On the Challenges Novice Programmers Experience in Developing IoT Systems: A Survey

On the Challenges Novice Programmers Experience in Developing IoT Systems: A Survey

Fulvio Corno, Luigi De Russis, and Juan Pablo Sáenz

Corso Duca degli Abruzzi 24, Torino

Abstract

The co-existence of various kinds of devices, protocols, architectures, and applications make Internet of Things (IoT) systems complex to develop, even for experienced programmers. When novice programmers are learning to implement these systems, they are required to deal with areas in which they do not have a deep knowledge. Furthermore, besides becoming proficient in these areas separately, they should integrate them and build a system whose components are heterogeneous from both software and hardware perspectives.

The accurate understanding of the most challenging issues that novices face is fundamental to envision strategies aimed at easing the development of IoT systems. This paper focuses on identifying such issues in terms of software development tasks that novice programmers encounter when working on IoT systems. To this end, a survey was conducted among 40 novice developers that worked in groups developing IoT systems during several years of a university course. Based on their own experiences, individually and as a group, the most challenging development tasks were identified and prioritized over a common architecture, in terms of difficulty level and efforts. In addition, qualitative data about the causes of these issues was collected and analyzed. Finally, the paper offers critical insights and points out possible future work.

Keywords: Novice programmers, Internet of Things, Software engineering, Survey

Email address: fulvio.corno@polito.it, luigi.derussis@polito.it, juan.saenz@polito.it (Fulvio Corno, Luigi De Russis, and Juan Pablo Sáenz)

1. Introduction

The Internet of Things (IoT) is built upon the idea of embedding computing and communication capabilities into objects of common use [1]. This paradigm gives rise to a programmable world and encourages the development of a broad range of solutions in diverse domains, ranging from smart homes to healthcare or logistics. As IoT systems rapidly gain prominence in several aspects of our everyday lives [2], so does the interest of academia and industry towards the need of supporting developers [3] and preparing different stakeholders[1] [4] to shape the future directions of IoT.

However, software engineering for the IoT poses several challenges regarding infrastructure, communications, interfaces, protocols, and standards [5]. These challenges encompass handling *heterogeneous* devices [3], exchanging data among them, providing localization and tracking capabilities, and enabling these devices to make simple decisions [6]. Just from the software perspective, the co-existence of various kinds of devices, protocols, architectures, and programming languages requires knowledge of various and disparate areas. Database design, mobile development, web development, embedded system development, authentication mechanisms, APIs design, and application level protocols are some examples of the areas that are typically involved in the implementation of IoT systems.

The inherent complexity of such IoT systems raises particular concerns if we focus on *novice programmers* and how to effectively and easily allow them to design and develop these systems. When novice programmers are working on these kind of systems, indeed, they are required to deal, conceptually and technically, with several development tasks in different areas, without having a deep knowledge nor previous experience in any of them. Furthermore, besides

[1] defined as in software engineering, i.e., people who are involved in any phases of the software development process.

becoming proficient in these areas separately, they are required to orchestrate the related tasks, and to build a system with heterogeneous components, both from the software and hardware perspectives. For these reasons, an accurate understanding of the most challenging issues that novices experience is fundamental to envision strategies to enable a smoother development of IoT systems. To our knowledge, no previous work assessed which are these issues.

Prior work on the topic of easing the development of IoT systems has focused on providing suitable *methodologies* and *frameworks* to *professional* developers [7, 8], starting from a few challenges extracted from unstructured analysis of IoT applications, at best [3].

In an effort to understand which challenges *novice developers* face when building IoT systems, in late 2017 we conducted a survey involving 40 novice programmers coming from an engineering background. Such developers were recruited among the former students of an undergraduate course, during which they worked in groups to develop different IoT systems. Naturally, the systems taken into account in this work are not large-scale systems but prototypes with didactic purposes. In particular, in this article, we report on the following three research questions addressed by the survey:

RQ1: How complex, in terms of time spent and difficulty, are the software development tasks needed to build an IoT system?

RQ2: Which are the software development tasks that are perceived as the most challenging to complete?

RQ3: Why are these tasks perceived as the most challenging?

This survey contributes to the body of research on easing the development of IoT systems, with a focus on novice programmers. Here, "IoT systems" is meant broadly and encompasses several application domains (e.g., healthcare, smart home, ...) as well as different IoT devices and technologies. For this reason, in the survey, a generic architecture for IoT systems was used as a reference, for providing a common vocabulary to the respondents. The survey

results present insights about novices' experiences when working on IoT systems, and reveal that the integration of those parts of an IoT system that require over-the-network communications is one of the most challenging tasks. The outcomes also reveal that the interaction and interfacing with third-party cloud services is perceived as an important issue to be tackled, mainly for the lack of proper documentation and examples. Overall, the results can inform the design and development of adequate methodologies and improved tools to ease the development of IoT systems in general, and for novice programmers in particular. Furthermore, we consider that the results of this survey, obtained from respondents belonging to the academic setting, might be partially valid as well to software companies, in particular by considering the work by Salman et al. [9], about how well students represent professionals in software engineering experiments. According to this work, a major differentiating factor affecting the results might be subject's experience levels rather than the experiment setting (classroom or industry).

The remainder of the paper is structured as follows. Section 2 presents the related work and provides some background information. Section 3 describes the survey design and research methods, while Section 4 reports the results. A discussion of the results, implications, and limitations are presented in Section 5. Section 6 discusses in detail the threats to validity of the survey and its results. Finally, Section 7 concludes the article.

2. Background and Related Work

This work builds upon and lies at the intersection of research in two related topics: *(i)* easing the development of IoT systems (Section 2.1) and *(ii)* novice programmers in the IoT (Section 2.2). Finally, in Section 2.3 we present works aimed at identifying the challenging issues that novice or experienced developers face in other areas of software development that are commonly involved as enabling technologies in the implementation of IoT systems. Specifically, we considered mobile development, APIs development, and programming, in

general.

2.1. Easing the Development of IoT Systems

Taivalsaari et al. [10] present a roadmap from today's cloud-centric, data-centric IoT systems to a world in which everyday use objects are connected and the network's edge is programmable. On the basis of the authors' experience, they highlight issues and technical challenges that the *Programmable World* poses to software developers. In their opinion, the average mobile or client-side web application developer is not well equipped to cope with the challenges of IoT systems development. Moreover, today's development methods, languages, and tools are poorly suited to the emergence of millions of programmable things. In particular, IoT developers must consider several dimensions that are unfamiliar to mobile and client-side web application developers, namely: multidevice programming; the reactive, always-on nature of the system; heterogeneity and diversity; the distributed, highly dynamic, and potentially migratory nature of software; and the need to write software in a fault-tolerant and defensive manner. Among all the statements discussed in that article, two of them are of special relevance:

- educating software developers to realize that IoT development truly differs from mobile and client-side web application development;
- to harness the Programmable World's full power, we will need new software engineering and development technologies, processes, methodologies, and tools.

Patel and Cassou [3] tackle the challenges brought by application development in the IoT by proposing an "high-level" development methodology that separates IoT application development into different concerns and provides a conceptual framework to develop an application. They recognize that software development in the IoT present various challenges, and they list four of them: *a)* lack of division of roles, *b)* heterogeneity (of devices), *c)* scale (of IoT systems), and *d)* different life cycle phases. However, these challenges were formulated

starting from the authors' unstructured analysis of example applications and from related work in closely related fields like Wireless Sensor Networks and Ubiquitous Computing [11]. The main goal of the work of Patel and Cassou is, indeed, to make IoT application development easy for stakeholders by taking inspiration from the MDD approach and building upon work in sensor network macroprogramming, thus reducing development efforts.

Datta and Bonnet [8], similarly, start from their own experience (i.e., from the IoT data cycle presented in [12]) to propose a list of the top 8 requirements for building an IoT application framework: interoperability, open source framework, strong security by design, etc. Then, they introduce DataTweet, a framework that decouples application logic from common IoT functionalities. This allows IoT stakeholders to focus on the application logic and use open source, standardized APIs for the latter. An example with an automotive IoT application for an Advanced Driver Assistance System was developed with the framework and its operational phases were highlighted in the paper. The framework aimed at simplifying the development process, hiding the complexities of programming and security mechanisms from developers, and reducing time to market for industries.

According to Weyrich and Ebert [13], too, software engineering for the IoT poses challenges in light of new applications, devices, and services. Moreover, such new and diverse applications, devices, and services "*pose specific challenges for specifying software requirements and developing reliable, safe software*" [13]. Weyrich and Ebert, in their paper, state that reference architectures may help developers meet those challenges. They focus on two major architectures from an industry standpoint: the *Internet of Things - Architecture* (IoT-A)[2] and *Industrial Internet Reference Architecture* (IIRA)[3]. IoT-A delivered a detailed architecture and model from the functional and information perspectives, while IIRA was delivered by the Industrial Internet Consortium (founded by AT&T,

[2] `https://cordis.europa.eu/project/rcn/95713_en.html`, last visited on May 24, 2019.
[3] `https://www.iiconsortium.org/IIRA.htm`, last visited on May 24, 2019.

Cisco, General Electric, IBM, and Intel) for a broad consideration and discussion. Such architectures can serve as an overall and generic guideline, and not all domain applications will require every details for real-life development. While such reference architectures are not equal and a "standard" architecture did not yet prevail, they form a superset of functions, information structures, and mechanisms that could provide developers with a more complete view of the IoT system they will implement.

In this paper, survey respondents were referred to a "generic architecture", which shares most of the functionalities and building blocks with the two aforementioned architectures and was customized to the type of IoT projects they had experience on. Moreover, it includes contributions coming from other IoT reference architectures, namely the Intel IoT Platform Reference Architecture [14], the IBM IoT Reference Architecture[4], and the Microsoft Azure IoT Reference Architecture [15]. As already highlighted by Weyrich and Ebert [13], all these architectures are not equal but they present some common features. The purpose of using this generic architecture was to provide respondents with a common understanding about the software components involved in an IoT system.

To our knowledge, there are no previous works assessing which are the most challenging issues faced by IoT novice programmers according to a concrete set of software development activities. The reported related works base their approaches on the authors' expertise, mainly. Additionally, the majority of the frameworks and toolkits for easing the development of IoT systems are constrained to a particular technological stack [7, 3, 8, 16, 17]. Instead, supported by our proposed generic architecture, our work aims at gaining a better understanding of such issues independently from the projects, the architectural decisions, and the technology stack. Moreover, these issues are expressed as concrete software development tasks that belong to a specific part of the architecture.

[4] `https://www.ibm.com/cloud/garage/architectures/iotArchitecture/reference-architecture/`, last visited on May 24, 2019.

2.2. Novice Programmers in the IoT

Literature on *novice programmers* in the IoT mainly consists of experience reports from college-level courses, in which teachers and instructors recognize the needs and challenges brought by the IoT and intervene either with new methodologies or with dedicated frameworks. However, a systematic collection and description of pain points and issues encountered by these novice developers was not performed in any of these works, to our knowledge. We report, here, four of the most representative works.

To provide its students with the systems-level skills needed to understand and develop complete IoT systems, in 2014 Politecnico di Torino, in Italy, initiated a course named "Ambient Intelligence." In this project-based course, a teamwork and design-driven *methodology* is applied to teaching IoT system design [4]; core student skills acquired in previous courses are exploited in a multidisciplinary project work. The main topic of the course is the design and the implementation of prototype Ambient Intelligence (AmI) systems [18], a field closely related to the IoT. This entails a strong focus on the application and on user needs. From the beginning of the course, students form three- to four-people teams and are guided to define the requirements for a system, and then to design and implement it. Every year, a theme is chosen for the projects. The theme is wide enough to generate around 20 projects, but sufficiently well-defined to determine whether a project fits. After teacher's approval, the teams develop their ideas according to the proposed design methodology, which follows four main steps: vision and goal definition; functional and non-functional requirements elicitation; system architecture design and component selection; and practical realization of the prototypical system. Projects cannot be mobile-only, software-only, or hardware-only solutions. Instead, they must exploit different platforms and mix hardware with software and user interaction, as typical IoT systems do. The resulting system and the "deliverables" produced throughout the semester are the focus of the course exam, which also includes a presentation of the team projects and an oral discussion. The authors, in their paper, provide positive qualitative and quantitative results about the students'

ability to understand and design IoT systems; the usage of required languages, frameworks, and protocols; and employed communication, collaboration, and management skills. Moreover, they present a series of "lessons learned" that may allow other instructors to design IoT-related courses by following a similar methodology. Indeed, the subject of the survey that we are presenting in this work is a subset of students of the Ambient Intelligence course.

In a similar way, Kortuem et al. [19] describe the experience of the Open University, in the United Kingdom, delivering an online course whose purpose was to "place the IoT at the core of the first-year computing curriculum and to prime students from the beginning to meet the coming changes in society and technology". Among the concepts that the course designers identified as fundamental for the IoT and essential for the course, they list: the merging of the physical and digital realms; the huge increase in the number of Internet-connected devices, objects, sensors, and actuators; and the emergence of novel embedded-device platforms below the level of personal mobile devices. A key goal of the course was to empower novices and to make IoT technologies accessible to students with no prior programming skills. One of the most challenging issues faced when designing and delivering the course was that most embedded device technologies require an understanding of software and hardware that cannot be expected from first-year undergraduates. To overcome this issue, an embedded networked sensor was custom-designed for this course, as well as a newly developed visual programming language and environment, and a cloud infrastructure that connected the embedded networked sensors of all students together. Authors determined, based on programming assignments and tests with prospective users during the design stage of the course, that new users can produce a working program in less than 20 minutes during their first session with the custom embedded networked sensor. Furthermore, after a few sessions, novices with no exposure to programming before the course, could understand and modify given programs and develop new ones on their own. Moreover, their proposed programming language and environment help novices to quickly develop an understanding of the principles of programming simple

IoT applications. To gain an understanding of the common issues that novices experience, authors conducted a preliminary analysis based on the activity of the help forum. However, the article just provides a very general description of the issues that concerned their proposed IoT teaching infrastructure.

Dobrilovic et al. [20] built a platform to teach communication systems, and designed a second one [21] to be used in university curricula for teaching IoT. The first platform was built to be used within the curricula of Information Technology and Software Engineering, where a strong background in electronics is not expected. The basis of the platform was built upon an Arduino micro-controller and includes Zigbee expansion shields, different types of sensors, and a packet sniffer specially developed for analyzing ZigBee, Bluetooth Low Energy, and IEEE 802.15.4 traffic. Moreover, the authors described three scenarios for the usage of this platform: a system for temperature monitoring, an RFID/ZigBee network for tracking human resources, and a smart agriculture and air pollution monitoring system. Starting from the architecture proposed for each scenario, students were asked to deploy such scenarios from the beginning, and develop applications on top of them. Specifically, the platform was used by a group of three students that were able to deploy the temperature monitoring scenario by adding more sensors and developing an application to gather real-time data from an Arduino micro-controller. The second platform [21] is proposed upon an open-source architecture for teaching IoT. It consists of a set of low-cost open-source hardware components (*IoT education kit*), along with the list of software components required to develop IoT custom applications, and the network protocols required to establish the communication between the layers of the proposed IoT teaching architecture. However, this platform was not used during laboratory exercises. Instead, it was presented to students as a part of the lectures, aiming at explaining the functionality and implementation of each layer of the architecture. Therefore, students' feedback was not collected in a formal questionnaire nor analyzed. The platform acceptation and effectiveness was assessed based on the positive comments that the students made. According to the authors, students accepted the platform with good

attention and interest to work with it.

2.3. Identifying programmers issues

Below we present a set of related works that relied on interviews, surveys, and controlled studies with software developers to identify the challenging issues present in mobile applications development, APIs usage, and learning to program. These related works concern areas of software development that are commonly involved in the implementation of IoT systems. For instance, mobile applications are generally the mean by which end users interact and configure the whole system; similarly, the integration between subsystems is typically achieved through RESTful APIs; and naturally, the implementation of the system may require programming expertise in more than one programming language.

Ahmad et al. [22] aimed at identifying the challenges that can undermine the successful development of native, web, and hybrid mobile applications. First, the authors identified the challenges through a systematic literature review, and then, they validated the challenges through interviews with practitioners. From the systematic literature review, nine challenges emerged, and from the interviews, four additional challenges were identified. In these interviews, 34 mobile developers with 2-5 years of experience were recruited and instructed to rate each challenge on a Likert scale. Interestingly, the distinction between the three types of mobile applications (native, web, and hybrid), that we also considered in our work, enabled the authors to accurately identify the most critical challenges on each type. Indeed, after comparing the development challenges of these three types of mobile applications, the authors determined that there are statistically significant differences among them. Concretely, fragmentation and change management are more critical in native, the user experience is more critical on the web, and compatibility is more severe on the web. As we will describe later in this article, the results of our survey are consistent with the fact that challenges vary according to the type of mobile application.

Joorabchi et al. [23] aimed at gaining an understanding of the main chal-

lenges developers face in practice when they build apps for different mobile devices. To that end, they first conducted a qualitative study consisting of interviews with 12 expert mobile developers, and then, they carried out a semi-structured survey with 188 respondents from the mobile development community at large. Authors identified the existence of multiple mobile platforms as a major challenge for developing mobile apps; developers are required to learn more languages and APIs for the various platforms and, at the same time, remain up to date with the frequent changes of each Software Development Kit (SDK). Additionally, due to this fragmentation, developers have to keep checking the correctness and consistency of the app across different platforms. Developers also indicated that testing tools and emulators (at the time in which this study was conducted) were not able to sufficiently support important features and scenarios such as mobility (changing network connectivity), location services, sensors, or various gestures and inputs. This lack of tools makes analysis and testing even more challenging. Finally, concerning usability, the study results suggested that the implementation of a reusable user interface is challenging due to the trade-offs that developers are required to achieve between maintaining consistency and adhering to each platform's standards. In this respect, the results obtained from our survey suggest that the configuration of the development environment (involving dependencies, SDKs, and run-time platforms) is perceived as challenging in the mobile application development.

Sohan et al. [24] conducted a controlled study with 26 experienced software engineers to understand the issues that REST API client developers face while using an API without examples. To that end, participants were divided into two groups and given the same set of 6 API tasks to complete. While one group was given the official REST API documentation, the other group was given an enhanced version of the official documentation where three usage examples were added. From the analysis of 539 API calls, 385 from the first group and 152 from the second, authors determined that, without examples, REST API client developers struggle with using the right data types, data formats, and required HTTP requests headers.

Similarly, Robillard et al. [25] conducted a study aimed at identifying some of the most severe obstacles faced by experienced developers, with an average of 9.8 years of professional experience, when learning new APIs. Such study involved 440 professional developers and was structured around: (i) an exploratory survey to broadly identify *what makes APIs hard to learn*; (ii) a set of qualitative interviews to understand API learning obstacles in detail; and (iii) follow-up survey to confirm the general findings and collect additional demographic data that would help to explain API learning obstacles. The study identified inadequate API documentation as the most severe obstacle facing developers learning a new API. For this reason, based on the qualitative analysis, the authors elicited a set of important factors to consider when designing API documentation. Among their various observations, they stated that *small examples that nevertheless demonstrate API usage patterns involving more than one method call will be more useful than single-call examples.* Furthermore, they determined that *a central challenge when learning APIs is discovering how to match scenarios with the API elements that support this scenario.*

Uddin et al. [26] conducted two surveys about API documentation quality involving 323 software professionals. In the first survey (exploratory), authors aimed at collecting good and bad examples of API documentation. The respondents were asked to provide examples of good or bad documentation, based on the last development task that they completed, in which they had to consult API documentation. In the context of this exploratory study, there was no standard definition for an API; it could be a library, a framework component, or even a Web API. In the exploratory survey participants were asked to: (i) describe their last development task they had completed that required them to consult API documentation; (ii) provide up to three examples of API documentation that they found useful their corresponding justification; (iii) provide up to three examples of API documentation that they did not find useful and their justification. From the analysis of the results, ten common documentation problems emerged, and they were categorized by the authors into content and presentation problems. Namely, content problems comprised incompleteness, ambiguity,

unexplained examples, obsoleteness, inconsistency, and incorrectness. Presentation problems, for their part, concerned bloat, fragmentation, an excess of structural information, and tangled information. In the second survey (validation), the authors assessed the frequency and severity of the previously identified problems. This validation survey was conducted with a different group of participants, and they were asked to (i) rate, for each one of problems, how frequently they were experienced and how severe they were when completing the participants' development tasks; and (ii) to identify the three most painful problems to be prioritized. In this manner the authors analyzed the problems' frequency, their severity, and the necessity to solve them. Ambiguity and incompleteness were identified as the most critical problems, and together with incorrectness, they were into the top three priorities for improving documentation. The major finding of the surveys was that the most frequent and common problems had to do with content. In fact, all content problems were prioritized over presentation problems. Additionally, the hardest problems with API documentation were also the ones requiring the most technical expertise to solve. Completing, clarifying, and correcting documentation require deep, authoritative knowledge of the API implementation. Finally, the authors envisioned recommendation systems as a mean to reduce as much of the administrative overhead of documentation writing as possible, enabling experts to focus exclusively on the value-producing parts of the development tasks.

Concerning these last three articles, the results from our survey identify the lack of documentation understandable by novices as one of the causes of the complexity behind subsystems integration.

Koulouri et al. [27] assessed the effect of three factors on learning to program, namely: choice of programming language, problem-solving training, and the use of formative assessment. To that end, the authors conducted a study that adopted an iterative approach and was carried out across four consecutive years involving four experimental groups of CS1 students. These groups corresponded to distinct full student cohorts and were organized in this manner: a control group that was taught using Java (157 students), a group that

was taught using Python (195 students), a group that received formative feedback (193 students), and a group that received initial problem-solving training (216 students). From the iterative process, the following outcomes emerged: (i) the choice of programming language seems to affect student learning, a simpler syntax could have a greater impact because it makes loops easier to use, and the underlying concept easier to understand; (ii) introducing problem-solving concepts before teaching more specific programming aspects has an impact on how students learn to program, it helps students to develop an ability to both break down complex problems into subtasks and produce the correct sequence of actions while accelerating the consolidation of concepts, such as data and control structures, introduced later in the course; (iii) Formative feedback may not be necessarily and effective as expected unless students are ready to have a proactive role in seeking and responding to feedback, it is advised that for formative feedback to yield observable benefits on their performance, novice programming students may need to be externally motivated and guided. The results of our survey also suggest that developers struggle with the development of the business logic within the various components involved in an IoT system.

3. Survey Design and Methods

Our survey was conducted to identify, based on the personal experience of novice developers, the most complex issues that they faced when developing an IoT system as well as the principal causes of such complexity. The survey was advertised and conducted online, and both quantitative and qualitative data were gathered. Hereafter we will present in detail the methodology of the survey.

3.1. Instrument development

As already mentioned, the term "IoT systems" in the context of this study is meant broadly and encompasses several application domains, as well as diverse IoT devices and technologies. For this reason, in the development of the

survey, it was imperative to provide the respondents with a common understanding about the software components involved in an IoT system. Hence, we structured the survey around a generic IoT systems architecture proposed by us along with a predefined set of software development tasks compatible with such architecture.

Such generic architecture was built, firstly, by taking into account the functionalities and building blocks suggested in the IoT reference architectures mentioned in the Related Work (Section 2.1), and secondly, by analyzing the architectures used in the development of prototype IoT systems during several years of the Ambient Intelligence course (described in Section 2.2), whose students were the subject of our study. In this manner, we made sure that the resulting common architecture would be understandable and familiar to the IoT novices participating in our study.

Consequently, five interconnected subsystems were characterized, as illustrated in Fig. 1. The resulting subsystems are:

Sensors monitor the End-user activities and detect changes in the environment by measuring variables such as temperature, humidity, and occupation, among others. They generally refer to wearable devices and environmental sensors.

Gateways gather the data coming from the Sensors and perform computation and reasoning tasks over it. If more computing or storage capacity is required, the Gateways communicate with the Back-End subsystem and delegate the most demanding tasks. Furthermore, Gateways also interact with the actuators. They control the acting devices based on the outputs from their own computations or based on the instructions that they receive from the Back-End subsystem. In the projects developed by the novice IoT developers, this subsystem typically consisted of single-board computers such as Raspberry Pis.

Back-end groups third-party services APIs, the main application server, and the persistence component. The functionalities provided by the applica-

tion server and the persistence component are typically exposed to the Gateways subsystem through RESTful web services. Finally, third-party service APIs are commonly used to interact with the wearable devices belonging to the Sensors subsystem.

Actuators span actuating devices that trigger changes in the physical environment. However, they also encompass push notifications through which end-users are informed about the occurrence of a given event. Acting devices are generally controlled by gateway devices via Bluetooth or Wi-Fi, while push notifications are commonly generated in the Back-End subsystem through the Android and iOS push notifications platforms APIs.

End-user refers to the interfaces with which the End-users are enabled to interact with the IoT system. These interfaces typically consist of mobile and web applications through which user preferences can be configured, Actuators can be activated or deactivated, and Sensors can be monitored and managed.

An important component that we decided not to represent in our generic architecture is **Security**. This choice was mainly made because it was outside the course syllabus. Therefore, survey respondents were not exposed to the issues and possible pain points that could be generated from security-related operations.

However, since our research questions concern the software development perspective of IoT systems, only the subsystems whose implementation and integration with other subsystems relied mainly on software development activities, were considered in this work. These software-intensive subsystems were: End-user, Gateways, and Back-end. Next, these three subsystems were "decomposed" into a list of software development tasks required for their implementation (e.g.,*Develop a native end-user mobile application*). These tasks, in turn, were decomposed into very punctual, unambiguous sub-tasks (e.g.,*Become familiar with the mobile application platform-specific programming language*).

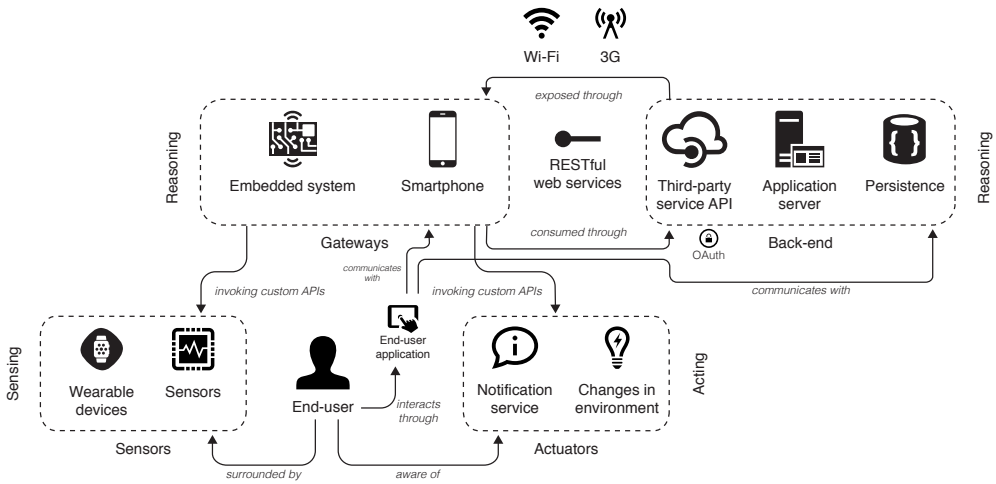


Figure 1: Proposed IoT systems reference architecture

As Gateways and Back-end subsystems resulted in a quite large number of tasks, these subsystems were split up in two: the first one for the subsystem development tasks, and the second one for the subsystem integration tasks. As a result, the survey was developed starting from the generic architecture and the set of tasks and sub-tasks that emerge from each identified subsystem.

3.2. Initial generation of question and answer options

We defined the survey structure in line with the research questions. The subsystems were mapped to a set of sections in the questionnaire, as shown in Table 1. Five sections were therefore defined, namely: End-user subsystem (Section A), Gateways subsystem development (Section B), Gateways subsystem integration (Section C), Back-end subsystem development (Section D), and Back-end subsystem integration (Section E). Since the sections belonging to the End-user subsystem were mutually exclusive as they corresponded to the three types of mobile applications, the survey was structured in 5 sections, with 24 tasks, and 67 sub-tasks.

Table 1: Survey structure

Subsystems	Sections	# Tasks	# Sub-Tasks
End-user	Section A1: Native mobile application	9	10
	Section A2: Hybrid mobile application		10
	Section A3: Web responsive mobile application		9
Gateways	Section B: Gateways development	5	10
	Section C: Gateways integration	3	10
Back-end	Section D: Back-end development	4	10
	Section E: Back-end integration	3	8
Total		24	67

As not all respondents participated in the development of every subsystem of the IoT system, the research questions were addressed at the subsystem level of detail. In fact, inside each subsection, they were instructed to skip the rating of the sub-tasks in which they did not participate. Therefore, for each section of the questionnaire, respondents were asked to:

- **Rate** the complexity of each sub-task in which they participated according to their difficulty level and the time spent completing them. Two Likert scales ranging from 1 to 5 were included in the questionnaire for each sub-task, as shown in Figure 2. This **rating** aims at answering the **RQ1**: How complex, in terms of time spent and difficulty, are the software development tasks needed to build an IoT system?

- **Rank** the sub-tasks of the concerned subsystem identifying the ones perceived as the most challenging to complete. In this case, such sub-tasks had to be ranked as the first, second and third most difficult task in the subsystem. In Figure 2, the field at the left of the sub-tasks is intended to the rank the three most challenging sub-tasks. This **ranking** aims at answering the **RQ2**: Which are the software development tasks that are perceived as the most challenging to complete?

- Assess the **perception** of the respondents about the reasons behind the

ranking choice on each subsystem. This perception is captured through an open question, where besides their justification, respondents could also mention any other task that they found complex to achieve, even if it was not in the set of suggested sub-tasks. The qualitative **perceptions** of the participants are intended to answer the **RQ3**: Why are these tasks perceived as the most challenging?

RQ1 (rate) was measured at the sub-task level, while RQ2 (rank) and RQ3 (perception) were measured at each section level.

Rank	Section A: End-user	Difficulty					Time spent				
Develop a native end-user mobile application											
	Become familiar with the mobile application platform-specific programming language	1	2	3	4	5	1	2	3	4	5
	Configure the development environment	1	2	3	4	5	1	2	3	4	5
	Develop the models' classes	1	2	3	4	5	1	2	3	4	5
	Develop the controllers' classes	1	2	3	4	5	1	2	3	4	5
	Develop the user interface (views)	1	2	3	4	5	1	2	3	4	5
	Connect the push notification module with the platform notification service	1	2	3	4	5	1	2	3	4	5
	Handle the notifications received in the end-user's smartphone	1	2	3	4	5	1	2	3	4	5

Figure 2: Example of a task decomposed in sub-tasks

3.3. Initial pilot survey

To validate the pertinence and completeness of the resulting survey, a preliminary study [28] was conducted and documented with a small group of participants (6). These participants belonged to the 2016 cohort of the previously mentioned Ambient Intelligence course. In this version of the course 18 projects related to Health and Well-Being were developed. The participants chosen for the pilot survey were the members of two groups whose final projects obtained outstanding grades, and whose implementation relied mainly on software development activities. The first group was composed by 4 students and the second group was composed by 3 students. However, one of the members of the second group was an international student who returned to her home university, therefore a total of 6 students were involved in the preliminary study.

The pilot was conducted by inviting the respondents to 2 interview sessions, with one group invited per each session. Each session consisted of three phases: an introduction, the questionnaire, and a discussion. The sessions were conducted by two researchers, and were held in English. In the introduction, one researcher briefly explained the objective of the study, the structure of the questionnaire and the general organization of the session. It was clarified that the questionnaire had to be filled individually from the personal point of view (each participant should respond to those activities in which they were directly involved, only), while the following discussion would involve their evaluation as a group.

The questionnaire was filled out on paper, and as described before, participants were asked to rate the sub-tasks according to their difficulty level and the time spent completing them, rank the three most difficult tasks per each section of the questionnaire, and justify their ranking choice with an open question, where they could also mention any other tasks that were not listed but resulted complex to achieve.

After all participants completed the questionnaire, a final discussion was held to identify, as a group, the most complex and painful tasks. The respondents were free to discuss among themselves, and with the researchers. The completion of the questionnaire took each participant, in average, approximately 30 minutes, while the later discussion about the most painful issues and the feedback about the completeness of the questionnaire took around 20 minutes.

Besides providing some preliminary insights about the most painful issues when developing IoT systems, this study provided valuable feedback regarding the pertinence and completeness of the proposed generic architecture, the identified subsystems, and their tasks and sub-tasks. The participants of this preliminary study did not suggest any modification to the architecture nor the addition of sub-tasks that the questionnaire could have overlooked.

3.4. Survey instrument

The structure of the questionnaire in terms of sections and their tasks is shown in Table 2, which also reports the number of sub-tasks defined per each task.

3.5. Administration and population

The survey was managed through the Lime Survey [29] platform, and invitations were sent by email to former students of the three cohorts of the Ambient Intelligence course between the years 2014 and 2016. Since some former students of the course were in Erasmus, when possible, the invitations were sent to both institutional and personal email addresses. Moreover, with the aim of motivating the participation, the draw of a Sonos wireless speaker among the people who completed the survey was announced. Recipients were free to participate if they chose and their ratings and opinions would be anonymous. Due to the draw of the wireless speaker, the survey had an explicit closing date (April 23, 2017).

The first invitation was sent on February 8, 2017, and two reminders were sent before the closing date. The number of potential recipients of the survey invitation was 150: 45 of them partially completed the survey (they were not taken into account in the survey results), while 40 completed the whole survey. Therefore, the estimated response rate was approximately 27% (40/150). The platform enabled participants to save partially finished surveys.

To make sure that there was not a substantial difference concerning the characteristics of the survey respondents and the non-respondents former students of the course (*response bias*), we decided to compare their final grades obtained at the end of the course. Table 3 presents the main statistics about the grades obtained by former students of the course that did not participate (Non-respondents) in the survey and those who did participate (Respondents)[5].

[5]The survey was anonymous, and we could not associate the responses to each student. However, we had the list of students who responded.

Table 2: Subsystems and tasks in the questionnaire

Section A: End-user subsystem	**# Sub-Tasks**
Develop a native end-user mobile application	7
Develop a hybrid end-user mobile application	8
Develop a web responsive end-user mobile application	6
Develop the integration between the end-user application and the gateways [computation node, smartphone]	2
Deploy the end-user mobile application into the smartphone	1
Section B: Gateways subsystem (Development)	
Configure the development environment	2
Develop the business logic of the gateway device [computation node, smartphone] application	2
Configure the OAuth authentication between the gateway device [computation node, smartphone] and third-party services APIs	3
Develop the module for generating notifications to be displayed on the end-user application	2
Deploy the software into the gateway devices [computation node, smartphone]	1
Section C: Gateways subsystem (Integration)	
Develop the integration between the gateway device [computation node, smartphone] and the sensors [wearable devices, static sensors]	4
Develop the integration between the gateway device [computation node, smartphone] and the back-end [third-party service API, application server, persistence] by consuming these last ones' custom APIs	4
Develop the integration between the gateway device and the actuators responsible for changes in environment	2
Section D: Back-end subsystem (Development)	
Configure the development environment	2
Design and develop the persistence component	3
Develop the business logic on the application server	2
Develop the RESTful web services	3
Section E: Back-end subsystem (Integration)	
Develop the integration between the application server and third-party services	2
Configure OAuth between the application server and third-party services	3
Develop the integration between the application server and the persistence component	3

As it may be observed, the grades do not differ greatly between the two groups.

Another possible bias factor would be an item bias: some respondents might have rushed through several sections of the questionnaire intentionally to quickly

Table 3: Comparison between the grades of the former students and the respondents

	Non-respondents	**Respondents**
minimum	18.0	19.0
maximum	31.0	31.0
mean	26.3	28.3
SD	4.1	3.1
n	110	40

complete the survey and participate in the draw. However, as shown in Table 4, the time spent completing the online survey was generally consistent with the number of subsystems they worked on. On average, respondents who worked on one subsystem took 16 minutes, two subsystems 21 minutes, and three subsystems 25 minutes. Moreover, these data show that 16% of the respondents were involved in the development all the subsystems.

Table 4: Time spent completing the online survey by number of subsystems on which they answered questions

	1 Subsystem	**2 Subsystems**	**3 Subsystems**
minimum	0:04:53	0:08:33	0:19:29
maximum	0:42:26	0:44:44	0:46:31
mean	0:16:37	0:21:27	0:25:56
respondents percentage	50.0%	34.0%	16.0%

As the respondents were asked to answer the survey just for the sub-tasks that they completed in the development of their IoT systems, Table 5 shows the percentage of completion for each subsystem. The subsystem with a higher average percentage of completeness was the End-user subsystem (79%), followed by the Back-end (75%), and finally the Gateways (60%).

Table 5: Percentage of sub-tasks rated on each subsystem

	End-user	**Gateways**	**Back end**
minimum	44.4%	15.0%	37.5%
maximum	100%	100%	100.0%
mean	78.9%	59.7%	75.2%
respondents percentage	63.2%	50.0%	50.0%

4. Results

This section presents the results from the survey according to the proposed research questions. These results are described as detailed as possible in terms of particular software development tasks. Concretely, Section 4.1 provides a brief description of the demographics of the respondents of the survey, while Section 4.2 introduces the outcomes that emerged from the survey and their connection to the research questions. Section 4.3 concerns the first research question, Section 4.4 regards the second research question, and Section 4.5 addresses the third research question.

4.1. Demographics

At the beginning of the survey questionnaire, several questions were included to characterize the demographics of the respondents. As shown in Table 6, a vast majority of the respondents were male (87.5%), and their ages were mainly from 22 to 24 years old (80%), which is consistent with the student population. Furthermore, a marked majority of respondents belonged to Computer Engineering (70%), followed by those that belonged to Electronic Engineering (20% of them), as illustrated in Figure 3.

At the beginning of the course, students were asked to complete an online questionnaire concerning their prior knowledge on a set of technical skills and programming languages. Specifically, they graded such skills and programming languages on a 5-point Likert scale. This grading is reported in Table 7, where the ratings in the Likert scale are categorized into low (1 and 2), medium (3),

and high (4 and 5). It is observed that the prior knowledge declared by the students of the Ambient Intelligence course was low in most of the topics and programming languages. The only exceptions were programming (in general) and the C language, since nearly all students had taken a basic programming course. Similarly, as briefly mentioned in the Introduction, we consider that the respondents of our survey are, to some extent, representative of novice professionals in the context of IoT systems development. We support this idea based on the observation made by Salman et al. [9], according to which "in an academic setting, we can find students who already possess industrial experience or in a field experiment, we can face novice professionals with regard to a particular technology".

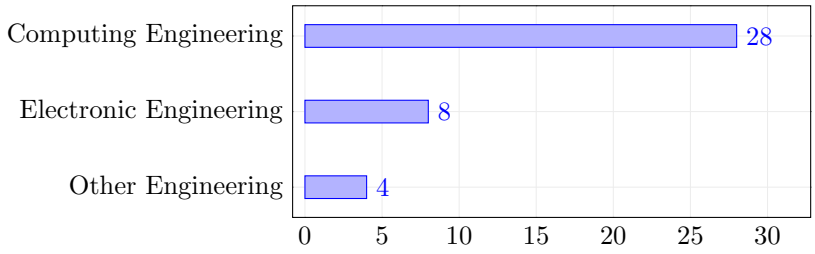
Table 6: Age and gender of the respondents

	22-24	**25-27**	**>28**	**Total**
Female	5	1	-	5
Male	27	4	3	33
Total	32	5	3	**40**

Table 7: Technical skills of the AmI students before taking the course

Topic	**Low**	**Average**	**High**
Programming (in general)	16.97%	**44.85%**	38.18%
Web architectures	**69.09%**	19.39%	11.52%
Mobile development	**86.06%**	9.09%	4.85%
Source control management	**90.91%**	4.85%	4.24%
Software requirements specification	**75.15%**	15.76%	9.09%
Python	**92.12%**	2.42%	5.45%
HTML/CSS	**81.21%**	9.09%	9.70%
JavaScript	**89.09%**	5.45%	5.45%
Java	**79.39%**	9.09%	11.52%
C	12.12%	27.27%	**60.61%**

Figure 3: Bachelor degree of the survey respondents when attending the course



4.2. Research questions

Three types of outcomes emerged for each of the subsystems in the survey:

1. The **rating** of each sub-task complexity in terms of its difficulty level and completion time (i.e., to answer RQ1).
2. The set of sub-tasks that were **ranked** as the most challenging (i.e., to answer RQ2).
3. The **perception** of each respondent about the most challenging tasks (i.e., to answer RQ3).

In this section, the three questions will be addressed for each subsystem. Since the **rating** of the difficulty level and time spent on each sub-task was captured on an ordinal scale, the results from this outcome were analyzed through the median and the correlation between these variables. As shown in Tables 8 to 10, almost all the development sub-tasks yielded positive correlations between the difficulty level and the time spent. Next, for analyzing the **ranking** of the most complex sub-tasks, they were prioritized according to the number of times they were included in the ranking and their position. To do this prioritization, a weighted sum was calculated for each sub-task and the three sub-tasks with the highest weights are presented below. Lastly, the **perception** about the most complex tasks was analyzed through the comments of the survey respondents, and three main categories were identified: *learning curve* issues, *integration between subsystems* issues, and *configuration and deployment* issues. At the end of this section, we will report, for each of these categories, some of the most rep-

resentative comments made by the respondents when they were asked to justify their ranking choice.

4.3. RQ1. Rating of the sub-tasks

Tables 8 to 10 present the development sub-tasks of each subsystem, along with two box plot diagrams illustrating the ratings for the difficulty level and time spent. Moreover, next to these diagrams we report the correlation between the ratings. When the p-value was less than 0.05, the correlation is flagged with a *, meaning that the correlation of the concerned sub-task is statistically significant.

4.3.1. Section A: End-user subsystem

End-user native mobile application (Section A1) The sub-task of *Becoming familiar with the mobile application platform-specific programming language (eu-nat-1)*, was rated as difficult and time-spending. Concretely, these programming languages are: Java, when developing Android applications, or Swift, when developing iOS applications. On the contrary, although also statistically significant, the sub-tasks of *Configuring the development environment (eu-nat-2)* and *Packaging the application into a compatible format that might be deployed on the smartphone (eu-nat-10)* were rated as not difficult and not time-spending, plausibly because the "official" IDEs provide the developers enough commands and visual interfaces to do so intuitively. *Developing the user interface (eu-nat-5)* yields a negative correlation, suggesting that it was considered to be more difficult than time spending. Probably because once the first views are developed, the learning curve is overtaken and the process becomes quicker. However, the magnitude (-0.19) and statistical significance of this correlation are weak. *Developing the controller's classes (eu-nat4)* was rated as a very difficult sub-task, and significantly time spending, with a high correlation (0.88) even if not statistically significant, according to the p-value. Effectively, the development of the controllers is critical since they are the bound between the models and the views, and they manage the interaction with external data

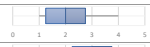
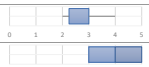
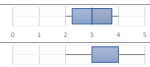
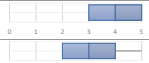
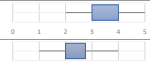
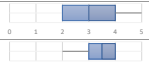
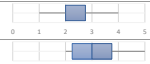
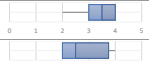
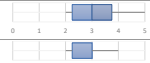
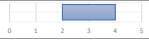
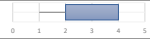
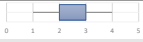
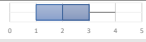
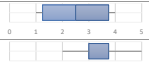
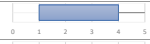
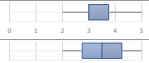
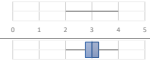
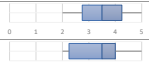
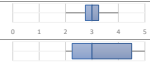
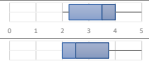
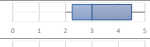
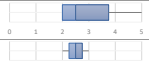
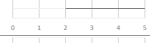
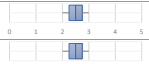
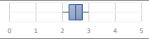
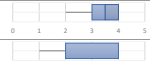
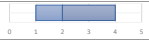
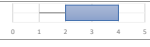
Section A1: End-user native mobile application		Difficulty	Time spent	Corr.
Develop a native end-user mobile application				
eu-nat-1	Become familiar with the mobile application platform-specific programming language			0.91*
eu-nat-2	Configure the development environment			0.93*
eu-nat-3	Develop the models' classes			0.59
eu-nat-4	Develop the controllers' classes			0.88
eu-nat-5	Develop the user interface (views)			-0.19
eu-nat-6	Connect the push notification module with the platform notification service			0.77
eu-nat-7	Handle the notifications received in the end-user's smartphone			0.77
Develop the integration between the end-user application and the gateways				
eu-nat-8	Implement the HTTP asynchronous requests through the RESTful web services exposed by the gateways			0.82
eu-nat-9	Parse and handle the JSON- or XML-formatted response			0.41
Deploy the end-user mobile application into the smartphone				
eu-nat-10	Package the application into a compatible format that might be deployed on the smartphone			0.94*
Section A2: End-user hybrid mobile application		**Difficulty**	**Time spent**	**Corr.**
Develop a hybrid end-user mobile application				
eu-hyb-1	Become familiar with the scripting programming languages			-0.08
eu-hyb-2	Configure the development environment			0.92*
eu-hyb-3	Incorporate into the project all the required plugins on their corresponding versions			0.61
eu-hyb-4	Develop the controllers			0.32
eu-hyb-5	Develop the user interface through HTML and CSS files (views)			0.40
eu-hyb-6	Develop the user interaction through JavaScript files (views)			0.80
eu-hyb-7	Connect the push notification module with the platform notification service			-
eu-hyb-8	Handle the notifications received in the end-user's smartphone			-
Develop the integration between the end-user application and the gateways				
eu-hyb-8	Implement the HTTP asynchronous requests through the RESTful web services exposed by the gateways			0.43
eu-hyb-9	Parse and handle the JSON- or XML-formatted response			0.82
Deploy the end-user mobile application into the smartphone				
eu-hyb-10	Package the application into a compatible format that might be deployed on the smartphone			0.94
Section A3: End-user web responsive mobile application		**Difficulty**	**Time spent**	**Corr.**
Develop a web responsive end-user mobile application				
eu-web-1	Become familiar with the scripting programming languages			0.69*
eu-web-2	Configure the development environment			0.87*
eu-web-3	Develop the controllers			0.70*
eu-web-4	Develop the user interface through HTML and CSS files (views)			0.18
eu-web-5	Develop the user interaction through JavaScript files (views)			0.47
eu-web-6	Deploy the application on the web			1.00
Develop the integration between the end-user application and the gateways				
eu-web-7	Implement the HTTP asynchronous requests through the RESTful web services exposed by the gateways			0.88*
eu-web-8	Parse and handle the JSON- or XML-formatted response			0.75*
Deploy the end-user mobile application into the smartphone				
eu-web-9	Package the application into a compatible format that might be deployed on the smartphone			0.96*

Table 8: End-user ratings

sources. *Connecting the push notification module with the platform notification service (eu-nat-6)* was also rated as a complex sub-task. This can be due to the parametrization and the functions that have to be properly implemented to generate and manage the notifications.

End-user hybrid mobile application (Section A2) *Configuring the development environment (eu-hyb-2)* was the only sub-task whose correlation between difficulty and time spent was statistically significant. Notwithstanding, the ratings of this task were scattered in both variables, most of them ranging from 1 to 4. Therefore, based on these numbers, it is not possible to determine accurately if this sub-task was perceived as complex or not. Unlike the development of the End-user native application, *Becoming familiar with the scripting programming languages (eu-hyb-1)* is not perceived as complex, and has a very weak, and negative, correlation between difficulty and time spent (-0.08). These results would suggest that an advantage for novice programmers of hybrid mobile applications over the native ones, is the use of scripting languages, which are more common and widespread. However, *Developing the user interface through HTML and CSS files (eu-hyb-5)* appears to be more difficult and time spending in hybrid mobile applications than in the native applications. Since the native applications IDEs are targeted at a specific mobile operating system, they provide a better consistency between the design, development, and deployment of the user interface. Finally, in the sub-task of *Connecting the push notification module with the platform service (eu-hyb-7)*, all the respondents rated the time spent with 2, as well as in *Handle the notifications received in the end-user's smartphone (eu-hyb-8)*, where all the respondents rated the difficulty as 3. In both cases the low number of responses did not allow the correlation to be computed.

End-user responsive mobile application (Section A3) The set of sub-tasks with a significant correlation were rated as not particularly complex. *Parsing and handling the JSON- or XML-formatted responses (eu-web-8)* had exactly the same ratings in the difficulty and time spent variables as well as *Packaging the application into a compatible format that might be deployed on*

the smartphone (eu-web-9). The difficulty when *Developing the controllers (eu-web-3)* had the same value (3) for the first quartile, the median, and the third quartile. *Implement the HTTP asynchronous requests through the RESTful web services exposed by the gateways (eu-web-7)* was rated as considerably difficult and time spending (median = 3, in both variables). This sub-task might be more complex in this kind of mobile applications due to the lack of libraries or frameworks to manage the connection and the HTTP requests and responses, such as the ones in the native or hybrid mobile applications IDEs. Naturally, the *Deployment of the application on the web (eu-web-6)* was rated as easy and quick. The *Development of the user interface through HTML and CSS files (eu-web-4)* was rated as easy but considerably time spending. Probably all the programmers were familiar with HTML and CSS, or at least could easily become skilled in them. However, the absence of a tool to graphically compose and link the views could affect negatively the time spent.

4.3.2. Sections B and C: Gateways subsystem

Gateways development (Section B) in almost all the sub-tasks the correlation between difficulty and time spent was statistically significant. *Configuring the development environment (gw-dev-1* and *gw-dev-2)* was rated as easy and quick. The ratings for *Developing the business logic of the gateway device application (gw-dev-3* and *gw-dev-4)* were mainly scattered from 2 to 4, meaning that while not extremely difficult, neither they were extremely easy. In the context of the IoT course projects, the concept of 'business logic' basically refers to the way in which the data gathered from the sensors will be used to accomplish the overall system functional requirements. *Setting up the parameters needed to establish the connection with third-party services APIs using OAuth (gw-dev-5)* was rated as very time spending while moderately difficult. In fact, even if this sub-task doesn't require a significant programming effort, it requires a good conceptual and technical understanding of OAuth and the registration of the application in the third-party service platform. Likewise, *Developing the methods or functions required to establish the connection with third-party services APIs*

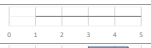
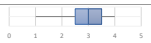
Section B: Gateways development		Difficulty	Time spent	Corr.
Configure the development environment				
gw-dev-1	Install and deploy the operating system			0.64*
gw-dev-2	Install the libraries and dependencies needed to develop on the gateway's controller			0.74*
Develop the business logic of the gateway device application				
gw-dev-3	Define and implement the required set of models			0.82*
gw-dev-4	Develop the methods or functions where the business logic is implemented			0.80*
Configure the OAuth authentication between the gateway device and third-party services APIs				
gw-dev-5	Set up the parameters needed to establish the connection			0.43
gw-dev-6	Install the required set of libraries			0.88*
gw-dev-7	Develop the methods or functions required to establish the connection			0.82*
Develop the module for generating notifications to be displayed on the end-user application				
gw-dev-8	Set up the parameters needed to establish the connection with the platform notification service			0.52
gw-dev-9	Generate the notifications by invoking the platform notification service APIs			0.80*
Deploy the software into the gateway devices				
gw-dev-10	Package the application into a compatible format that might be deployed on the gateway			0.87*
Section C: Gateways integration		**Difficulty**	**Time spent**	**Corr.**
Develop the integration between the gateway device and the sensors				
gw-int-1	Develop the methods or functions required to establish the Wi-Fi connection with the sensors			0.85*
gw-int-2	Develop the methods or functions required to establish the Bluetooth connection with the sensors			0.90*
gw-int-3	Develop the methods or functions required to obtain data from the sensors			0.78*
gw-int-4	Develop a component for receiving real-time streaming data coming from the sensors			0.90*
Develop the integration between the gateway device and the back-end by consuming these last ones' custom APIs				
gw-int-5	Implement the HTTP asynchronous requests through the RESTful web services exposed by third-party services APIs			0.17
gw-int-6	Parse and handle the JSON- or XML-formatted response obtained from third-party services APIs			0.78*
gw-int-7	Implement the HTTP asynchronous requests through the RESTful web services exposed by the application server			0.28
gw-int-8	Parse and handle the JSON- or XML-formatted response obtained from the application server			0.08
Develop the integration between the gateway device and the actuators responsible for changes in environment				
gw-int-9	Develop the methods or functions required to establish the connection			0.88*
gw-int-10	Develop the methods or functions required to handle the actuators behaviour			0.54

Table 9: Gateways ratings

using OAuth (gw-dev-7), was rated as a difficult and time spending sub-task. It demands to consult several documentation sources and devote a significant amount of time ensuring a successful connection. *Generating the notifications by invoking the platform notification service APIs (gw-dev-9)* had exactly the same ratings in the difficulty and time spent variables.

Gateways integration (Section C) the correlation of all the sub-tasks regarding the *Integration between the gateway device and the sensors (gw-int-1 to gw-int-4)* were statistically significant. In particular, *Developing the methods or functions required to establish the Bluetooth connection with the sensors (gw-int-2)* was rated as easy and quick, and the *Development of the methods or functions to obtain data from the sensors (gw-int-3)* was rated as time-spending. The difficulty in the *Development of a component for receiving real-time streaming data coming from the sensors (gw-int-4)* had the first quartile, the median, and the third quartile rated as 3. The rest of the sub-tasks in this subsection did not exhibit a clear trend concerning their low or high complexity.

4.3.3. Sections D and E: Back-end subsystem

Back-end development (Section D) The sub-tasks corresponding to the *Configuration of the development environment (be-dev-1* and *be-dev-2)*, and the *Design and development of the persistence component (be-dev-3* and *be-dev-4)* were not rated as complex. On the contrary, the *Development of the business logic on the application server* was rated as difficult and time spending. Especially the *Development of the methods or functions where the business logic is implemented (be-dev-7).* Naturally, the complexity of this sub-task has to do with the fact that each group had to program its business logic methods from scratch, and without the guidance of previous implementations. The sub-tasks concerning the *Development of the RESTful web services (be-dev-8* to *be-dev-10)* were not rated as difficult but as moderately time spending.

Back-end integration (Section E) *Parsing and handling the JSON- or XML-formatted response (be-int-2)* was rated as easy and quick. When configuring the OAuth authentication between the application server and third-party services, the *Installation of the required set of libraries (be-int-4)* was rated as easy and quick. Moreover, the *set up of the parameters (be-int-3) and the development of the methods or functions to establish the connection (be-int-5)*, do not exhibit a clear trend about how complex they resulted to the respondents. Lastly, all the sub-tasks concerning the *Development of the integration between*

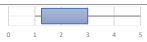
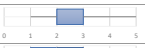
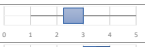
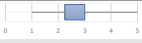
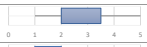
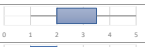
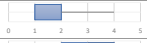
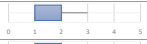
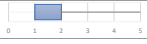
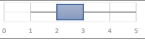
Section D: Back-end development		Difficulty	Time spent	Corr.
Configure the development environment				
be-dev-1	Install and deploy the application server			0.68*
be-dev-2	Install the libraries and dependencies needed for the application server development			0.89*
Design and develop the persistence component				
be-dev-3	Design the entity-relationship model or the corresponding data model, if NoSQL is used			0.80*
be-dev-4	Install the database server			0.92*
be-dev-5	Deploy the database with the corresponding data model			0.49
Develop the business logic on the application server				
be-dev-6	Define the required set of models			0.88*
be-dev-7	Develop the methods or functions where the business logic is implemented			0.86*
Develop the RESTful web services				
be-dev-8	Define the HTTP methods along with their URI and associated operation			0.75*
be-dev-9	Set up the framework required to implement the RESTful web services			0.66*
be-dev-10	Implement the mapping between the business logic models and the exposed RESTful web services			0.41
Section E: Back-end integration		**Difficulty**	**Time spent**	**Corr.**
Develop the integration between the application server and third-party services				
be-int-1	Implement the HTTP asynchronous requests through the RESTful web services exposed by third-party service APIs			0.81*
be-int-2	Parse and handle the JSON- or XML-formatted response			0.74*
Configure the OAuth authentication between the application server and third-party services				
be-int-3	Set up the parameters needed to establish the connection			0.99*
be-int-4	Install the required set of libraries			0.77*
be-int-5	Develop the methods or functions required to establish the connection			0.90*
Develop the integration between the application server and the persistence component				
be-int-6	Set up the connection between the application server and the database			0.80*
be-int-7	Implement the queries to be performed over the database			0.70*
be-int-8	Parse and handle the database response			0.82*

Table 10: Back-end ratings

the application server and the persistence component (be-int-6, be-int-7, and be-int-8) were rated as easy and not time spending.

4.3.4. Summary of RQ1

RQ1: How complex, in terms of time spent and difficulty, are the software development tasks needed to build an IoT system?

Almost all the sub-tasks hold a positive correlation between the two variables

that we used to measure the complexity. It means that all the sub-tasks rated as significantly difficult were also rated as considerably time spending. By comparing the ratings of the sub-tasks and the correlation between their two variables, we could preliminarily identify for each subsystem those sub-tasks that stand out as complex. Hereafter we present a summary of the main findings for each subsection of the questionnaire.

In the **End-user native mobile application**, becoming familiar with the platform-specific programming language (eu-nat-1), and developing the application controllers (eu-nat-4) were rated as difficult and time-spending sub-tasks. In the **End-user hybrid mobile application**, the rating of the sub-tasks and their statistical significance did not allow to identify a clear tendency about the correlation between the difficulty and the time spent. In the **End-user responsive mobile application**, the implementation of the HTTP asynchronous requests through the RESTful web services exposed by the gateways (eu-web-7) was rated as considerably difficult and time spending. On the contrary, the development of the user interface through HTML and CSS files (eu-web-4) was rated as easy but time spending. In the **Gateways development**, implementing the methods or functions to establish the connection with third-party services APIs using OAuth (gw-dev-7), was rated as difficult and time-spending. In the **Gateways integration**, the correlation of the sub-tasks concerning the integration between the gateways and the sensors (gw-int-1 to gw-int-4) were statistically significant. While the methods or functions to establish the connection with the sensors was rated as easy and quick (gw-int-2), the methods or functions to gather information from those sensors were rated as time-spending (gw-int-3). In the **Back-end development** the implementation of the business logic on the application server was rated as difficult and time spending (be-dev-7). On the contrary, the sub-tasks concerning the development of the RESTful web services were not rated as difficult but as moderately time spending (be-dev-8 to be-dev-10). In the **Back-end integration** subsystem no sub-task was

rated as particularly difficult or time spending.

4.4. RQ2. Ranking of the sub-tasks

Table 11 presents, for each subsystem, the three sub-tasks that the respondents placed in the first position of the ranking they were asked to do in the survey, as the most complex ones. In the End-user subsystem, these sub-tasks are listed depending on the kind of End-user mobile application developed whether it was a native, hybrid or web-responsive mobile application. Next to each subtask we show a triplet of numbers representing the number of times in which the concerned sub-task was ranked in the first, second, and third place, respectively. As expected, this ranking matches with the **rating** of the sub-tasks in terms of difficulty level and time spent.

According to the sub-tasks ranked as the most complex, it can be observed that despite the kind of End-user mobile application implemented (native, hybrid or web-responsive), the development of the user interface was perceived as complex (eu-nat-5, eu-hyb-5, eu-web-4). This observation is somehow surprising, particularly concerning native mobile applications, where we may presume that the IDE would ease the design of the user interface views and their connection with the business logic of the application. Moreover, the configuration of the development environment was perceived as complex both in the native mobile applications as well as in the web responsive mobile application (eu-nat-2, eu-web-2). Once again, it is striking that even with specialized IDEs as the ones used to develop native mobile applications, the configuration of the development environment was ranked as complex. Finally, concerning the End-user subsystem, the development of the controllers (which can be understood as the binding between the views and the business logic), was ranked as one of the most complex sub-task both in the hybrid mobile applications and in the web-responsive mobile applications.

In the Gateways development and the Gateways integration subsystems there is a clear correspondence between the *development of the methods or func-*

tions to establish the connection between the gateway device and the third party services APIs using OAuth, in the Gateways development (gw-dev-7), and the *parsing and handling the JSON or XML-formatted response obtained from the third-party services APIs*, in the Gateways integration (gw-int-6).

Similarly, in the Back-end subsystem, there is a clear mapping between the *deployment of the database with the corresponding data model*, in the Back-end development (be-dev-5), and the *parsing and handling of the database response in the application server*, in the Back-end integration (be-int-8).

However, between Gateways and Back-end subsystems, there are also some coincidences. For instance, the *development of the functions or methods where the business logic is implemented* was ranked in both subsystems as a complex sub-task (gw-dev-4 and be-dev-7). Furthermore, there is another correspondence between the *implementation of the HTTP asynchronous request through the RESTful web services exposed by the application server*, in the Gateways integration (gw-int-7), and the *implementation of the mapping between the business logic models and the exposed RESTful web services*, in the Back-end development (be-dev-10).

4.4.1. Summary of RQ2

RQ2: Which are the software development tasks that are perceived as the most complex to complete?

The development tasks that are perceived as the most complex regard aspects that are common to various subsystems, as might be the development of user interfaces, the configuration of the development environments, and the development of the business logic. Some other aspects, however, are split across various subsystems. Such is the case of the integration between the Gateway devices and the third-party services, the implementation and integration with the persistence component, and the implementation, exposure, and consumption of custom RESTful web services.

Due to the above, besides determining the most complex sub-tasks on each

Table 11: Sub-tasks ranked as the most complex

Section A1: End-user native mobile application		
eu-nat-1	Become familiar with the mobile application platform-specific programming language	3-3-0
eu-nat-2	Configure the development environment	1-1-1
eu-nat-5	Develop the user interface (views)	1-0-2
Section A2: End-user hybrid mobile application		
eu-hyb-7	Connect the push notification module with the platform notification service	2-0-0
eu-hyb-5	Develop the user interface through HTML and CSS files (views)	1-1-0
eu-hyb-4	Develop the controllers	1-0-0
Section A3: End-user web responsive mobile application		
eu-web-2	Configure the development environment	3-0-0
eu-web-3	Develop the controllers	2-1-1
eu-web-4	Develop the user interface through HTML and CSS files (views)	2-0-1
Section B: Gateways development		
gw-dev-4	Develop the methods or functions where the business logic is implemented	5-2-1
gw-dev-7	Develop the methods or functions to establish the connection (between the gateway device and third-party services APIs using OAuth)	5-0-1
gw-dev-5	Set up the OAuth parameters needed to establish the connection with the third-party services APIs	2-3-1
Section C: Gateways integration		
gw-int-3	Develop the methods or functions required to obtain data from the sensors	3-2-1
gw-int-6	Parse and handle the JSON- or XML-formatted response obtained from the third-party services APIs	3-1-1
gw-int-7	Implement the HTTP asynchronous requests through the RESTful web services exposed by the application server	2-3-1
Section D: Back-end development		
be-dev-5	Deploy the database with the corresponding data model	2-4-1
be-dev-7	Develop the methods or functions where the business logic of the application server is implemented	2-3-1
be-dev-10	Implement the mapping between the business logic models and the exposed RESTful web services	2-2-2
Section E: Back-end integration		
be-int-1	Implement the HTTP asynchronous requests through the RESTful web services exposed by third-party service APIs	9-0-0
be-int-8	Parse and handle the database response (in the application server)	2-3-3
be-int-3	From the server application, set up the OAuth parameters needed to establish the connection with the third-party services	4-0-3

subsystem, the ranking section of the survey helped to identify, firstly, the set of sub-tasks that are common to various subsystems, and secondly, a set of sub-tasks that complement each other in the development of some portions of the IoT system.

In the first set of sub-tasks there are:

- Sub-tasks concerning the development of the user interfaces were ranked as complex regardless the kind of End-User mobile application (eu-nat-5, eu-hyb-5, and eu-web-4).

- The configuration of the development environment was also ranked as complex both in the native mobile application and in the web responsive mobile application.

- The development of the controllers (binding between models and views in the MVC architecture) was ranked as complex in the hybrid mobile applications and in the web responsive mobile applications.

In the second category there are:

- The development of the methods or functions to establish the connection between the Gateway device and the third party service APIs using OAuth (gw-dev-7), and the parsing and handling of the response obtained from the third party service APIs, in the Gateways Integration subsystem (gw-int-6).

- The design, implementation, and integration of the persistence component, which spans across the Back-end development (be-dev-5) and the Back-end integration (be-int-8).

- The design, implementation and later consumption of the custom RESTful web services. Their implementation and mapping between the business logic and the exposed RESTful web services belong to the Back-end De-

velopment subsystem (be-dev-10), while the consumption of those services belongs to the Gateways Integration (gw-int-7).

4.5. RQ3. Qualitative perception of the survey respondents

We now present and analyze the comments given by the respondents when asked about why did they perceive certain sub-tasks as the most challenging ones. Such comments were analyzed through inductive thematic analysis, that involved two researchers and followed the six-phase framework proposed by Braun and Clarke's [30]. The first step consisted in becoming familiar with the comments of the respondents and was carried out by the two researchers. Secondly, the first researcher categorized the material at the sentence level and generated the initial codes. The second researcher, for his part, discussed and validated them. This discussion was in person and using hard copies of the respondents' comments. Since an open code approach was used, there were not preset codes. Instead, the codes were being developed and modified while advancing through the coding process. Thirdly, the first researcher identified produced a set of themes with their corresponding codes; initially, 18 open codes were used, later grouped into three broader themes. Further on, in the fourth step, the second researcher validated and approved the proposed themes under the rationale that they were supported by the previously generated codes. In this manner, in the fifth step, the themes were jointly analyzed, and in the last step, the researchers' observations around these themes were documented. The thematic analysis revealed three key themes, based also on the commonalities of the comments across all the subsystems: *Learning curve issues*, *Integration between subsystems issues*, and *Configuration and deployment issues*.

4.5.1. Learning curve issues

In the **End-user subsystem**, comments about the Learning curve issues, highlight that most of the survey respondents were developing for the first time a mobile application. Those groups that decided to implement a native End-

user mobile application, faced the challenge of learning a new programming language and becoming familiar with the concerned development environment. Respondents expressed: *"I knew almost nothing on Android when I started developing the app, I had to learn everything from scratch"*, *"Becoming familiar with a new programming language in a very little time is very difficult"*. *"I spent a lot of time researching how to complete everything and make it work"*.

In the **Gateways subsystem development**, Learning curve issues concerned how to handle the data gathered from the End-user mobile application, as well as from the sensing devices: *"We worked with GPS, so we had to do some research about playing with coordinates and GPS accuracy."* Moreover, since the communication with End-user and Back-end subsystems is typically achieved through RESTful web services, the understanding of these services (both the ones that had to be implemented as well as the external ones that had to be consumed), was perceived as challenging by some respondents: *"I had never heard about APIs, and there were tasks in which it was required to work with them."*

Similarly, the **Gateways subsystem integration** required the knowledge about how to adequately implement the *web services* so that, through HTTP requests coming from the other subsystems, the gateway is directed to perform some given business logic function. *"I was a very beginner with no background in 'Implement the HTTP asynchronous requests through the RESTful web services exposed by the third-party services APIs' and 'Parse and handle the JSON- or XML-formatted response obtained from the third-party services APIs'."* Likewise, managing the integration through other transmission protocols was perceived as challenging due to the lack of adequate documentation: *"Bluetooth documentation for Android wasn't clear, and there were not enough examples of how to use it"*, *"There was no documentation for some smart home sensors, or sometimes we found very poorly written documentation"*.

Once again, as occurred in the Gateways subsystem, in the **Back-end subsystem development**, the RESTful web services concept was also challenging to apprehend. *"It took me some time mostly because of the scarce knowledge of*

Flask, but as I figured things out it all got easier ('Set up the framework required to implement the RESTful web services' and 'Develop the methods or functions where the business logic is implemented')." In the same way respondents expressed: *"I had to study a lot of things to understand how to implement my logical function in HTTP"*, *"It took some research to understand how to implement it ('Define the HTTP methods along with their URI and associated operation')."* Finally, among the **Back-end subsystem integration**, the learning curve issues concerned how to deal with the data exchanged with the back-end: *"Needed to learn how to 'Parse and handle the database response' properly."*

4.5.2. Integration between subsystems issues

As noted before, across the three types of mobile applications, the integration between the **End-user subsystem** and the Gateways subsystem was among the most challenging issues. Respondents commented *"Interaction between the server and the mobile app was quite difficult for us because we decided to manage it in a 'custom' way."* The integration between the End-user subsystem and the third-party services was particularly challenging: *"We had several problems interfacing the Fitbit APIs with our application"*.

The most painful integration among the **Gateways subsystem development**, according to the respondent's comments, regards the OAuth 2.0 authentication. In fact, this authentication protocol consists of a flow, with a set of roles (*resource owner, resource server, client, and authorization server*) interacting across various steps (*authorization request, access token request, and protected resource request*), and exchanging several resources (*authorization grant, access token, refresh token, redirect URI*). Respondents commented *"OAuth 2.0 authentication, to gain access to the Fitbit API, was a mess. There were hardly any tutorials for the method, and it took a lot of time to figure it out"* and *"Authentication is a nightmare."* Moreover, concerning the integration that did not required OAuth, the main reason given by the respondents was the lack of experience: *"I had no experience on how to connect the devices."*

When working on the **Gateways subsystem integration**, dealing with

the data coming from the back-end resulted challenging, particularly when receiving streaming data coming in real-time: *"While receiving generic data was easily done, when we stepped up to the real-time stream we spent an enormous amount of time just to figure out how to access it and then how to handle it." "Establishing the connection between front and Back-end is not an easy task! I'd never worked with JSON."*

The integration issues in the **Back-end subsystem development** regarded the communication with the Persistence component, the Gateways, and even the Sensing devices, for those groups that decided to communicate the Sensors subsystem directly with the Back-end subsystem. Finding out how to handle incoming data into the business logic that is implemented in the Application server is perceived as the most challenging issue. Concerning the managing of Persistence component, respondents commented: *"It took a lot of time to develop an error-free function in Python to manage the SQLite database."* When dealing with the data gathered directly from the sensors, respondents commented: *"It was hard to make things run together (i.e. retrieve information from the sensors without stopping the web-application)."* Finally, when incorporating the incoming data into the business logic of the Application server, respondents commented: *"It was necessary to implement a set of models that were compatible with hardware (Arduino) and the web and application server (Flask). The link between these two architectures was not so easy to design."*

According to the respondents comments, the most challenging issues in the **Back-end subsystem integration** had to do with integrating third-party APIs: *"I spent a lot of time because it was the first time I dealt with it ('Implement the HTTP asynchronous requests through the RESTful web services exposed by the third-party service APIs'). I had to change my implementation several times due to the limitations of the commercial third-party APIs. Likewise, it took a while to understand how to use them to achieve our goals. In the final implementation, we used three external APIs'."* As already pointed out in the Gateways subsystem issues, third-party APIs have their own specific protocols, formats, and authentication mechanisms. These specifics imply a higher

level of complexity for the novices when integrating third-party services in their projects.

4.5.3. Configuration and deployment issues

Configuration and deployment issues concerning the **End-user subsystem** were mainly related to the development environment and the dependencies required to develop the End-user mobile applications. *"The development environment used throughout the course (Eclipse) is quite simple to use but requires a lot of effort to configure it for a given development project."* Depending whether the mobile application was native, hybrid, or web-responsive, diverse development environments and their dependencies had to be configured: *"I found very hard to configure PhoneGap and Apache Cordova in Windows."* As part of the **Gateways subsystem development**, various libraries have to be installed in order to ease and manage the communication with the End-user and the Back-end subsystem: *"The libraries were always difficult to install, and also very time consuming"* and *"In my personal experience, the OAuth authentication was very difficult to set up for its first use."*

In the **Back-end subsystem development** the sub-tasks aimed at designing, setting up and deploying the Persistence module, were perceived as time-consuming: *"While not difficult per se, these were the tasks that took most of my time. For example, deploying the database, designing the Entity-relationship model, and setting up the hardware."* Moreover, fixing technical details that may affect the development of the Back-end had some minor impact in the development of the Back-end: *"It takes a while to deploy the system because of some port conflicts derived from some issues in the configuration of the development environment."* Once implementing the **Back-end subsystem integration**, and specifically its integration with third-party services, OAuth requires configuring the Back-end from which the authenticated request will be made. Commonly it involves installing libraries, setting up various parameters, and configuring the third-party APIs in their corresponding web platforms. *"I had a lot of problems, and spent a lot of time, properly configuring the OAuth authentication between*

our application server and the one of Jawbone, it needed a lot of permissions."

4.5.4. Summary of RQ3

RQ3: Why are these tasks perceived as the most complex?

By analyzing the comments of the respondents concerning their reasons behind the ranking of the sub-tasks, there might be identified, on the one hand, the lack of adequate documentation that might be understandable by the novices. And on the other hand, the lack of knowledge and expertise required to deal with several protocols, formats, authentication mechanisms, and real-time data. Concretely, their feedback could be categorized into: *Learning curve issues*, *Integration between subsystems issues*, or *Configuration and deployment issues.* Hereinafter we provide a brief summary of the most common perceptions on each category

Learning curve issues: In the **End-user** subsystem, these kinds of issues concerned the fact that most of the survey respondents were developing for the first time a mobile application. They faced the challenge of learning a new programming language and becoming familiar with the concerned IDE. When developing the **Gateways**, respondents struggled with understanding conceptually the RESTful web services before dealing with their implementation and consumption later (when implementing the **Gateways** integration subsystem). Furthermore, also concerning **Gateways** integration, achieving the integration with the **End-user** subsystem using other transmission protocols such as Bluetooth was perceived challenging due to the lack of adequate documentation. In the **Back-end** subsystem, the RESTful web service concept was mentioned once again as difficult to apprehend, particularly regarding the mapping between business logic methods and the web service endpoints.

Integration between subsystems issues: Without a doubt, the issues regarding the integration between subsystems were the most common ac-

cording to the respondent's comments. The integration with both, self-implemented software components as well as with third-party services APIs, was perceived as complex. In the End-user subsystem, the integration with third-party services was particularly challenging. In the Gateways subsystem development, dealing with the OAuth authentication was perceived as complex given the flow, roles, and steps of this authentication mechanism. According to respondent's comments, the inherent complexity of OAuth was worsened by the lack of documentation understandable by novices. In the Gateways integration, dealing with streaming data coming in real-time from the Back-end was perceived both as extremely difficult and time-spending. In the Back-end, the main issues have to do with appropriately handling the data coming from the other subsystems into the application server business logic. As pointed out in the Gateways subsystem issues, dealing with the particular protocols, formats, and authentication mechanisms of each third-party service APIs was perceived as very challenging when implementing the Back-end integration.

Configuration and deployment issues: In the **End-user** subsystem, these kinds of issues concerned the proper configuration of the development environment and the dependencies required to implement the mobile applications. When developing the **Gateways** subsystem, the installation, and configuration of several libraries to manage the communication with the other subsystems were perceived as difficult and time spending, especially when the communication involved OAuth. In the **Back-end** development, the design, setting up, and deployment of the persistence component (typically a relational database) was perceived as time-consuming. Also, the deployment of the Back-end itself (application server) was not trivial. Same as the Gateways integration, in the **Back-end** integration, dealing with third-party services requires several configuration and parametrizations so that authenticated request could be sent to the third-party services APIs.

Finally, from a system-level view, the sub-tasks that were perceived as the most challenging, are concerned with the integration between the subsystems. Whether classified as *Learning curve issues*, *Integration between subsystems issues*, or *Configuration and deployment issues*, many of the comments stressed the complexity inherent to the integration of heterogeneous subsystems. On the other hand, the Learning curve issues may be explained basically with the lack of knowledge of the respondents, or with the lack of adequate documentation that might be understandable for them.

5. Discussion

The *rating* of the sub-tasks provided a first perspective about how difficult did IoT novice developers find the implementation of concrete development sub-tasks. These sub-tasks were presented as detailed as possible without linking them to a specific programming language, framework or run-time environment. Differentiating difficulty level from time spent helped to understand which sub-tasks are complex just from the practical point of view (such as the time spending sub-tasks) and which other sub-tasks are also complex from the conceptual and learning curve perspective (such as the difficult and also time spending sub-tasks).

Later, aiming at clearly identifying the most complex sub-tasks, it was necessary to ask respondents to prioritize the three most complex sub-tasks they faced on each subsystem. In this way, from the *ranking* of the sub-tasks, we got a perspective of the most painful development issues. From this perspective, there were identified complex sub-tasks common to many subsystems, and complex sub-tasks that complement between them across various subsystems.

Thirdly, to understand the previous ranking, it was essential to capture qualitatively the *perceptions* of the respondents about the choice they made and their reasons behind that selection. These impressions led to the identification and categorization of the sources of complexity present in the implementation of particular software development tasks in the context of an IoT system. These

categories were: *Learning curve* issues, *Integration between subsystems* issues, and *Configuration and deployment* issues.

By combining *rating*, the *ranking* and the *perceptions*, some specific programming areas were recurrently mentioned as particularly hard and painful. In particular, the integration of different subsystems, that require over-the-network communication protocols, and their debugging resulted considerably difficult, due to the diversity of client and server environments and the difficulty of tracing the remote calls. This was worsened by the fact that some third-party services are proprietary, give little visibility over their behavior, and each of them requires to follow different approaches and programming patterns. The integration with third-party services is particularly painful, whether it is the push notification service or the APIs that require OAuth authentication.

From the ranking of the sub-tasks, the development of the user interfaces in the End-user subsystem was among the most complex sub-tasks regardless of the kind of mobile application implemented (native, hybrid or web-responsive). This fact is somehow surprising if considering that the development of native mobile applications relies on specialized IDEs that in theory would ease the implementation of the views. These IDEs typically provide drag and drop features through which placing the user interaction components on the views should be easy enough. Notwithstanding, based on the comments of the respondents in the last section of the survey, we think that the difficulties experienced by the novice IoT developers concern the binding between the business logic and the events generated at the user interface of the mobile application. In fact, the development of the controllers was also ranked as a complex sub-task in the implementation of hybrid and web responsive mobile applications. Relatively in line with the complexity experienced in the views and controllers development, the configuration of the development environment was also ranked as complex in native and hybrid mobile applications implementation. Once again, it is unexpected since the IDEs of such kind of applications usually have features to deal with external dependencies, as well as to debug and simulate the application execution.

Furthermore, also concerning the ranking of the sub-tasks, we observed that sub-tasks aimed at integrating were among the most complex in several subsystems. Moreover, we identified a complementarity between these sub-tasks even if belonging to different subsystems. This complementarity involves:

- The design, implementation, and consumption of custom RESTful web services, that span across Back-end Development and Gateways Integration subsystem.
- The development of methods to achieve the connection between the Gateway devices and the third party services APIs, that spans across Gateways Development and Gateways Integration subsystems.
- The design, implementation, and integration of the persistence component, that spans across the Back-end development and the Back-end integration.

This observation emerged from the ranking and confirmed by the comments of the respondents, reinforce our perception of the significant complexity around the integration of heterogeneous software components. Both when consuming external services as well as when implementing custom integrations based on each project business logic. As outlined in the literature, dealing with the required level of interoperability is considerably painful to novices when developing IoT systems.

Furthermore, the lack of proper documentation and examples about how to integrate the subsystems is one of the main reasons why integration tasks are perceived as challenging. The difficulty to find well-structured documentation that might be understood by a novice was repeatedly mentioned by the respondents of the survey. For this reason, in all the subsystems the sub-tasks regarding integration were ranked among the first three.

The results from the survey point to the need to support novice IoT developers in dealing with several protocols, formats, authentication mechanisms, and streaming data coming in real-time. However, this support must be ad-

dressed both from the conceptual and technical perspectives. On the one hand, providing documentation that may be easily comprehended by non-expert IoT developers, and on the other, tools that would ease the configuration and development of certain software components integration.

From our experience with the university course that the survey respondents attended, we observed that the implementation of the integration between software components is similar across different projects developed by the respondents (especially when third-party services are involved). For this reason, we envisioned that, if documented, the code developed by the novices might provide some guidance to other programmers that are in the process of overcoming the same learning issues. In fact, being able to observe how someone else coded, what others paid attention to, and how they solved problems all support learning better ways to code and access to superior knowledge [31]. In this regard, we worked in what we named *Code Recipes* [32], i.e., summarized and well-defined documentation modules, independent from programming languages or run-time environments. They are specified through a set of metadata and consist of multiple code fragments along with documentation and links to ease the understanding of such code, in order to implement a given integration between subsystems of an IoT system.

Another possible approach might consider the automation of the sub-tasks that were rated as the most time spending. Both code generation and its deployment across several devices could be automated in such a way that novices just have to specify some custom parameters. Software Product Lines [33][34] might be a viable solution in this direction as long as it deals with the heterogeneity of IoT devices, protocols, and programming languages. Finally, it would be worth focusing on the lack of tools to debug the communication between devices and software components across the subsystems, giving some insight to the novices about possible failures in the data exchange process.

In this work we tried to propose a reference architecture as generic as possible, taking distance from any technological stack. For this reason, we consider that the survey structured around this architecture could be useful to a wide

range of IoT developers. Nevertheless, it must be said that since that our research questions concern the software development perspective, our findings do not take into consideration the physical configuration and deployment of IoT systems. Future work could address the identification of the most complex issues concerning the Security implementation from the software perspective.

5.1. Implications

The survey presented in this work aimed at gaining an understanding of the challenges that novice programmers face when developing an IoT system. The first research question enabled us to weight all development tasks, identifying their difficulty level and the time that novices spent completing them. In the second research question, we could determine that some of the development tasks perceived as complex concern aspects that are common across various subsystems. Examples are: the development of user interfaces, the configuration of development environments, and the development of the business logic. On the other hand, there are development tasks that concern aspects that are split across various subsystems. Examples are: the methods or functions to integrate the Gateway with the third-party service APIs, the implementation and integration of the persistence component, and the design, implementation, and consumption of the RESTful web services. The third research question provided insights about the causes behind the challenging issues faced by the novices. We identified learning curve issues, integration between subsystems issues, and configuration and deployment issues. The most frequently reported issues concerned: the difficulty to find well-structured documentation that might be understood by a novice, the complexity inherent to the integration of the subsystems, and the integration with third-party services.

We consider that identifying these challenges might have an impact both in academic and industrial contexts. Our findings could be used to ease the learning curve in the teaching scenario, and to make the IoT systems development more efficient in the software industry by improving on-boarding time estimations, hiring criteria, and human resource management within the projects. We

recommend that special attention should be given to the integration of different subsystems, taking into account the various protocols, formats, authentication mechanisms (specially OAuth), and real-time data streaming. Among these integrations, the integration with third-party services resulted particularly painful. Therefore we suggest that research efforts should envision automation or debugging tools, as well as improved documentation strategies. Finally, more empirical studies are required to validate our findings with a diverse set of practitioners.

6. Validity of results

This section examines the threats to the validity according to the classification schema proposed by Cook and Campbell [35]. In that schema, four types of threats to validity are defined: conclusion validity, internal validity, construct validity, and external validity. In the following sections we present a detailed description of each type of threats to validity, as well as some considerations about the repeatability of our work (Section 6.5).

6.1. Internal validity

Threats to internal validity regard issues that may indicate a causal relationship, although there is none [36]. In this survey, threats to internal validity concerned the *instrumentation*, the *subjects selection*, and the *maturation*.

With regard to the *instrumentation*, the pilot survey (Section 3.3) enabled us to inspect and determine to what extent the generic architecture, its subsystems, and the corresponding sections, tasks, and sub-tasks were understandable, pertinent and complete. In this preliminary study, the pilot students could give their impressions on the questionnaire and point out any other tasks that were not listed but resulted complex to achieve. This iteration with the initial pilot survey contributed to avoid possible threats concerning the *instrumentation* due to poor question wording, unclear documentation, or bad instrument layout.

Concerning the *subjects selection*, students took part in the survey voluntarily. Nevertheless, we used the draw of a wireless speaker to motivate their

participation. However, as already mentioned in the Survey Design and Methods section, the consistency between the time spent and the number of sections answered, indicates that respondents did not skip several sections or sub-tasks of the questionnaire intentionally to participate in the draw. Consequently, we might say that the voluntary basis and the motivation draw did not influence the obtained results significantly. Furthermore, as stated in Section 3.5 and shown in Table 3, we checked the distribution of exam scores to exclude the risk that only the best students were motivated to participate.

Finally, to avoid the *maturation* threats due to the fatigue or boredom, we allowed the participants to save partially finished surveys, so they had the chance to complete it in different moments. Nevertheless, as shown in Table 4, all the participants completed the survey answering it in their first *attempt*, and completed it within 46 minutes.

6.2. External validity

Threats to external validity are conditions that limit the ability to generalize the experiment results outside the experiment settings [36]. In this survey, external threats to validity had to do with the *representativeness of the subject population* and the *representativeness of the experimental setting*.

In what refers to the *representativeness of the subject population*, in our survey we had that, although most of the respondents belonged to Computer Engineering and Electronic Engineering, the invitations to participate in the survey were addressed to students belonging to 7 different engineering degrees. Additionally, as mentioned in Section 3, some of them were foreign students (Erasmus or other student exchange programs). In fact, on average, the course hosts a cohort (20%-25%) of students from foreign universities [37]. This implies that they received education at different universities and under a different curriculum. Additionally, the ages of the participants ranged from 22 to 39 years old. Finally, we did not exclude participants on the basis of their final grades.

Consequently, the *subject population*, given their heterogeneity regarding the bachelor degree, the home university, the age, and the performance in the

course, although not statistically representative, is a mix that is frequently encountered among IoT novice developers. However, it would be desirable to have had more women participating, as well as a higher percentage of students from other disciplines, apart from Computing Engineering and Electronic Engineering. Nevertheless, given the demographics of the course itself, such a scenario was not likely in the survey.

On the other hand, in pursuit of an *experimental setting representative of the studied context*, we decided to design our survey based on the current IoT state of the art, considering the industrial perspective and using it to analyze the academical projects that the novices implemented. Concerning the industrial perspective, we started from the analysis of various reference IoT architectures that were developed by some of the most influential industry actors in the IoT landscape. Later, concerning the academical perspective, we faced the challenge to make the reference IoT architectures understandable to the students based on the experience that they acquired when developing their projects. To that end, we analyzed the architectures of the Ambient Intelligence course projects developed over the years, identified commonalities, and mapped their components to the building blocks of the industry IoT reference architectures. Based on these analyses, from the reference architectures and the course projects architectures, we developed a generic architecture on which we structured the survey. The purpose of structuring the survey upon this generic architecture was, on one hand, to provide respondents with a common understanding about the software components involved in an IoT system, and on the other hand, to be in line with the industry state of the art, notwithstanding the specificities of the projects that were developed in the course. Similarly, we tried to avoid as much as possible to tie our generic architecture to a specific architectural pattern or technology stack. In this manner, we aimed at guaranteeing that, independently from the software stack, our proposed generic architecture and the survey would be useful in other scenarios to assess the most painful issues that novices experience while developing IoT projects.

However, we must point out that, as already described in Section 3, we

decided not to represent in our generic architecture the Security component because it was outside the course syllabus. Therefore, if this survey would be applied in an IoT-related course in which novice programmers are exposed to the security issues that emerge from security-related operations, a new component must be added to our generic architecture as well as its associated tasks and sub-tasks. Nonetheless, according to the experience reports that we studied in the Related Works, it is very unlikely to introduce security concerns in a course targeted at IoT novice programmers. In the same vein, our focus in this survey was on the software development perspective of IoT systems. If a later study aims at getting understanding about the most painful issues concerning the deployment of the hardware, we consider that the generic architecture is still valid, but the Sensors and Actuators subsystems, which are already represented in the generic architecture, must be defined in terms of a new set of tasks and subtasks. Similarly, on every subsystem, new tasks and subtasks must be added to the ones that we already defined.

6.3. Construct validity

Threats to construct validity refer to the extent to which the experiment setting actually reflects the construct under study [36]. In this survey, construct threats to validity concerned the *inadequate preoperational explication of constructs*, the *mono-method bias*, and, in the field of social threats, the *evaluation apprehension*.

To avoid the *inadequate preoperational explication of constructs*, when formulating the first research question, we opted to study the complexity, which is an ambiguous concept by itself, in terms of well-defined constructs such as difficulty-level and time-spent. This way, before translating these constructs into measures, we made sure that the criteria to quantify the complexity was clear enough to the respondents. Furthermore, to avoid *confounding constructs and levels of constructs*, we decided to translate these difficulty-level and time-spent constructs in a 5-point ordinal scale, considering that more than their absence or presence, it was the level of difficulty and time-spent which is of

importance to the outcome.

Furthermore, to avoid *mono-method bias*, we addressed the research questions using different kinds of measures. Namely, the rating of the sub-tasks (measured using a Likert scale) was complemented with the ranking of the most complex ones and cross-checked with the justification of the participants about their ranking choice (captured through an open question). Thus, by including quantitative and qualitative measures, we aimed at drawing conclusions free from measurement bias.

Regarding the social threats, we opted to conduct the survey anonymously, and only after the participants completed with success the course. In this manner, we were seeking to avoid *evaluation apprehension* so that the participants were guaranteed that they could be sincere about all the issues that they may have experienced without being afraid of some negative impact in their course grade, nor expecting some bonus. Furthermore, it was also explained very clearly to the participants that the study was not aimed at evaluating their skills or performance but to identify the most challenging issues that they experienced during the development of their projects.

6.4. Conclusion validity

Threats to conclusion validity regard issues that affect the ability to draw the correct conclusion about relations between the treatment and the outcome of an experiment [36]. In this survey, conclusion threats to validity regarded the *low statistical power*, the *reliability of measures*, the *reliability of treatment implementation*, and the *random heterogeneity of subjects*.

Concerning the *low statistical power*, in our survey, we would have liked to have a larger sample size to achieve higher statistical power and afford better generalizability. Nevertheless, the length of our survey hampered higher participation (for every completed survey, we had 1.12 abandoned surveys, i.e., students who started but did not complete the survey) and, on the other hand, a shorter version would have covered only a few topics. However, in this regard, it must be clarified that in our survey, more than attempting to infer a prop-

erty of a population, we were trying to explore the variety of challenges that IoT novice developers face. Besides, our open questions could contribute to detecting a mismatch between our interpretation of the data and respondents' experience.

The validity of an experiment is highly dependent on the *reliability of the measures*, and they can be negatively affected by factors such as poor question wording, bad instrumentation, or bad instrument layout. In this sense, we consider that, as previously mentioned in the internal validity threats, the pilot study (Section 3.3) enabled us to avoid these instrumentation-related factors. Furthermore, the *reliability of treatment implementation* describes the risk that the implementation is not similar between different persons applying the treatment or across different occasions. In our case, since the survey was applied online with the same platform, and under the same conditions for all the participants, we consider that the implementation is rather similar between the respondents. Finally, as previously stated in the external validity threats, the subject population has a reasonable degree of heterogeneity but framed in the context of novice IoT programmers. Hence, we consider that such heterogeneity does not lead to *random heterogeneity of subjects*, which describes the risk that the variation due to individual differences is larger than due to the treatment [36].

6.5. Repeatability

Our proposed generic architecture, all its tasks, and their sub-tasks, as well as the structure of the survey, and the followed methodology are available from the authors (Section 3). Therefore, concerning the artifacts used for the survey execution, we consider the repeatability of the results of our study to be good. Nevertheless, we identify three limits to repeatability: although the architecture was proposed to be as generic as possible if it has to be used in specific domains or contexts, new sections must be added; secondly, if our survey is to be used in a different educational scenario, the course to be analyzed has to be project-based, otherways the instrument cannot be applied; finally, if specific technologies, tools

or languages are imposed, our proposed set of tasks and subtasks have to be redefined to match the technical specificities of the concerned technology stack, instead of being technology-neutral as in our work.

7. Conclusion

In this work, a survey was conducted to identify the most complex issues experienced by novice programmers when developing IoT systems. The survey was framed into a generic interpretation framework in which the architecture, subsystems, and software development tasks of a significant subset of these kind of systems were abstracted. This survey was conducted among 40 undergraduate students that developed an IoT project during three editions of a university course. The most complex issues were identified on the basis of the rating of software development tasks according to their difficulty level and completion time; the ranking of the most complex tasks; and the qualitative perception of each respondent about such complexity. Through a generic interpretation framework, a system-level view of the main issues was achieved and presented. To the best of our knowledge, this is the first study to express the complex issues as concrete development tasks that are not dependent on a particular kind of project, its architecture, or its technology stack.

The results that emerged from the application of the survey allowed us to determine that the most challenging issues reported by unexperienced IoT developers concerned: the difficulty to find well-structured documentation that might be understood by a novice, the complexity inherent to the integration of the subsystems, and the integration with third-party services. Moreover, we could also identify, on the one hand, aspects that were perceived as complex across various subsystems (development of user interfaces, the configuration of development environments, and the development of the business logic), and on the other hand, aspects whose complexity is split across various subsystems (integration between the Gateway and the third-party service APIs, the implementation and integration of the persistence component, and the design,

implementation, and consumption of the RESTful web services). We consider that our findings enable to ease the learning curve in the teaching of IoT, and might help to improve on-boarding time estimations, hiring criteria, and human resource management within the industry IoT projects. Our future work will start from the results of the survey for designing and developing mechanisms (both tools and methodologies) targeted at supporting novice programmers in realizing IoT systems. Indeed, a first approach were the aforementioned Code Recipes [32]; summarized and well-defined documentation modules, independent from programming languages or run-time consisting of multiple code fragments along with documentation and links, and whose main purpose is to ease the understanding of such code, in order to implement a given integration between subsystems of an IoT system. Finally, based on our results, we suggest that research efforts should envision automation or debugging tools, as well as improved documentation strategies for the development of IoT systems.

References

[1] D. Miorandi, S. Sicari, F. D. Pellegrini, I. Chlamtac, Internet of things: Vision, applications and research challenges, Ad Hoc Networks 10 (7) (2012) 1497 – 1516. doi:10.1016/j.adhoc.2012.02.016.

[2] J. A. Stankovic, Research directions for the internet of things, IEEE Internet of Things Journal 1 (1) (2014) 3–9. doi:10.1109/JIOT.2014.2312291.

[3] P. Patel, D. Cassou, Enabling high-level application development for the internet of things, Journal of Systems and Software 103 (2015) 62 – 84. doi:10.1016/j.jss.2015.01.027.

[4] F. Corno, L. De Russis, Training engineers for the ambient intelligence challenge, IEEE Transactions on Education 60 (1) (2017) 40–49. doi:10.1109/TE.2016.2608785.

[5] L. Atzori, A. Iera, G. Morabito, The internet of things: A survey, Comput. Netw. 54 (15) (2010) 2787–2805. doi:10.1016/j.comnet.2010.05.010.

[6] C.-W. Tsai, C.-F. Lai, A. V. Vasilakos, Future internet of things: Open issues and challenges, Wirel. Netw. 20 (8) (2014) 2201–2217. doi:10.1007/s11276-014-0731-0.

[7] S. Chauhan, P. Patel, F. C. Delicato, S. Chaudhary, A development framework for programming cyber-physical systems, in: 2016 IEEE/ACM 2nd International Workshop on Software Engineering for Smart Cyber-Physical Systems (SEsCPS), 2016, pp. 47–53. doi:10.1109/SEsCPS.2016.016.

[8] S. K. Datta, C. Bonnet, Easing iot application development through datatweet framework, in: 2016 IEEE 3rd World Forum on Internet of Things (WF-IoT), 2016, pp. 430–435. doi:10.1109/WF-IoT.2016.7845390.

[9] I. Salman, A. T. Misirli, N. Juristo, Are students representatives of professionals in software engineering experiments?, in: 2015 IEEE/ACM 37th IEEE International Conference on Software Engineering, Vol. 1, 2015, pp. 666–676. doi:10.1109/ICSE.2015.82.

[10] A. Taivalsaari, T. Mikkonen, A roadmap to the programmable world: Software challenges in the iot era, IEEE Software 34 (1) (2017) 72–80. doi:10.1109/MS.2017.26.

[11] J.-P. Vasseur, A. Dunkels, Interconnecting Smart Objects with IP: The Next Internet, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2010.

[12] S. K. Datta, C. Bonnet, R. P. Ferreira Da Costa, J. Härri, Datatweet: An architecture enabling data-centric iot services, in: 2016 IEEE Region 10 Symposium (TENSYMP), 2016, pp. 343–348. doi:10.1109/TENCONSpring.2016.7519430.

[13] M. Weyrich, C. Ebert, Reference Architectures for the Internet of Things, IEEE Software 33 (1) (2016) 112–116. doi:10.1109/MS.2016.20.

[14] Intel, The Intel IoT Platform, white paper (2015).
URL https://www.intel.com/content/www/us/en/internet-of-things/white-papers/iot-platform-reference-architecture-paper.html

[15] Microsoft, Microsoft Azure IoT Reference Architecture, white paper (2018).
URL http://aka.ms/iotrefarchitecture

[16] G. Stubbings, S. Polovina, Levering object-oriented knowledge for service-oriented proficiency, Computing 95 (9) (2013) 817–835. doi:10.1007/s00607-013-0304-6.

[17] I. P. Cvijikj, F. Michahelles, The Toolkit Approach for End-user Participation in the Internet of Things, Springer Berlin Heidelberg, Berlin, Heidelberg, 2011, pp. 65–96. doi:10.1007/978-3-642-19157-2_4.

[18] D. J. Cook, J. C. Augusto, V. R. Jakkula, Ambient intelligence: Technologies, applications, and opportunities, Pervasive and Mobile Computing 5 (4) (2009) 277 – 298. doi:10.1016/j.pmcj.2009.04.001.

[19] G. Kortuem, A. K. Bandara, N. Smith, M. Richards, M. Petre, Educating the internet-of-things generation, Computer 46 (2) (2013) 53–61. doi:10.1109/MC.2012.390.

[20] D. Dobrilovic, Z. Stojanov, B. Odadzic, V. Sinik, Platform for teaching communication systems based on open-source hardware, in: 2015 IEEE Global Engineering Education Conference (EDUCON), 2015, pp. 737–741. doi:10.1109/EDUCON.2015.7096051.

[21] D. Dobrilovic, S. Zeljko, Design of open-source platform for introducing internet of things in university curricula, in: 2016 IEEE 11th International Symposium on Applied Computational Intelligence and Informatics (SACI), 2016, pp. 273–276. doi:10.1109/SACI.2016.7507384.

[22] A. Ahmad, K. Li, C. Feng, S. M. Asim, A. Yousif, S. Ge, An empirical study of investigating mobile applications development challenges, IEEE Access 6 (2018) 17711–17728. doi:10.1109/ACCESS.2018.2818724.

[23] M. E. Joorabchi, A. Mesbah, P. Kruchten, Real challenges in mobile app development, in: 2013 ACM / IEEE International Symposium on Empirical Software Engineering and Measurement, 2013, pp. 15–24. doi:10.1109/ESEM.2013.9.

[24] S. M. Sohan, F. Maurer, C. Anslow, M. P. Robillard, A study of the effectiveness of usage examples in rest api documentation, in: 2017 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC), 2017, pp. 53–61. doi:10.1109/VLHCC.2017.8103450.

[25] M. P. Robillard, R. DeLine, A field study of api learning obstacles, Empirical Software Engineering 16 (6) (2011) 703–732. doi:10.1007/s10664-010-9150-8.

[26] G. Uddin, M. P. Robillard, How api documentation fails, IEEE Software 32 (4) (2015) 68–75. doi:10.1109/MS.2014.80.

[27] T. Koulouri, S. Lauria, R. D. Macredie, Teaching introductory programming: A quantitative evaluation of different approaches, Trans. Comput. Educ. 14 (4) (2014) 26:1–26:28. doi:10.1145/2662412.

[28] F. Corno, L. De Russis, J. Sáenz, Pain points for novice programmers of ambient intelligence systems: An exploratory study, in: 2017 IEEE 41st Annual Computer Software and Applications Conference (COMPSAC), Vol. 1, 2017, pp. 250–255. doi:10.1109/COMPSAC.2017.186.

[29] Limesurvey: the online survey tool - open source surveys, https://www.limesurvey.org, accessed: 2018-12-11.

[30] V. Braun, V. Clarke, Using thematic analysis in psychology, Qualitative Research in Psychology 3 (2) (2006) 77–101. doi:10.1191/1478088706qp063oa.

[31] L. Dabbish, C. Stuart, J. Tsay, J. Herbsleb, Social coding in github: Transparency and collaboration in an open software repository, in: Proceedings of the ACM 2012 Conference on Computer Supported Cooperative Work, CSCW '12, ACM, New York, NY, USA, 2012, pp. 1277–1286. doi:10.1145/2145204.2145396.

[32] F. Corno, L. De Russis, J. P. Sáenz, Easing iot development for novice programmers through code recipes, in: Proceedings of the 40th International Conference on Software Engineering: Software Engineering Education and Training, ICSE-SEET '18, ACM, New York, NY, USA, 2018, pp. 13–16. doi:10.1145/3183377.3183385.

[33] K. Pohl, G. Böckle, F. J. v. d. Linden, Software Product Line Engineering: Foundations, Principles and Techniques, Springer-Verlag, Berlin, Heidelberg, 2005.

[34] A. Abbas, I. F. Siddiqui, S. U. Lee, A. K. Bashir, Binary pattern for nested cardinality constraints for software product line of iot-based feature models, IEEE Access 5 (2017) 3971–3980. doi:10.1109/ACCESS.2017.2680470.

[35] T. D. Cook, D. Campbell, Quasi-Experimentation: Design and Analysis Issues for Field Settings, Houghton Mifflin, 1979.

[36] C. Wohlin, P. Runeson, M. Hst, M. C. Ohlsson, B. Regnell, A. Wessln, Experimentation in Software Engineering, Springer Publishing Company, Incorporated, 2012.

[37] F. Corno, L. De Russis, D. Bonino, Educating internet of things professionals: The ambient intelligence course, IT Professional 18 (6) (2016) 50–57. doi:10.1109/MITP.2016.100.