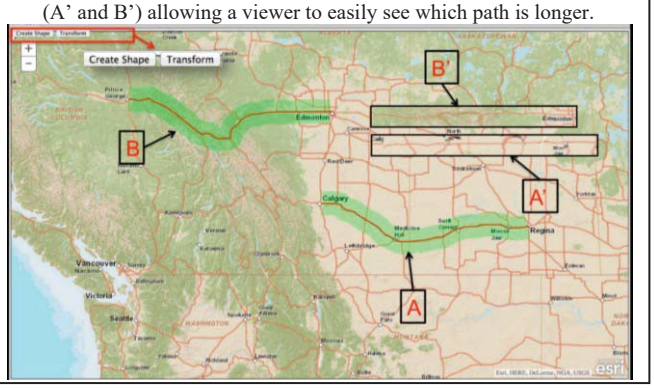


decides and prepares the development environment (such as programming languages and IDE) and other setup work if required, such as licensing development tool. The next step is the **Iteration Cycle** that is made up of several software development phases. The **Requirements and Planning** phase is used for planning each iteration (i.e. selecting user stories and breaking each user story into smaller tasks [12]), creating or updating user stories as new requirements are obtained. In the **Design** phase, the API architecture is created (such as classes, functions, function signatures, names, contents of functions and classes), and resources (e.g., function descriptions, and code snippets). Furthermore in the design phase, the outcome of this phase (i.e. names of functions and classes, content of function and classes, resources) are evaluated using the CDs to find usability issues. Through the **Development** phase the API-designer writes code based on the design, unit tests, and prepares documentation. Furthermore the outputs of this phase, source code and documentation, is evaluated using all of the CDs to detect usability issues. In **Testing** scenarios are created based on the tasks that an API-user might want to accomplish that was implemented in this phase. These scenarios help the API-designer to perform acceptance testing and evaluate the API from the API-user perspective. These scenarios are also evaluated against all dimensions of the CDs. This cycle repeats for each iteration until all features are implemented and usability issues are addressed. In the **Release** phase, API-designer makes the API available for use.

IV. GIST-API

To demonstrate that XP + CDs is helpful in building a usable API, designed a case study API, the GIST-API, using this process. The GIST-API is a web-based visualization API that implements advanced graphics techniques for transforming graphical images on the fly (transmogrification [29]) for web-based geospatial applications.

Figure 2. An application created with the GIST-API. In this screenshot (A and B) has been transformed into two easily comparable rectangles (A' and B') allowing a viewer to easily see which path is longer.



A. Demo and Application Interface

The end-user of the application can compare two routes and determine which one is longer. The end-user clicks the create shape button which creates the green highlighted region (i.e. A in **Figure 2.**) by moving the mouse cursor on the web page. These green highlighted regions are the input shapes for transmogrification. After this the end-user clicks the transform button, which generates the transformed image (i.e. A' in **Figure 2.**) to represent selected road route of arbitrary geometry into the rectangular geometry. The end-user repeats the tasks accomplished for the other road route (i.e. B in **Figure 2.**), which generates another transformed image (i.e. B' in **Figure 2.**). Now the end-user can move the transformed images side-by-side and check the length of the rectangles (A' and B'). Finally, by visually comparing the length of the rectangle shapes the end-user can answer that A' (i.e. path A) was longer than B' (i.e. path B). The end-user can repeatedly perform these steps to compare multiple routes.

B. Impact of the development process on GIST-API design

This section presents observations from the case study on the development process designed by combining a subset of XP practices with CDs. During the case study, even though all the CDs were applied on each phase, it was observed that not all the CDs have an impact on the GIST-API's design.

In the *design phase*, it was noticed that the CDs had a significant impact on the different elements of the API's architecture such as the class organization, naming methods, and deciding the amount of tasks that can be accomplished using a particular function. For example, the name of the function was verified in the case study using three CDs (i.e. **consistency, role expressiveness,** and **domain correspondence**). **Role expressiveness** helped in checking that the "particular" selected name for the function expectations was derived from the function name. **Domain Correspondence** helped in showing how clear function names map to the domain such as the "Transmogrifer()" function which represents the transmogrification technique. Function signatures were validated using the consistency dimensions. **Consistency** helped in checking that the implementation was coherent throughout the development phase. Furthermore, it was also noticed that the **abstraction level** and **working step unit** impacted the API design as this helped in checking the goal that can be achieved by the use of a function from the API. Finally, in this phase **working framework** helped in deciding classes and the functions.

For the *development phase* it was observed that all of the CDs had a significant impact on the overall design of the GIST-API and documentation. While building the GIST-API using the development process it was observed that three CDs (**consistency, role expressiveness,** and **domain correspondence**) were used more often than other dimensions (e.g. **premature evaluation** was only used once during the development phase). During this phase the CDs helped in finding possible usability issues, and reduced the potential refactoring that could have generated after the development due to the issues such as, function names, class organization and missing documentation, which was avoided. A few improvements were found while applying **progressive evaluation**. During this phase, the **premature commitment** dimension can also be verified but in the GIST-API there was no situation where an API-user has to make any assumptions in the GIST-API while implementing any feature.

The *testing phase* is the last phase of the iteration cycle. During this phase, it was found that the API design and documentation could be validated using the four CDs (i.e. **learning style, penetrability, API elaboration, and API viscosity**). For example, while evaluating the function input parameter the **API elaboration** dimension helped in validating whether documentation had sufficient descriptions or not.

The subset of the XP practices not only helped in accommodating change but also helped in organizing the development process. Iteration and release planning helped in defining priorities of the user stories. Short iterations made the development process fast and productive. User stories helped in designing features from customer perspective. Refactoring helped in improving the design on a continuous basis. Unit testing helped verify that the code is working as expected.

V. User Study

We conducted a user study to investigate the perceived effectiveness and usability of the GIST-API. Before conducting the user study a pilot study was conducted to identify major usability issues and to test the study protocol. Based on the pilot study we improved the GIST-API design and improved the resources (i.e. documentation, code snippets, and videos). For the main user study we recruited 16 participants (referred to as P1-P16), all of whom were Computer Science graduate students from the University of Calgary. All the participants were offered an honorarium of CAD $15 for participating in the study. The study had four steps: pre-study questionnaire, training, programming, and post-study questionnaire. During the pre-study questionnaire the participant's demographics were collected. Training involved a video introducing the transmogrification concept and showing demo code illustrating how to develop a prototype with the API. In the programming tasks research step, participants were asked to complete programming tasks using the GIST-API. The post-study questionnaire step collected participants' feedback regarding the usability of the API and suggestions on how to improve the usability.

Table 1. Data Analysis Table for the main user study.

Categories	Codes	Participants
API Design	Easy to use function	P1 – P7, P9, P10, P13, P15, P16
	Self-Documenting name	P1, P3, P10, P13, P15
	Keyword search	P2, P6, P9, P14
	Lack intuitiveness	P2, P10, P11
	Few top level variables	P4
Role of Knowledge	Programing language knowledge helped	P15
	Lack domain knowledge	P5, P6, P8- P10, P14, P15
API Debugging	Error message description is self-sufficient for solving the error	P1-P6, P8, P10, P13-P16
	Line number for error would be helpful	P10, P13, P15, P16
	Bug hindered usability	P6, P7

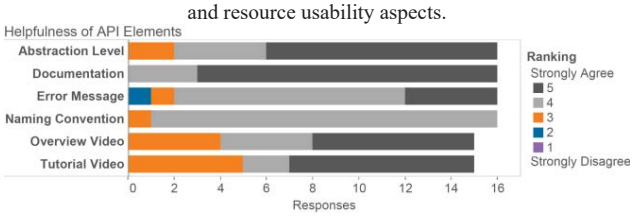
The study data was processed and transcribed manually (i.e. listening to recorded interviews) and we created a summary of the activities, comments and suggestions observed during the study. **Table 1** was created from the analyzed activities, which has categories, codes, and the participants.

API Design: It was observed that 12 participants mentioned *easy to use function* as they did not face problems in understanding the function names and tasks they were performing. Five participants mentioned function names are *self-documenting name* (i.e. names are sufficient description about the task that function performs). **Figure 3**. shows the responses from the questionnaire, where 14 participants perceived that the GIST-API has good naming conventions. Four participants preferred *keyword search* in the function list. The keywords were the words that the API-users can think of regarding the function they wanted to use. For example, an API-user was looking for a feature in the API that deals with colour and the keyword will be "colour". The findings suggests that the naming conventions based on the domain and role, ease the API understanding. As these types of function names helped the participants during the study in locating a function in the documentation by searching for the keywords. Based on the findings from the study no conclusion can be made grouping functions into classes helped API-users or not. Except P4 none of the participants made any comments on the classes.

Role of knowledge: Transmogrification was a new concept for all the participants, seven participants explicitly mentioned that the *lack of domain knowledge* slowed their progress during the beginning of the study. According to these participants it took time for them to get used to the transmogrification concept. Only one participant (i.e. P15) commented that *previous programming language knowledge* (i.e. JavaScript, and GIS) helped in learning the API syntax. The study findings implied that domain knowledge could play an important role in reducing the learning time required by the API-users.

API Debugging: Error messages were added in the GIST-API to help the API-users in debugging errors. According to the participants, the *error message was self-sufficient for debugging errors* (i.e. documentation was not required to be checked by the participants for suggestions to solve the bug). P4 commented, "*Debugging isn't straightforward, as JavaScript code runs even if the code has an error, but this error message helped me finding errors.*" According to P4, the error messages helped in locating mistakes made while using the API. P10 and P16 suggested to show line numbers where an error is generated. P6, P7, and P9 liked showing of error. The findings of the study suggest that adding error messages makes debugging easy as error can be traced and corrected.

Figure 3. Participants' responses regarding API Design and resource usability aspects.



VI. Discussion

Objective of the study was to explore usability of the GIST-API which helps in understanding whether XP + CDs can help in designing usable API. During the study data analysis, we did not find usability issues in the GIST-API design. The study findings demonstrated that participants were satisfied with the API. Participants found that the resources and the content were helpful, but not all resources resulted from the development process. Few of the resources were implemented after receiving feedback from the pilot study (such as function list, flow chart, jargon descriptions). The study findings suggests that the development process resulted into a usable API design but was not able to address all concerns around the resources. Furthermore, the study findings also indicate that conducting a user study to gather API-users perceptions can help in designing useful resources for APIs.

Designing a usable API is complex and time consuming [30], there are different API evaluation approaches that can be used to evaluate usability of the API. In general most of the approaches to evaluate an API are applied once an API is developed. This paper presented a development process designed by combining a development process with an evaluation technique. To help the API-developer to address different difficulties faced while designing a usable API such as requirements uncertainty, time pressure to deliver an API and different usability issues [2]. In order to address these difficulties we created a development process (XP + CDs) by combining a subset of XP practices (i.e. incremental designing, release and iteration planning, user stories, short iteration, refactoring, unit tests, and acceptance testing) with the Cognitive Dimensions (CDs) framework (an evaluation process) to design a usable API. Furthermore we developed the GIST-API using the development process and conducted a user study to evaluate the API usability in order to determine the effectiveness of the development process in designing a usable API. The findings from the user study suggests that the combined process helped in making the API design usable but not able to address all the concerns for the resources.

Acknowledgments

This research was funded by a NSERC SurfNet Project Grant and a Mitacs Accelerate Postdoctoral Fellowship.

References

[1] D. Dig and R. Johnson, "How do APIs evolve? A story of refactoring," *J. Softw. Maint. Evol.*, vol. 18, no. 2, pp. 83–107, 2006.

[2] M. Robillard, "What makes APIs hard to learn? answers from developers," *IEEE Softw.*, vol. 26, no. 6, pp. 27–34, 2009.

[3] S. McLellan, A. Roesler, J. Tempest, and C. Spinuzzi, "Building more usable APIs," *IEEE Softw.*, vol. 15, no. 3, pp. 78–86, 1998.

[4] J. Stylos, S. Clarke, and B. Myers, "Comparing API design choices with usability studies: A case study and future directions," in *Proceedings of the 18th Workshop of the Psychology of Programming Interest Group (PPIG)*, 2006, pp. 131 – 139.

[5] U. Farooq, L. Welicki, and D. Zirkler, "API Usability peer reviews: a method for evaluating usability of application programming interfaces," in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI)*, 2010, pp. 2327–2336.

[6] C. Bore and S. Bore, "Profiling software API usability for consumer electronics," in *Proceedings of International Conference on Consumer Electronics, (ICCE) Digest of Technical Papers*, 2005, vol. 1, pp. 155–156.

[7] T. R. G. Green and M. Petre, "Usability Analysis of Visual Programming Environments: A 'Cognitive Dimensions' Framework," *J. Vis. Lang. Comput.*, vol. 7, no. 2, pp. 131–174, 1996.

[8] S. Clarke, "Describing and measuring API usability with the cognitive dimensions," in *Proceedings of the 10th Anniversary Workshop Cognitive Dimensions of Notations*, 2005.

[9] R. B. Watson, "Improving software API usability through text analysis: A case study," in *Proceedings of IEEE International Professional Communication Conference, (IPCC)*, 2009, pp. 1–7.

[10] M. Huo, J. Verner, L. Zhu, and M. a Babar, "Software quality and agile methods," in *Proceedings of the Computer Software and Applications Conference, COMPSAC*, 2004, pp. 520–525.

[11] K. Beck, *Extreme programming eXplained : embrace change*. Reading, MA: Addison-Wesley, 2000.

[12] K. Beck and M. Fowler, *Planning extreme programming*, Edition 1. Boston: Addison-Wesley Professional;, 2001.

[13] J. Highsmith and A. Cockburn, "Agile software development: the business of innovation," *Computer (Long. Beach. Calif).*, vol. 34, no. 9, pp. 120–122, 2001.

[14] M. F. Zibran, "What Makes APIs Difficult to Use?," *Int. J. Comput. Sci. Netw. Secur.*, vol. 8, no. 4, pp. 255–261, 2008.

[15] E. Folmer and J. Bosch, "Architecting for usability: A survey," *J. Syst. Softw.*, vol. 70, no. 1–2, pp. 61–78, 2004.

[16] J. Blanchette, "The Little Manual of API Design," *Trolltech, Nokia*, 2008.

[17] M. Zibran, F. Eishita, and C. Roy, "Useful, but usable? Factors affecting the usability of APIs," in *Proceedings - Working Conference on Reverse Engineering, WCRE*, 2011, pp. 151–155.

[18] D. Rowley and D. Rhoades, "The cognitive jogthrough: a fast-paced user interface evaluation procedure," in *Proceedings of the SIGCHI conference on Human factors in computing systems (CHI)*, 1992, pp. 389–395.

[19] T. Memmel, F. Gundelsweiler, and H. Reiterer, "CRUISER: A Cross-Discipline User Interface and Software Engineering Lifecycle," *Human-Computer Interact. Interact. Des. Usability*, pp. 174–183, 2007.

[20] M. Singh, "U-SCRUM: An agile methodology for promoting usability," in *Proceedings of IEEE Agile Conference*, 2008, pp. 555–560.

[21] W. Ahmad, S. Butt, and L. Rahim, "Usability Evaluation of the Agile Software Process," *Adv. Vis. Informatics Springer*, pp. 640–651, 2013.

[22] P. McInerney and F. Maurer, "UCD in agile projects," *Interactions*, vol. 12, no. 6, pp. 19–23, 2005.

[23] M. Robillard and R. Deline, "A field study of API learning obstacles," *Empir. Softw. Eng. Springer*, vol. 16, no. 6, pp. 703–732, 2011.

[24] M. Henning, "API Design Matters," *ACM QUEUE*, no. June, pp. 25–36, 2007.

[25] K. Bierhoff, N. Beckman, and J. Aldrich, "Practical API protocol checking with access permissions," *Proceeding Eur. Conf. Object-Oriented Program.*, pp. 1–25, 2009.

[26] R. Watson, "Applying the Cognitive Dimensions of API Usability to Improve API Documentation Planning," in *Proceedings of the 32nd ACM International Conference on The Design of Communication CD-ROM*, 2014, pp. 2–3.

[27] S. Clarke, "Measuring API Usability," *Dr. Dobb's J. Wind. Suppl.*, vol. 10, no. 1, pp. S6–S9, 2004.

[28] S. Clarke and C. Becker, "Using the Cognitive Dimensions Framework to evaluate the usability of a class library," in *Proceedings of the First Joint Conference of EASE and PPIG*, 2003, no. April, pp. 359–366.

[29] J. Brosz, M. a. Nacenta, R. Pusch, S. Carpendale, and C. Hurter, "Transmogrification: casual manipulation of visualizations," *Proc. 26th Annu. ACM Symp. User interface Softw. Technol. - UIST '13*, pp. 97–106, 2013.

[30] C. De Souza and D. L. M. Bentolila, "Automatic evaluation of API usability using complexity metrics and visualizations," in *Proceedings of International Conference on Software Engineering, (ICSE)*, 2009, pp. 299–302.