Myllärniemi *et al. Journal of Software Engineering Research and Development* (2018) 6:6
https://doi.org/10.1186/s40411-018-0050-8

Journal of Software Engineering Research and Development

RESEARCH Open Access

Development as a journey: factors supporting the adoption and use of software frameworks

Varvana Myllärniemi[1], Sari Kujala[1], Mikko Raatikainen[2,1*] and Piia Sevón[3]

*Correspondence:
mikko.raatikainen@helsinki.fi
[2]University of Helsinki, Helsinki, Finland
[1]Aalto University, Espoo, Finland
Full list of author information is available at the end of the article

Abstract

From the point of view of the software framework owner, attracting new and supporting existing application developers is crucial for the long-term success of the framework. This mixed-methods study explores the factors that support developers in adopting and continuously using a framework. Data was collected from two sources: interviews with experienced practitioners and a longitudinal survey of novice developers. According to the results, developers use API (application programming interface) capabilities and peer experiences to justify the selection of the framework. To commit developers and to ensure continuous use, enjoyment of using the framework and its perceived usability are important factors. Instead of focusing solely on the API, the framework owner should consider all platform boundary resources: API, development tools and information. In addition, the boundary resources should support developers' needs throughout the developer journey, from early adoption to continuous use.

Keywords: Software framework, Application development, Software ecosystem, Usability, Developer experience, Developer journey

1 Introduction

Software frameworks are nowadays extensively used to develop different kinds of software applications efficiently. For example, using a framework, such as Spring, Django, Node.js or Angular.js, is the *de facto* standard approach for developing web software. Similarly, mobile applications are developed using a mobile application framework; examples include PhoneGap and Ionic. Selecting a framework is a critical design decision in any application development. This is because the selected framework will dictate the application architecture (Johnson 1997) — for example, Django will dictate that the application is organized to follow a specific Model-View-Controller design pattern. Hence, once the development has started, changing the selected framework may require the redesign and reprogramming of the application.

Software frameworks can often be paralleled at least from development perspective with software ecosystems. Software ecosystems have become an increasingly popular means to collaboratively develop software (Hanssen and Dybå 2012). In both software ecosystems (Bosch and Bosch-Sijtsema 2010) and frameworks, there exists a shared

common technology, i.e., platform, that is developed by the platform owner, and developers build applications on top of the platform to satisfy the end-users' needs.

In software ecosystems, facilitating application development is important for the success of the ecosystem (Bosch 2009; Ghazawneh 2012). This is likewise in software frameworks. Frameworks, however, are not typically usable by or visible to end-users, but exist mainly to serve application developers and to facilitate application development. Therefore, we considered that it is worthwhile studying in the context of frameworks how the framework owner can attract new developers and support them in their activities throughout the developer journey. In this study, a developer journey refers to the lifecycle of the developer interacting with the framework, corresponding to the concept of a customer journey (Zomerdijk and Voss 2010). Besides just attracting new developers, it is important to serve existing developers in their journey. Although initial acceptance is important for any software system, its long-term success depends on continued use and user adoption (Bhattacherjee 2001).

Specifically, we focus on platform boundary resource. In software ecosystems, platform boundary resources have been identified as the key means to facilitate application development (Dal Bianco et al. 2014). Platform boundary resources expose and extend the platform to application developers and are the externally visible assets that application developers use from the platform (Ghazawneh and Henfridsson 2010; Ghazawneh 2012). Examples of boundary resources include APIs (Application Programming Interfaces) but also all other resources or assets provided for developers such as development tools, documentation, and forums (Dal Bianco et al. 2014). Similar boundary resources also exist in software frameworks. Therefore, it is worthwhile studying how platform boundary resources can support the developers also in the context of frameworks.

1.1 Research problem and contributions

Our goal is to investigate the following overall research question: *What factors support application developers in adopting and continuously using a software framework?* The detailed research questions are set as follows:

RQ1 How do platform boundary resources support or hinder the adoption and continuous use of a framework?

RQ2 What factors support the customer loyalty of application developers after initial framework use?

The results of the study help framework owners to attract new developers and support tehm in their activities. To answer the research questions, we conducted an explorative, mixed-method study. First, we interviewed experienced practitioners to explore how platform boundary resources affect the usage of the framework. The interviews focused on one specific framework, Qt, and all practitioners were from companies that use the Qt framework. Second, we conducted a longitudinal student survey to explore the selection of the framework and first steps of its initial use both qualitatively and quantitatively. The survey was conducted during a university course where students freely selected and used any framework, including Qt, to build an application. This enabled us to follow adoption and initial use while they were taking place, compared to practitioners who adopted the framework a long time ago. Hence, our study included both experienced and novice Qt developers, and combined both qualitative and quantitative data collection and analysis.

Moreover, we studied the phenomenon in different phases: before the adoption of the framework, after initial use, and in continuous use.

This study provides the following contributions. We did not solely focus on the API as is often done in the previous work, but took a broader view of all platform boundary resources and through the different phases of the developer journey. Thus, we were able to identify how different platform boundary resources support framework selection, adoption, initial use, and continuous use. We also explored the developers' overall evaluation of the framework measured in terms of enjoyment and usability. Indeed, we found that enjoyment and usability predict developers' customer loyalty to the framework.

2 Background

2.1 Software frameworks

Software frameworks, also termed application development frameworks, specify reusable architecture design and partial implementation of an application (Fayad and Schmidt 1997; Johnson 1997). Software frameworks also have a few drawbacks. Because frameworks are complex and involve specific designs, they are difficult to learn and may require extensive hands-on training (Johnson 1997; Fayad and Schmidt 1997). The suitability of a framework for a particular application may not be apparent until some investment in learning has already taken place (Fayad and Schmidt 1997). Debugging applications can be tricky due to the inverted flow of control between the framework and application-specific method callbacks (Fayad and Schmidt 1997).

The current research on frameworks tends to focus on their technical capabilities, for example comparisons between and selection guidelines for mobile application frameworks (Serrano et al. 2013; Ribeiro and da Silva 2012) and web frameworks (Shan and Hua 2006). There are also descriptions of specific frameworks, such as Ruby on Rails (Bächle and Kirchberg 2007) and Java (Johnson 2005). In contrast, there is little research on how frameworks are selected and used by developers in practice.

However, when focusing only on the API provided by the framework, there exists a wealth of research from different viewpoints. In particular, API usability (Rama and Kak 2015; Burns et al. 2012; Piccioni et al. 2013; Robillard and Deline 2011) and API learnability (Robillard and Deline 2011; Stylos and Myers 2008) have been commonly studied. This is because poor API usability can lead to programmer frustration, reduced productivity and potential bugs (Rama and Kak 2015). API usability is affected by, for example, method naming and grouping, parameter naming and grouping, exception declarations, and method-level documentation (Rama and Kak 2015). Similarly, an API should clearly map to the problem domain concepts (Ratiu and Jurjens 2008). One take-away is that documentation is crucial in supporting developers in learning and using an API (Burns et al. 2012; Robillard and Chhetri 2015; Rama and Kak 2015; Piccioni et al. 2013).

As another popular topic, API evolution has been studied extensively (Robbes and Lungu 2011; Espinha et al. 2015), also from the viewpoint of frameworks (Schäfer et al. 2008; Robbes and Lungu 2011). When an API changes, client applications may have to propagate the changes on their side (Robbes and Lungu 2011). This is made worse by the sporadic communication between the API owner and application developers (Espinha et al. 2015; Robbes and Lungu 2011). An API-breaking change may take months to be noticed (Robbes and Lungu 2011). Therefore, the API owners should put more effort into

early versions of APIs, and application developers should separate parts that have the potential to change through good architectural design (Espinha et al. 2015).

2.2 Platforms and platform boundary resources

For many companies, customer needs cannot be satisfied through internal development alone (Bosch 2009). By opening up their products as software ecosystems, companies give up part of their control and design capabilities (Ghazawneh and Henfridsson 2012) to gain greater speed and wider coverage. Consequently, software ecosystems are also gaining popularity as a research topic. Over 100 studies on existing ecosystems have been reported in the literature (Manikas 2016).

A software ecosystem is defined as "a set of businesses functioning as a unit and interacting with a shared market for software and services, together with the relationships among them, where relationships are frequently underpinned by a common technological platform or market" (Jansen et al. 2009).

Hence, a software ecosystem exists around a central technology that is often referred to as a software platform (Manikas and Hansen 2013; Bosch 2009; Bosch and Bosch-Sijtsema 2010). Within software ecosystems, the platform is a key technological resource (Manikas and Hansen 2013; Taylor 2013). The platform may be, for example, an end-user application, an operating system, a standard system architecture, or a communications protocol (Bosch 2009; Taylor 2013). The platform can be used directly by end-users (for example, the Google Drive ecosystem), but can also be used primarily by other software developers (for example, the Eclipse plugin ecosystem or Apache plugin ecosystem). Sometimes the platform resembles an infrastructure that is utilized during software development in different domains (Manikas 2016). An important property of the software platform is that it is extensible beyond the borders of the organization controlling it (Hanssen 2012), and that it is important for interaction within the ecosystem (Manikas 2016).

The platform owner needs to provide resources, *platform boundary resources* (Ghazawneh and Henfridsson 2010; Ghazawneh 2012), so that external developers can build applications on top of the software platform. The platform boundary resources are a way for the platform owner to control and influence what happens in the software ecosystem, while benefiting from the generativity of external application development (Ghazawneh and Henfridsson 2010). Furthermore, platform boundary resources minimize the coordination effort between application developers and the platform owner. Finally, platform boundary resources enable coordination between application developers. This is especially crucial when companies that build similar applications want to collaborate with each other directly (van Angeren et al. 2016).

There are three different kinds of platform boundary resources (Fig. 1) (Dal Bianco et al. 2014). *Application boundary resources* expose the services of the platform to applications and are used by applications at runtime. Examples include APIs, libraries, and binaries. *Development boundary resources* are the tools provided to developers to support them in application development. Examples include integrated development environments (IDE), software development kits (SDKs), debuggers, compilers, editors, and application stores. *Social boundary resources* transfer the information needed to develop the applications and explain the platform to developers. Examples include documentation, training material, events, development guidelines, intellectual property rights, agreements between the platform owner and application developers, social media and online forums. Social

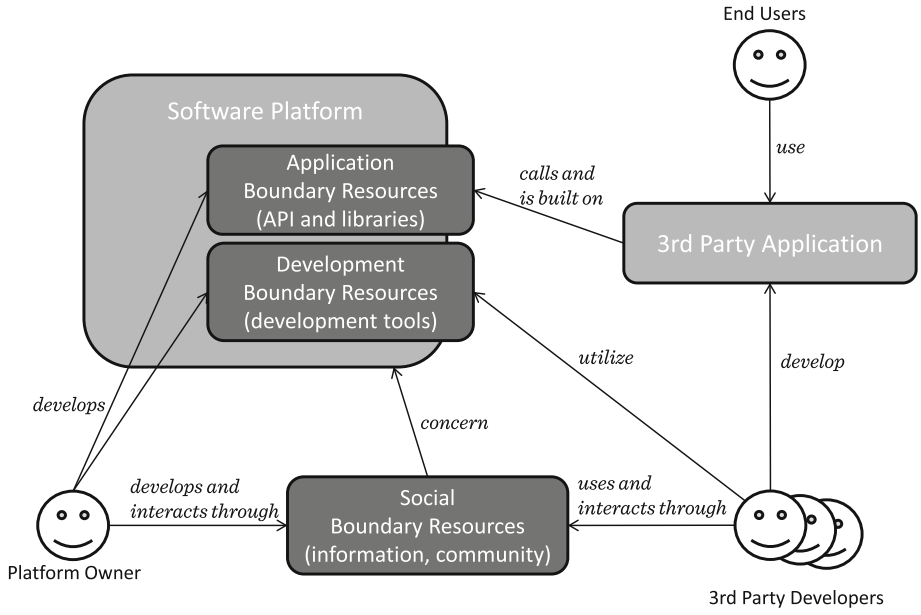

Fig. 1 Different kinds of platform boundary resources enable the developers to build applications on top of the software platform (Dal Bianco et al. 2014)

boundary resources also enable interaction within the development community; software development is ultimately a social activity (Messerschmitt and Szyperski 2003).

3 Methods

We carried out a mixed-methods study on software frameworks and collected both qualitative and quantitative data (Fig. 2). First, we interviewed nine experienced practitioners from two companies that develop applications using the Qt framework. Second, we conducted a longitudinal survey among students who represented less experienced developers and were selecting and starting to use Qt or other frameworks to develop an application.

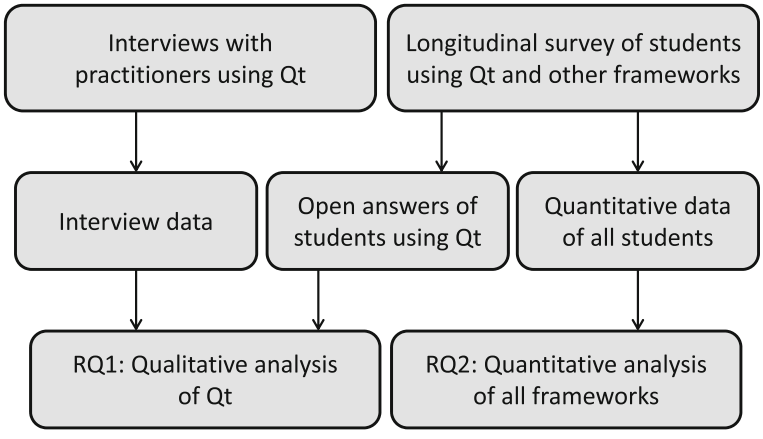

Fig. 2 Overview of the research method

3.1 The studied frameworks

The main framework in the focus of this research was Qt[1]. In addition, the student survey included data from other frameworks that the students used in their course assignment (Fig. 2): this enabled comparison between Qt and other frameworks.

Qt is a software framework that is targeted especially for cross-platform mobile application development, graphical UI (user interface) development, and embedded application development. To support cross-platform development, Qt applications can be compiled directly to multiple operating systems working in a wide variety of devices such as PCs, tablets, mobile phones, and embedded systems. Qt is offered under a dual model of open source and proprietary licenses. Qt was originally released around 20 years ago and has reached version 5.7.0. The development of Qt is now managed by Qt Company. Originally, Qt was developed in Norway and later acquired by Nokia. Several notable applications around the world have been developed using Qt: examples include the KDE desktop environment and Google Earth. Qt supports applications written in C++ and QML. QML (Qt Markup Language) is a proprietary JavaScript-like scripting language, using which UIs can be programmed declaratively.

We selected Qt as the main study subject because of its accessibility. Qt is also relatively popular especially in commercial embedded products and it has long history that has made it mature and quite large. For example, there exist a lot of different types of boundary resources and each type of boundary resource are significant in size.

3.2 Data collection: interviews

We conducted nine semi-structured interviews with practitioners using Qt (left side of Fig. 2). Hence, this part of the study focused on the views of experienced and already loyal developers who use Qt regularly.

Selecting participants The target population was experienced practitioners working in companies that use Qt in their application development.

The practitioners were selected using convenience sampling. We asked one Qt manager to name external application developers and projects. Based on his recommendation, we recruited participants from two Finnish companies. The contact persons from those companies were approached via e-mail to identify practitioners for interview.

The selected practitioners' professional experience ranged from 10 to 25 years. They represented different kinds of stakeholders: three developers and architects, three consultants, and three managers (Table 1). Their experience in using Qt also varied. Five of them had used Qt for several yeas, up to 11 years. Four interviewees had used Qt less than one year. Six of the practitioners were from Company A, whereas three of the practitioners were from Company B. The practitioners were using Qt to build either mobile applications across several platforms (Android, iOS and Windows Phone) or embedded software that was delivered together with the hardware.

Table 1 Number and roles of interviewed practitioners

Participants	Practitioner role	Practitioner tasks with Qt
3	Developers and architects	Develop and design applications
3	Qt consultants	Develop applications, provide customer guidance and analysis
3	Managers	Create demos, take part in the framework selection process

Data collection The interviews were conducted in a semi-structured style. A set of themes with predefined questions were used, but additional questions were asked for each theme. The themes were as follows: *background information*, *selection of Qt*, *getting started with Qt*, *daily use of Qt*, *use of Qt platform boundary resources*, *willingness to recommend Qt*, and *future expectations and needs*.

The average length of the interviews was 1 hour and 16 minutes. The interview protocol followed the recommendations in (Seaman 1999). The interviews were voice-recorded and captured into initial notes. Soon after each interview session, extensive and detailed notes were written by listening to the recording.

The first two interviews were conducted as pilot interviews. The pilot interviewees represented different kinds of stakeholders, which also gave an understanding on how the themes fit to different types of interviewees.

3.3 Data collection: student survey

We conducted a longitudinal student survey (right side of Fig. 2) to investigate how novice developers select, adopt and initially use frameworks.

Participants and procedure The target population was less experienced developers. To select participants to match this population, we conducted the study among Master's level university students who were attending the User Interface Construction course in the department of Computer Science.

During the course, students developed an application that consisted of a UI and some mock-up functionality to enable interaction. Three versions of the application for various platforms were developed during the course. Although Qt was advertised by the course personnel, the students were free to use any framework and change it during the course. The application was created in teams of three students, and the project lasted approximately three months.

The longitudinal survey was timed in three phases to capture students' expectations, initial experiences and final experiences (Table 2). Students were invited to participate in the study during a lecture and through the course web page. Participation was not mandatory, but students received two extra points for their course for responding to all three questionnaires. Out of 86 students who participated the course, 74 answered questionnaire I (response rate 86.0%), 64 students answered questionnaire II, and 51 students answered questionnaire III. The drop in participants is typical, and unavoidable in longitudinal studies (Ployhart and Ward 2011).

Out of those 51 students who responded to all three questionnaires, 57% were male. Their ages ranged from 19 to 50, with the majority (67%) belonging to the age group

Table 2 The student survey consisted of three questionnaires

Questionnaire	Measurement point	Information gathered
I Expectations before usage	At the beginning of the course	Background information, expectations, expected usability, expected enjoyment
II Initial experiences	3 weeks from project start	Usage, enjoyment, usability, support from platform boundary resources
III Final experiences	Project end after 3 months	Usage, customer loyalty, enjoyment, usability, support from platform boundary resources

20-25 years. The students had little previous experience with Qt: 6.8% of the students had tried out Qt and 6.8% had used Qt repeatedly. The students' overall software development experience ranged from a few programming courses to 15 years of professional experience. About 30% of students mentioned having experience from hobby projects and 33% of the respondents already worked in a company as a software developer.

Survey design The longitudinal survey used similar design and quantitative measures as a previous longitudial study focusing on the role of expectations and experiences in service evaluation (Kujala et al. 2017). The survey included three questionnaires at three time points: before use, after three weeks of use, and after three months of use (Table 2). Enjoyment and subjective usability were measured in all three questionnaires to find out how they evolve over time: first the expectations, then the experiences. Questionnaire I asked about students' background, experience of using frameworks and expectations, while questionnaires II and III focused on students' experiences (Table 3). Enjoyment was used as a measure of how intrinsically motivating the framework was (Davis et al. 1992), and the statements that measured enjoyment were adapted from (Davis et al. 1992)

Table 3 The questions used in both questionnaires II and III

Theme and questions	Response type
Usage:	
What framework have you primarily used for implementing your course assignment?	Open
Estimate how many working hours you have spent so far on finding information on or installing the selected framework.	Open
Estimate how many working hours you have spent so far on implementing the assignment with the selected framework.	Open
Customer loyalty:	
How likely would you recommend the framework to a friend who is interested in it?	Likert 0-10
Enjoyment:	
I felt good about the framework.	Likert 1-7
I enjoyed using the framework.	Likert 1-7
I think using the framework was rewarding.	Likert 1-7
If you wish, you can explain your experiences here.	Open
Usability:	
The framework met my requirements.	Likert 1-7
Using the framework was a frustrating experience.	Likert 1-7
The framework was easy to use.	Likert 1-7
I needed to spend too much time correcting things when using the framework.	Likert 1-7
If you wish, you can explain your experiences here.	Open
Support from platform boundary resources:	
How well did the APIs support the implementation of your course assignment?	Likert 1-7
Describe any problems with the APIs.	Open
Describe any strengths of the APIs.	Open
How well did the development tools support the implementation of your course assignment?	Likert 1-7
Describe any problems with the development tools.	Open
Describe any strengths of the development tools.	Open
What information sources related to the framework did you use?	Open
How well did the information sources support using the framework?	Likert 1-7
Describe any problems with the information sources.	Open
Describe any strengths of the information sources.	Open

and (Mitchell et al. 1997). Subjective usability of the framework was measured with the Usability Metric for User Experience (UMUX) (Finstad 2010). UMUX measures usability through three ISO 9241-11 dimensions: effectiveness, efficiency, and satisfaction. To study customer loyalty, we measured users' overall evaluation of the framework using likelihood-to-recommend measure that can be used to calculate the Net Promoter Score (NPS), a strong indicator of customer loyalty and growth (Reichheld 2003). To study how the platform boundary resources support development, the students were asked to rate the different boundary resources and to identify their strengths and weaknesses (Table 3).

The questionnaires were piloted (Kitchenham and Pfleeger 2002) and corrected based on the feedback. In the pilot, two Master's students and one student from the course answered the questionnaires and thereafter described how they understood each question.

3.4 Data analysis

The data analysis was split into two parts (Fig. 2). The qualitative analysis focused on the Qt framework and on answering **RQ1**. The quantitative analysis covered all of the selected frameworks to answer **RQ2**.

Qualitative analysis The analysis for **RQ1** utilized two data sources: practitioner interviews and open answers from the student survey (Fig. 2). The analysis was conducted as qualitative content analysis using both a priori and emerging codes (Lazar et al. 2010). The coding of the content utilized open coding, similar to Grounded Theory (Strauss and Corbin 1998).

The process of analyzing the practitioner interviews (Section 3.2) started immediately after the first interview. The data to be analyzed consisted of the detailed notes written when listening to interview recordings. *A priori* codes, based on the interview themes and on existing knowledge on Qt, were used to mark and organize the data. Examples of such codes were *social boundary resource* and *selecting Qt*. The data was organized into an Excel spreadsheet, where each row contained the data corresponding to one code and each column corresponded to one interviewee. In addition, emerging codes were used to expand and refine the analysis; examples of emerging concepts included *open source licensing* and *peer support*. Initial analysis was carried out immediately after each interview. The results of the analysis and new codes were then taken into account in the following interviews and previous interviews were refined if needed. The cross-analysis of the interviews was carried out after all the interviews had been conducted; it combined data related to one code as well as compared data from different codes.

The process of analyzing the student survey results was based on the responses to the open questions in the student survey (Section 3.3). This analysis focused on the final experiences (questionnaire III) and on those students who selected the Qt framework (15 students out of 51). The open responses from these 15 students yielded rich qualitative content to be analyzed (four pages of plain text in total). The content was coded in similar way as the interviews: using predefined codes for each platform boundary resource type, and using emerging codes (e.g., *deployment*, *taking into use*).

Finally, a cross-analysis of interviews and survey was performed to answer **RQ1**. The data and analysis results were combined and categorized per platform boundary

resources. We compared practitioner and student experiences for each platform boundary resource type, to identify any similarities and differences. Emerging concepts were used to construct main findings for each platform boundary resource type. An example of such an emerging concept was *fragmentation of social boundary resources*. In addition, we identified whether such concepts supported or hindered development, and to which development phase they were related.

Quantitative analysis The analysis for **RQ2** was based on the quantitative data from the student survey (Fig. 2). The statistical analysis began with establishing the summary measures of the scales and assessing their reliability. Data were only included from those 51 students that answered all three questionnaires, so that the trends of the variables could be reliably analyzed. Moreover, 10 students were excluded from the statistical analysis for partly missing data. Cronbach's alpha coefficients of enjoyment were .885, .867 and .954 from the first to the third questionnaire. Cronbach's alpha coefficients of usability were .746, .747 and .870 accordingly, demonstrating a high degree of internal reliability of the scales. To study the relation between students' loyalty to the framework and other study variables (Table 3), bivariate correlations were used to compute the correlation coefficients.

4 Results for RQ1

The following presents how the platform boundary resources in Qt support or hinder the adoption and use of the framework (**RQ1**). The results focus on Qt and are based on practitioner interviews and open answers from the student survey (Fig. 2).

4.1 Application boundary resources

The application boundary resources in Qt (Table 4) consist of the Qt APIs, modules, and programming interfaces. Table 5 identifies how these boundary resources support or hinder the adoption and use of Qt.

API capabilities are the key justification when developers select the framework. For example, students mentioned the cross-platform support in the API as the reason why they chose Qt in the first place. Similarly, practitioners recommended selecting Qt based on the API capabilities: an important justification was the API support for cross-platform development and for embedded software development. Conversely, if there was a need to develop a mobile application on top of one platform only, practitioners would not recommend selecting Qt.

Table 4 Main platform boundary resources in Qt

Boundary resource type	Main boundary resources in Qt
Application boundary resources	Qt modules in C++, e.g., Qt Core and Qt UI
	QML (JavaScript-style declarative language)
Development boundary resources	Qt Creator, Qt Designer, various compilers, etc.
Social boundary resources	Qt website, e.g., Qt Documentation, Qt Blog
	Qt Project site, e.g., Downloads, Forums
	Documentation embedded in Qt Creator tool
	Qt Resource Center, Qt Account, Qt social media
	Other: searching Google, StackOverflow, Youtube

Table 5 How the different platform boundary resources supported or hindered the use and adoption of Qt

Application boundary resources:	
API capabilities	*Supports adoption*: used to justify selecting a framework
Unfamiliarity of the programming languages	*Hinders adoption*: adds to the learning curve
Specialized APIs for different developers and purposes	*Supports use*: improves productivity
	Hinders use: if lacks necessary features
Framework source code available	*Supports use*: improves framework visibility
	Supports adoption: ensures framework continuity
Development boundary resources:	
Difficulty in installing tools or initializing projects	*Hinders adoption*: getting started takes time and effort
Unnecessary libraries in installed tools	*Hinders adoption*: download size, unexpected issues
Complicated deployment	*Hinders (initial) use*: major showstopper
Editor capabilities and embedded documentation	*Supports use*: improves efficiency
Social boundary resources:	
Personal recommendation and peer experiences	*Supports adoption*: are valued when selecting the framework
Active, long-lived community	*Supports adoption*: continuity is important when selecting the framework
Insufficient or irrelevant information on getting started	*Hinders adoption*: not adequate guidance
Information not targeted and organized for developers' needs	*Hinders use*: lowers satisfaction, searching information takes too much time
Code examples, videos	*Supports use*: give easy-to-apply, hands-on guidance
Peer help, online or face-to-face	*Supports use*: sharing knowledge about similar problems

When developers adopt a framework, the need to learn a new programming language adds to the initial learning curve. Most Qt adopters have to learn a new language, since Qt uses a proprietary language, QML. According to practitioners, if a developer needs to learn a new language, this is a possible negative factor against selecting the framework. This learning curve from a new programming language was also experienced in the student survey. In general, most students were willing to invest in learning: 56.8% wanted to learn a new framework during the course. However, some students felt it took too much time to learn QML, despite the fact that QML resembles JavaScript and thus was designed to be relatively easy to adopt for UI developers. According to one student, "a fundamental issue, like how to add functionality to QML elements, required several hours of brainwork [to learn]". Similarly, some students were not familiar with C++ and reported having difficulties every time they had to write their applications in C++.

Specializing APIs to serve different kinds of application developers and purposes improves productivity, provided that the specialized APIs do not lack the necessary features. Qt APIs are specialized to serve two purposes: C++ APIs are meant for developers of embedded software and those who are familiar with C++, whereas QML APIs are meant for UI developers. According to practitioners, QML APIs make it easy to create UI-oriented applications by declaring UI elements. In addition, students who had to create a UI-oriented application, confirmed that QML APIs improved productivity. According to one student, "the main strength of QML is its ease of use: some things that are difficult

at first sight can be easily implemented in a small amount of code". However, challenges arise if developers cannot solely rely on their specialized APIs. Students reported that QML was still lacking important features needed for UI development, and they had to write their applications partly on top of C++ APIs. One student reported, "It was difficult to understand the relation between UI elements that needed to be implemented in C++ and QML, respectively".

Having the framework implementation available as open source code improves the visibility and continuity of the framework. According to practitioners, the source code was used for debugging, diagnosing problems, and evaluating the usefulness of Qt for a certain use. Furthermore, when selecting the framework, practitioners need to be assured of the continuity of the framework. When the framework is published as open source code with a suitable licensing model, the framework and its future are not tied only to one company.

4.2 Development boundary resources

The Qt development boundary resources (Table 4) consist of development tools that are provided for Qt application developers. Table 5 identifies how these boundary resources support or hinder the adoption and use of Qt.

If the development tools are not easy to install, or it is difficult to initialize the first project, adoption is hindered. Some students had difficulties in installing the necessary development tools, which caused a great deal of frustration. For example, one student reported that the default compiler in the Qt IDE was not installed properly. Students reported having difficulties in initializing the project, setting up the integration with the configuration management tool, and importing resources into a new project. One student reported, "It was, for instance, difficult to know which project type to choose for a new desktop project". According to the quantitative data, if students spent a lot of time installing framework tools, their satisfaction with the framework was lower (Table 6).

If the installed development tools include unnecessary libraries, adoption is hindered. Almost all practitioners counted the size of the downloaded libraries as a downside: in order to write even a few-row application, megabytes of libraries had to be downloaded along with the tools. Students also noted that the development tools referred to libraries that were not needed by the application. For example, one student said she encountered OpenGL driver issues when starting a project that contained a simple 2D user interface.

Table 6 Correlation between different factors and students' customer loyalty to the framework

Factor	Correlation with loyalty
Enjoyment	.91[a]
Usability	.84[a]
Quality of social boundary resources	.63[a]
Quality of development boundary resources	.57[b]
Quality of application boundary resources	.45[a]
Working hours spent on finding information on and installing the framework	-.33[b]
Working hours spent in total	-.23[ns]

[a]: Significant at the 0.05 level
[b]: Significant at the 0.01 level
[ns]: Not significant

If the development tools do not support the deployment of applications, initial use is hindered. Students had difficulties in understanding how to deploy the compiled programs correctly and to manage the dependencies. Deployment seemed to be quite critical for early developer experience: those students that reported deployment problems were dissatisfied with the whole framework, and regretted choosing Qt over other frameworks. Interestingly, none of the practitioners mentioned deployment as a challenge. Hence, easy deployment seems to be especially important for initial use.

Good editor capabilities and documentation embedded in the editor improve development efficiency. Several students mentioned Qt code editor as being easy to use and handy. In addition, students and practitioners said that the embedded documentation in the editor was very helpful.

4.3 Social boundary resources

The social boundary resources of Qt (Table 4) include all sources of information and communication means used by Qt developers. Table 5 identifies how these boundary resources support or hinder the adoption and use of Qt.

When selecting the framework, personal recommendation and peer experiences are valued highly. When selecting the framework, practitioners do not entirely trust the marketing material on the Qt website, but they would like to see real references from somebody who has actually used the framework. For example, posts on the official Qt blog were not considered as neutral as blog comments, which reflected real-life experiences. Practitioners considered that the best source of recommendation and experience was an expert who knew about the framework.

When selecting the framework, active and long-lived community is valued. When selecting the framework, practitioners valued framework continuity. Wide framework usage and an active, long-lived existing community create trust in the framework's future.

If there is insufficient information on getting started, adoption is hindered. Students reported there was a gap between what was described in the documentation and what was actually required when starting application development. One student noted, "It was difficult to get started straight away by trying to implement any more complicated system than in any of the tutorials". Students hoped for a step-by-step getting-started tutorial as well as a description of the "big picture" of the framework. One student compared it with another framework and its package of getting-started videos: after watching a few videos, a novice can start application development right away. In contrast, the more experienced practitioners complimented Qt documentation as being highly suitable for their needs.

If the information is not targeted at and organized to meet developers' needs, the use of the framework becomes less satisfactory. Students felt that the available documentation and information was more targeted at experienced developers and hence did not serve their needs well. Even though practitioners valued the Qt documentation, they felt that the Qt website was mostly not targeted at them, but contained merely marketing and managerial material. Moreover, both students and practitioners said it was sometimes difficult to locate correct information: one reason was because the documentation was organized along the lines of framework versions. Furthermore, too many or fragmented information channels, such as blogs, tutorial, manuals and websites, make it difficult to search for and use information. Both practitioners and students said there were many different information channels and it took too much time to search for information from

such channels. The importance of meeting developers' needs with information was also highlighted in the student survey. According to the quantitative data, if students spent a lot of time searching for information, their satisfaction with the framework was lower (Table 6).

Realistic enough code examples and how-to videos give practical, hands-on guidance in using the framework. Practitioners and students felt that code examples were useful: code examples show how to solve a particular problem or what can be done with a particular feature. One practitioner pointed out that code examples should also demonstrate how to create attractive applications with good user experiences, and hence the examples should be created with UI designers. Furthermore, students valued how-to videos highly: videos were considered to be particularly helpful in the early stages of getting started with development. In contrast, practitioners indicated that Qt's social media and other such channels are not really useful: they add to the already huge number of social media channels that need to be followed. Consequently, it seems that the information sources are primarily used on a need-to-know basis, for trying to solve a particular problem.

Peer help, either online or face-to-face, is valuable for solving problems in using the framework. Both students and practitioners used peer help, such as StackOverflow[2] discussions, frequently. Such peer help often focused on particular problems that students had. Practitioners said the best way to solve a problem is to personally ask an expert. For three practitioners, this expert was a dedicated contact person within the Qt organization.

5 Results for RQ2

The following presents the factors that support the customer loyalty of developers after initial framework use (**RQ2**). For this purpose, we analyzed the quantitative data of the longitudinal student survey of Qt and other frameworks (Fig. 2). In the analysis, loyalty was measured as willingness to recommend, whereas usage hours, enjoyment, usability and different platform boundary resources constituted the supporting factors (Table 3).

All survey participants, regardless of the framework they used, were included into the analysis. Out of 51 analyzed students, 15 selected Qt and used it to the end of the assignment. Other choices included using various frameworks, such as Ionic, Django and node.js, with a general-purpose editor, such as Eclipse. Frameworks with their own editing environments, such as Android Studio and Visual Studio, were also used.

Enjoyment experienced in using the framework particularly supported customer loyalty. When students enjoyed using the framework, they were more willing to recommend the framework (Table 6). As students were free to select the framework, it obviously was relevant for them that using the framework was emotionally rewarding and meaningful. Only a few of the respondents described their enjoyment in the open answers, but they particularly enjoyed it and felt powerful when they could accomplish their tasks with the framework. It was also described that "the framework is enjoyable to use and pleasing to the eye", "there are many example projects that look well documented", and "tools are very convenient". Many respondents described usability problems that destroyed their enjoyment. For example, one person stated: "At the moment it feels rather frustrating. There are so many options and configurations that it is difficult to know, which things are relevant and which not".

The perceived usability of the framework supported customer loyalty. When students felt the framework was usable, they were more willing to recommend the framework (Table 6). As indicated in the qualitative results (Table 5), the usability of all platform boundary resources is crucial. In particular, development tools and information sources created usability problems that were related to installing tools, finding relevant information, and deploying applications (Table 5).

Students initially had different levels of experience in using the frameworks. Hence, a rival explanation could be that those students who were more familiar with the frameworks would rate their usability higher. Therefore, we checked whether students' experience with the frameworks correlated with their usability rating. After three weeks into the assignment, the students' previous experience was statistically significantly related to usability (r=.48**, p=.002). After the assignment was completed, the students' previous experience was no longer related to usability. Hence, in the early adoption phase, low experience was related to low experienced usability, but after more use, previous experience did not affect the evaluation.

The quality of the framework boundary resources (API, development tools, information) supported customer loyalty. When students rated the platform boundary resources better, they were more willing to recommend the framework (Table 6). Information quality was particularly related to higher loyalty, but the quality of the development tools was also strongly related. This is in line with the findings in Table 5: all platform boundary resources are important in supporting application development.

Spending more time looking for information or installing the framework affected customer loyalty negatively. The working hours spent on finding information and installing the framework were negatively related to loyalty (Table 6). In contrast, the total number of working hours was not related statistically significantly. With the Qt framework, students spent much more time looking for information and installing the framework: 12.9 hours on average compared to 4.7 hours in other frameworks. In comparison, the hours spent on using the framework to develop the application were more or less similar: 23.2 hours compared to 24.7 hours in other frameworks.

Lower enjoyment and usability ratings were due to problems encountered during use, not due to initial expectations. We also analyzed how usability and enjoyment ratings evolved over time in order to check that framework reputation or students' initial expectations were not biasing their evaluations. Figures 3 and 4 show the trends of usability and enjoyment of Qt and other frameworks. Initially, expectations of usability and enjoyment were positive for both Qt and other frameworks. Through experience, the perceived usability and enjoyment of Qt declined, but usability and enjoyment ratings for other frameworks were steady over time. This indicates that Qt developers did not have any preconceived negative prejudice toward Qt, but the lower usability and enjoyment were caused by problems encountered during use. The lower evaluations of usability and enjoyment were also reflected in the lower willingness to recommend. In the last questionnaire, the willingness to recommend was considerably lower for Qt ($M = 4.3$, SD=3.41, scale 0-10) than for other frameworks ($M = 7.6$, SD=2.11). If the percentage of detractors is subtracted from the percentage of promoters to calculate the Net Promoter Score (NPS) (Reichheld 2003), the measure of loyalty, the NPS of Qt is -57.2%, which is considerably lower than that of other frameworks (NPS 12.1%). Hence, problems encountered during use lead to lower usability, enjoyment, and customer loyalty.

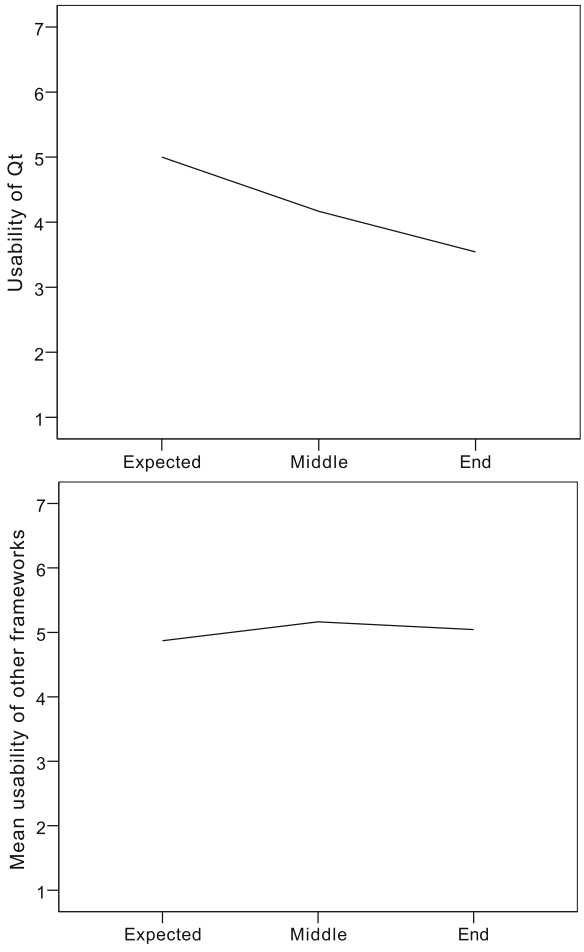

Fig. 3 Expected and experienced usability of the Qt framework (top) and other frameworks used in the student survey (bottom)

6 Discussion

In this study, we identified several characteristics of platform boundary resources that support or hinder adoption and use of the framework (Table 5). We also identified factors that support developers' loyalty after initial framework use, as indicated by their willingness to recommend the framework (Table 6). Combining resuls from the practioners' interviews and the student survey, the following proposes a number of recommendations for the framework owner to support the developer journey. Our recommendations are summarized in Table 7.

A central finding was that the needs of developers vary depending on the phase of their journey, ranging from the earliest interest to continuous use. These different phases can be called the developer journey, corresponding to the concept of a customer journey (Zomerdijk and Voss 2010). Hence, the developer journey refers to the lifecycle of the developer interacting with the framework.

When selecting a framework, both practitioners and less experienced developers justify their choice through API capabilities, but also value peer recommendation and experiences. This is in line with previous work on how customers select software products: The choice is often justified in terms of product features (Mack and Sharples 2009), and peer

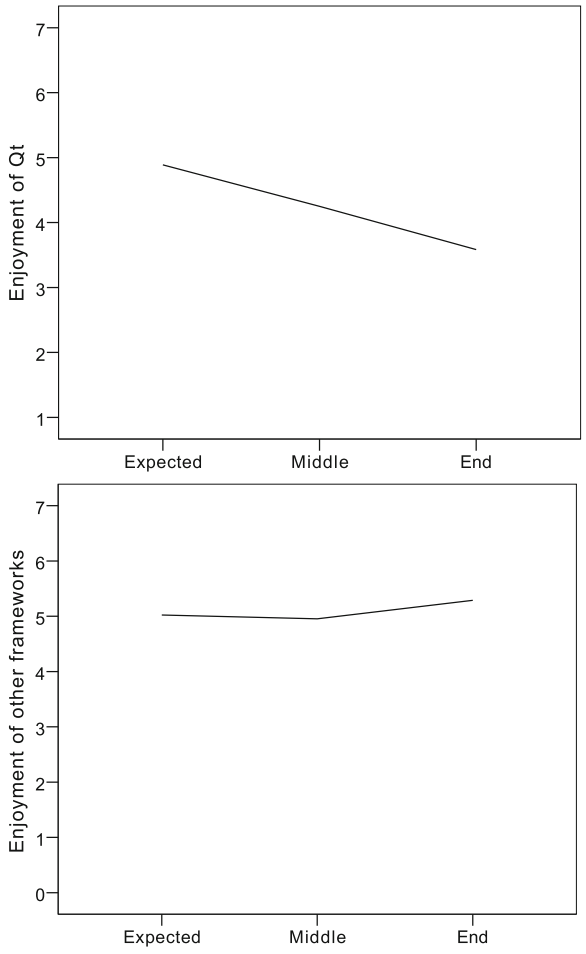

Fig. 4 Expected and experienced enjoyment of the Qt framework (top) and other frameworks used in the student survey (bottom)

Table 7 Recommendations for the framework owner to support the developer journey

1	Consider the whole developer experience, including usability and enjoyment, and all platform boundary resources.
2	Consider the whole developer journey and support developers in different journey phases.
3	Pay specific attention to framework adoption and initial use; that is, to ease of getting started, installing tools, initializing projects and deploying applications.
4	Design API capabilities and select the programming languages to match developers' know-how and application development purposes.
5	Focus on information quality, not quantity: centralize information sources, avoid redundancy, and communicate information sources and their target audience, for example, in a boundary resource map.
6	Provide both high-level documentation and low-level, e.g., method-level documentation; the high-level documentation should explain the framework's big picture.
7	Participate in building and sustaining an active community around the framework; this ensures peer help, peer recommendation and visibility of the framework.
8	Evolve the framework actively and convince developers about the framework's continuity; publishing the framework as open source is one way to do this.

experiences are important in selection (Bristor 1990). When selecting a framework, the API capabilities, the "features", are often advertised and visible. Even the existing comparisons of different frameworks (Serrano et al. 2013; Ribeiro and da Silva 2012; Shan and Hua 2006) focus on API capabilities. In contrast, the quality of the framework may be difficult to evaluate before actually trying it out. This may be the reason why peer recommendation is important. As indicated in our student survey study, the framework quality experienced affects the willingness to recommend.

However, the results of our student survey showed that when initially using a framework, the real experiences are strongly affected by the quality of the framework as a whole, not just by the APIs. In order to support adoption and initial use, it is critical to help developers to get started, install tools, initialize projects, and deploy applications. Previous work has recognized that frameworks may be difficult to learn (Fayad and Schmidt 1997), and documentation is pivotal in learning (Robillard and Deline 2011).

Another central finding was that different kinds of platform boundary resources support developers in all phases of the developer journey. Several of our findings have been previously identified in the literature: the value of source code visibility (Robillard and Deline 2011), embedded documentation in development tools (Burns et al. 2012), code examples (Burns et al. 2012; Piccioni et al. 2013; Nasehi et al. 2012), and tutorials (Tiarks and Maalej 2014). As a novel contribution, we took a broader view and explicitly identified development tools and information sources as important factors that support developer loyalty. This contrasts with previous work, which typically focuses on APIs (Section 2).

Of the platform boundary resources, information sources are important for both novice and experienced developers. Our results highlighted that developers in different phases of the developer journey need different kinds of information. For example, experienced developers were satisfied with Qt's documentation, whereas novice student developers needed information on the big picture of the framework and on how to get started. This may indicate that the technical specification of the Qt API is well documented, but information on supporting adoption and initial use could be improved. Similar findings have been made in virtual communities: having personalized and good quality information ensures satisfaction among community members (Lin 2008).

In previous work, framework information has been addressed from the API documentation point of view (Rama and Kak 2015; Burns et al. 2012; Robillard and Deline 2011), often focusing on documenting individual methods. However, besides having low-level (method-level) documentation, there also needs to be high-level, conceptual documentation (Robillard and Deline 2011). When learning the API, problems caused by high-level documentation are more severe than problems related to low-level documentation (Robillard and Deline 2011). Our student survey results confirmed this; high-level documentation is especially needed when adopting and initially using the framework.

A central finding of the student survey was that developers' customer loyalty is not only affected by the technical characteristics of the framework. Perceived enjoyment and usability, i.e., developers' feelings and subjective experiences, are strongly correlated with their loyalty to the framework, even more than different boundary resources. The importance of enjoyment is in line with the earlier research. When the use of a software system is voluntary, enjoyment experienced by the end-user is a good predictor of continuous use (Davis et al. 1992; Cyr et al. 2009; Kujala et al. 2017). As confirmed by our study,

such a finding applies when the software system is a framework and the end-user is an application developer.

Enjoyment can be seen developers' overall evaluation of the framework and how it meets their needs. Previous studies have shown that positive emotions are mostly related to user experience and negative emotions to low usability, cf. (Kujala and Miron-Shatz 2013). Thus, it is important to design for good user experience to ensure the loyalty of developers in addition to correcting usability problems of the frameworks.

The importance of usability has also been recognized previously from the API point of view (Burns et al. 2012; Rama and Kak 2015). Our results show that framework usability is also affected by development tools and information sources, in particular, during early adoption and initial use. Moreover, API usability has typically been evaluated in brief user studies with small and limited tasks (Burns et al. 2012), which may not reveal all aspects of the phenomenon. In contrast, we studied usability in a team project spanning several months.

Having an active community is crucial for a framework's success and continuity. An active community provides better peer help, since it is easier for developers to find solutions to similar problems. A community also generates peer recommendations and success stories, which motivates other developers to select the framework. When selecting a framework, a lively community creates trust in the future of the framework. Since framework selection is an investment, developers need to be convinced that the framework exists and will be evolved actively in the future. One way to expand the community and to ensure continuity is to publish the framework as open source. Open source frameworks have gained popularity; they offer documentation and support without imposing licensing fees (Johnson 2005). Nevertheless, our study did not focus specifically on how to build the community, what kind of community type suits a specific framework (Tamburri et al. 2013), or how community members interact with each other. This calls for future work.

7 Threats to validity

Construct validity as a class of validity covers whether constructs are appropriate measures of real software engineering practice. Collecting data from students may pose a threat to construct validity: how well do students understand the constraints and supporting factors of real-world framework use? To mitigate this threat, we also collected data from practitioners. As an example, practitioners pointed out the importance of community longevity, whereas future evolution and support may not be relevant for students who are focusing on one course assignment. In addition, 33% of students already had work experience in industry.

Furthermore, ambiguous and poorly-worded questionnaires are potential threats to construct validity. To mitigate this risk, we pilot-tested our questionnaires to ensure that the questions were as unambiguous as possible. Moreover, we used qualitative data to support the quantitative results and complemented the results with practitioner interviews.

For qualitative data, construct validity was mitigated by a case study protocol that defined the activities. A threat for validity is also related to interpretations of the qualitative data. We alleviated this treat by having several researchers to analyze the data and data analysis was carried out iteratively.

External validity refers to generalizations beyond the subject population of the study. Although we covered many frameworks in the survey, the focus of the interviews was on practitioners experienced in Qt. Although Qt is relatively popular and mature, Qt is also quite specific for, e.g., embedded devices so that generalization to other kinds of frameworks, such as web frameworks remains a threat to external validity. However, many frameworks also bear similarities to Qt. Generalizations beyond the examined frameworks are not studied here but seem possible and worthwhile for further assessment.

8 Conclusions

We presented a mixed-method study about supporting application developers in adopting and continuously using software frameworks. We interviewed nine experienced Qt practitioners from two companies and conducted a longitudinal student survey targeted at less experienced developers.

To conclude, developers justify framework selection with API capabilities, such as cross-platform support, and with peer experiences, such as recommendations from other developers. When developers actually adopt and initially use the framework, the quality of the development tools and information affect developer experience as well. In particular, the framework owner should make sure that information is targeted at developers, taking into account the fact that developers in different phases of the developer journey need different information. For example, developers who are new to a framework need more high-level, conceptual information, whereas experienced developers appreciate comprehensive technical documentation. Although frameworks are technical products, enjoyment and usability, i.e., feelings and subjective experiences, predict developers' customer loyalty to the framework after their initial use. Consequently, instead of just focusing on the APIs, the framework owner should ensure all platform boundary resources in the framework meet developers' needs and expectations. Besides programming, other developer tasks should be supported as well, such as installing the framework, setting up projects, and deploying the resulting applications.

As future work, a holistic view of different kinds of boundary resources throughout the developer journey could benefit from more research. It would also be beneficial to test out our findings with other software frameworks as well as discuss in more depth the similarities with software ecosystems. In particular, a survey among experienced practitioners would shed light on the generalizability of our results. Moreover, the factors that lead to both enjoyment and an experience of usability should be studied further. Our results provide some early insights about factors creating enjoyment among less experienced developers, but experienced developers may have different sources of enjoyment. Furthermore, it would be interesting to know how critical is the usability of development tools compared to the usability of APIs?

Endnotes

[1] http://www.qt.io/

[2] http://stackoverflow.com/

Abbreviations

API: Application programming interface; IDE: Integrated development environment; SDK: Software development kit; UI: User interface; QML: Qt markup language; UMUX: Usability metric for user experience; NPS: Net promoter score

Acknowledgements
We acknowledge the financial support of TEKES as part of the Need for Speed (N4S) program and OpenReq project that has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 732463. We also thank Digia and Qt Company for enabling this study.

Funding
This research is funded by the Need for Speed (N4S) program of TEKES and the OpenReq project of the European Union's Horizon 2020 research and innovation programme under grant agreement No 732463.

Availability of data and materials
Please contact author for data requests.

Authors' contributions
Authors VM, SK, MR, and PS contributed to study design, data analysis and approval of the manusript. PS and MR planned and conducted the interview study. VM, SK and MR planned and conducted the survey study. VM had the main responsibility for the manuscript and the other authors actively contributed to the manuscript. All authors read and approved the final manuscript.

Competing interests
The authors declare that they have no competing interests.

Publisher's Note

Author details
[1] Aalto University, Espoo, Finland. [2] University of Helsinki, Helsinki, Finland. [3] Digia Plc., Helsinki, Finland.

Received: 20 February 2018 Accepted: 15 May 2018
Published online: 04 June 2018

References

Bächle M, Kirchberg P (2007) Ruby on rails. IEEE Softw 24(6):105–108
Bhattacherjee A (2001) Understanding information systems continuance: An expectation-confirmation model. MIS Q 25(3):351–70
Bosch J (2009) From software product lines to software ecosystems. In: Software Product Line Conference, San Francisco. pp 111–119
Bosch J, Bosch-Sijtsema P (2010) From integration to composition: On the impact of software product lines, global development and ecosystems. J Syst Softw 83(1):67–76
Bristor JM (1990) Enhanced explanations of word of mouth communications: The power of relationships. Res Consum Behav 4(1):51–83
Burns C, Ferreira J, Hellmann TD, Maurer F (2012) Usable results from the field of API usability: A systematic mapping and further analysis. In: Visual Languages and Human-Centric Computing (VL/HCC). IEEE, Innsbruck. pp 179–182
Cyr D, Head M, Ivanov A (2009) Perceived interactivity leading to e-loyalty: Development of a model for cognitive-affective user responses. Int J Hum Comput Stud 67(10):850–869
Dal Bianco V, Myllärniemi V, Komssi M, Raatikainen M (2014) The role of platform boundary resources in software ecosystems: A case study. In: IEEE/IFIP Conference on Software Architecture (WICSA). IEEE, Sydney. pp 11–20
Davis FD, Bagozzi RP, Warshaw PR (1992) Extrinsic and intrinsic motivation to use computers in the workplace. J Appl Soc Psychol 22(14):1111–1132
Espinha T, Zaidman A, Gross H-G (2015) Web API growing pains: Loosely coupled yet strongly tied. J Syst Softw 100:27–43
Fayad M, Schmidt DC (1997) Object-oriented application frameworks. Commun ACM 40(10):32–38
Finstad K (2010) The usability metric for user experience. Interact Comput 22(5):323–7
Ghazawneh A (2012) Towards a boundary resources theory of software platforms. PhD thesis, Jönköping University
Ghazawneh A, Henfridsson O (2010) Governing third-party development through platform boundary resources. In: The International Conference on Information Systems (ICIS), Paper 48, St. Louis
Ghazawneh, A, Henfridsson O (2012) Balancing platform control and external contribution in third-party development: the boundary resources model. Inf Syst J 23(2):173–192. AIS Electronic Library (AISeL). (http://aisel.aisnet.org)
Hanssen GK, Dybå T (2012) Theoretical foundations of software ecosystems. In: International Workshop on Software Ecosystems (IWSECO), Cambridge. pp 6–17. CEUR Workshop Proceedings. (http://ceur-ws.org/)
Hanssen GK (2012) A longitudinal case study of an emerging software ecosystem: Implications for practice and theory. J Syst Softw 85(7):1455–1466
Jansen S, Finkelstein A, Brinkkemper S (2009) A sense of community: A research agenda for software ecosystems. In: International Conference on Software Engineering, Companion. IEEE, Vancouver. pp 187–190
Johnson RE (1997) Frameworks = components + patterns. Commun ACM 40(10):39–42
Johnson R (2005) J2EE development frameworks. Computer 38(1):107–110
Kitchenham B, Pfleeger SL (2002) Principles of survey research part 4: questionnaire evaluation. ACM SIGSOFT Softw Eng Notes 27(3):20–23
Kujala S, Miron-Shatz T (2013) Emotions, experiences and usability in real-life mobile phone use. In: SIGCHI Conference on Human Factors in Computing Systems. ACM, Paris. pp 1061–1070
Kujala S, Mugge R, Miron-Shatz T (2017) The role of expectations in service evaluation: A longitudinal study of a proximity mobile payment service. Int J Hum Comput Stud 98:51–61
Lazar J, Feng JH, Hochheiser H (2010) Research methods in human-computer interaction. Wiley, Glasgow

Lin H-F (2008) Determinants of successful virtual communities: Contributions from system characteristics and social factors. Inf Manag 45(8):522–527
Mack Z, Sharples S (2009) The importance of usability in product choice: A mobile phone case studylisbon, portugal. Ergonomics 52(12):1514–1528
Manikas K (2016) Revisiting software ecosystems research: A longitudinal literature study. J Syst Softw 117:84–103
Manikas K, Hansen KM (2013) Software ecosystems—a systematic literature review. J Syst Softw 86(5):1294–1306
Messerschmitt DG, Szyperski C (2003) Software Ecosystem: Understanding an Indispensable Technology and Industry. The MIT press, Cambridge
Mitchell TR, Thompson L, Peterson E, Cronk R (1997) Temporal adjustments in the evaluation of events: The "rosy view". J Exp Soc Psychol 33:421–448
Nasehi SM, Sillito J, Maurer F, Burns C (2012) What makes a good code example? A study of programming Q&A in StackOverflow. In: International Conference on Software Maintenance. IEEE, Trento. pp 25–34
Piccioni M, Furia C, Meyer B, et al (2013) An empirical study of API usability. In: International Symposium on Empirical Software Engineering and Measurement. IEEE, Baltimore. pp 5–14
Ployhart RE, Ward A-K (2011) The "quick start guide" for conducting and publishing longitudinal research. J Bus Psychol 26(4):413–422
Rama GM, Kak A (2015) Some structural measures of API usability. Softw Pract Exper 45(1):75–110
Ratiu D, Jurjens J (2008) Evaluating the reference and representation of domain concepts in APIs. In: IEEE International Conference on Program Comprehension. IEEE, Amsterdam. pp 242–247
Reichheld FF (2003) The one number you need to grow. Harward Business Rev 81(12):46–54
Ribeiro A, da Silva AR (2012) Survey on cross-platforms and languages for mobile apps. In: Conference on the Quality of Information and Communications Technology (QUATIC). IEEE, Lisbon. pp 255–260
Robbes R, Lungu M (2011) A study of ripple effects in software ecosystems. In: International Conference on Software Engineering. ACM, Waikiki. pp 904–907
Robillard MP, Chhetri YB (2015) Recommending reference API documentation. Empir Softw Eng 20(6):1558–1586
Robillard MP, Deline R (2011) A field study of API learning obstacles. Empir Softw Eng 16(6):703–732
Seaman C. B (1999) Qualitative methods in empirical studies of software engineering. IEEE Trans Softw Eng 25(4):557–572
Schäfer T, Jonas J, Mezini M (2008) Mining framework usage changes from instantiation code. In: International Conference on Software Engineering. IEEE, Leipzig. pp 471–480
Serrano N, Hernantes J, Gallardo G (2013) Mobile web apps. IEEE Softw 30(5):22–27
Shan TC, Hua WW (2006) Taxonomy of Java web application frameworks. In: International Conference on e-Business Engineering (ICEBE). IEEE, Shanghai. pp 378–85
Strauss A, Corbin J (1998) Basics of Qualitative Research. 2nd edn. Sage, Thousand Oaks
Stylos J, Myers BA (2008) The implications of method placement on API learnability. In: Foundations of Software Engineering. ACM, Atlanta. pp 105–112
Tamburri DA, Lago P, Vliet HV (2013) Organizational social structures for software engineering. ACM Comput Surv (CSUR) 46(1):3
Taylor R (2013) The role of architectural styles in successful software ecosystems. In: Software Product Line Conference. ACM, Tokyo. pp 2–4
Tiarks R, Maalej W (2014) How does a typical tutorial for mobile development look like?. In: Working Conference on Mining Software Repositories. ACM, Hyderabad. pp 272–281
van Angeren J, Alves C, Jansen S (2016) Can we ask you to collaborate? Analyzing app developer relationships in commercial platform ecosystems. J Syst Softw 113:430–445
Zomerdijk LG, Voss CA (2010) Service design for experience-centric services. J Serv Res 13(1):67–82