Easing IoT Development for Novice Programmers Through Code Recipes

Original

Availability:
This version is available at: 11583/2698336 since: 2018-06-19T13:44:49Z

Publisher:
ACM

Published
DOI:10.1145/3183377.3183385

Terms of use:
openAccess
This article is made available under terms and conditions as specified in the corresponding bibliographic description in the repository
Easing IoT Development for Novice Programmers
Through Code Recipes

Fulvio Corno  
Politecnico di Torino  
Turin, Italy  
fulvio.corno@polito.it

Luigi De Russis  
Politecnico di Torino  
Turin, Italy  
lugi.derussis@polito.it

Juan Pablo Sáenz  
Politecnico di Torino  
Turin, Italy  
juan.saenz@polito.it

ABSTRACT
The co-existence of various kinds of devices, protocols, architectures, and programming languages make Internet of Things (IoT) systems complex to develop, even for experienced programmers. Perforce, Software Engineering challenges are even more difficult to address by novice programmers. Previous research focused on identifying the most challenging issues that novice programmers experience when developing IoT systems. The results suggested that the integration of heterogeneous software components resulted one of the most painful issues, mainly due to the lack of documentation understandable by inexperienced developers, from both conceptual and technical perspectives. In fact, novice programmers devote a significant effort looking for documentation and code samples willing to understand them conceptually, or in the worst case, at least to make them work. Driven by the research question: “How do the lessons learned by IoT novice programmers can be captured, so they become an asset for other novice developers?”, in this paper, we introduce Code Recipes. They consist of summarized and well-defined documentation modules, independent from programming languages or run-time environments, by which non-expert programmers can smoothly become familiar with source code, written by other developers that faced similar issues. Through a use case, we show how Code Recipes are a feasible mechanism to support novice IoT programmers in building their IoT systems.

CCS CONCEPTS
• Social and professional topics → Computational science and engineering education; Software engineering education;  
• Computer systems organization → Embedded and cyber-physical systems;

KEYWORDS
Novice programmers, Internet of Things, Documentation, Code Fragments

1 INTRODUCTION
The development of IoT systems is challenging. On one hand, it relies on various areas such as distributed systems, mobile computing, web information systems, and cloud computing, among others. On the other hand, it differs from mainstream mobile-app and client-side web application development in a sense that IoT developers are required to consider aspects such as: the multidevice programming; the reactive, always-on nature of the system; heterogeneity and diversity; the distributed, highly dynamic, and potentially migratory nature of software [8].

Naturally, these challenging issues are even more painful for novice programmers since they are not expected to have deep knowledge or experience in all those areas or aspects [5]. Our previous research aimed at identifying the pain points that novice programmers experienced when developing IoT systems [2]. An exploratory study was conducted among Electronic and Computer Engineering undergraduate students of a university course in which, following a project-based learning approach, three to four people groups were assigned to develop an IoT system. In accordance with the course learning goals, these IoT systems had to include mobile applications, web applications, cloud computing services, wearable devices, single-board computers, and IoT sensing devices [1].

The results from this exploratory study suggested that the integration of heterogeneous software components is one of the most painful issues. It commonly implies dealing with several protocols, formats, and authentication mechanisms, that are usually unknown to the students. Moreover, the lack of clear and complete documentation, or merely, the absence of documentation that can be understood by a novice developer, make this integration issue even more difficult to overcome.

Looking for solutions to support novice IoT developers in overcoming these integration issues, we noticed that despite the specificity of each project, implementations of the integration between software components were similar across most of them, especially when third-party services were involved. However, although the source code of the projects from the past years’ courses is on GitHub, it was not being reused among groups in later versions of the course. Therefore, the lessons learned by a group when implementing its project is not useful for the next year’s groups.

Taking into account the results of the exploratory study and the lack of code reuse between the course groups, we envisioned that the the solutions found by the students, that were finally included in the working prototype built at the end of the course, could become a valuable asset for the novices that are about to start implementing their projects. The source code of these prototypes reveals architectural decisions and strategies adopted by other groups to achieve the integration of diverse software components. This code should, therefore, provide some guidance to other programmers that are in the process of overcoming the same learning curve issues.

Moreover, if documented, this code would be a solution to the reported lack of documentation understandable by inexperienced developers [9]. 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 [3].

The present work is driven by the research question: “How do the lessons learned by IoT novice programmers can be captured, so they become an asset for other novice developers?”. The current proposal
aims at easing the learning curve to IoT novice developers, not by automating code reusing and hiding the code from the developers, but instead, by enabling non-expert programmers to easily become familiar with source code, written by other developers that faced similar issues.

2 USE CASE

As mentioned earlier, the results from our previous research [2] suggested that among the most challenging issues novices face when developing IoT systems, the integration with other software components was perceived by many students as the most painful issue. In particular, the integration with third-party APIs that require OAuth 2.0 authentication was a time-consuming and difficult task. The OAuth 2.0 authorization framework enables a third-party application to obtain limited access to an HTTP service, either on behalf of a resource owner by orchestrating an approval interaction between the resource owner and the HTTP service, or by allowing the third-party application to obtain access on its own behalf. Broadly speaking, this authentication protocol consists of a flow, with a set of roles (resource owner, resource server, client, and authorization server) interacting across various steps such as authorization request, access token request, refresh token, redirect URI.

In the development of IoT systems, OAuth authentication protocol becomes fundamental since most of the third party service APIs use it. The integration with the Fitbit activity tracker is a concrete example of the OAuth protocol usage. In order to gather the data captured by this wearable device, the third party application (i.e., the one developed by the novices) must obtain users authorization through the OAuth protocol.

However, due to the roles, steps, and resources that the protocol comprises, the adoption of the OAuth authentication is not trivial. The appropriate implementation of this protocol requires a clear understanding of the various steps, both from the conceptual and the technical perspective. Novice programmers struggle considerably with the adoption of OAuth, mainly due to the lack of documentation that might be understandable by non-expert programmers.

Fitbit, for instance, has a documentation website that provides guidance about the Web API for accessing data from Fitbit activity trackers. Although the developer’s site has an API explorer built in Swagger, and an API debug tool, it does not provide a fully implemented functional source code sample. Moreover, despite the clarity, readability and good overall structure of the documentation, it is targeted at experienced programmers, as with most of the developer’s documentation.

In this scenario, novice programmers are required to search code samples, willing to understand them conceptually, or in the worst case, at least to make them work. Typically, this involves the reference of the Google OAuth Client Library documentation, the Fitbit developers website, several posts published in Stack Overflow, and various code samples available on GitHub. Hence, from the experience of the novice programmers adopting the OAuth protocol, we have that: (i) a significant amount of effort is devoted looking for documentation and samples; (ii) just through the source code it is not possible to understand the whole learning process behind it; (iii) the code fragments must be surrounded by summarized, structured and well-defined documentation modules, so they become an asset for other IoT novice programmers.

3 CODE RECIPES

Code Recipes aims at capturing the most important information and documentation about one or more code fragments, to ease the development of an IoT system for novice developers. Code Recipes 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. The joint presence of metadata and links allow novice developers to explore alternative solutions and, at their will, deepen their knowledge about a specific IoT subsystem, thus contributing to their learning process.

Our approach lies in the fact that code examples, when used effectively, can be a powerful learning resource [4]. However, while examples are a valuable resource for programmers, the rich context surrounding them is often crucial for adaptation and integration [6]. This proposal enables the integration of several software components through code fragments that might belong to different programming languages and might be deployed across various runtime environments, as it is common in IoT systems. The decoupling between the recipes and the technological stack is fundamental given the heterogeneity of the software components that are involved in an IoT system. Code Recipes, therefore, are defined as summarized and well-defined documentation modules, independent from programming languages or runtime environments.

By defining Code Recipes as documentation modules structured around code fragments, they can be incorporated in various kind of tools that might handle them in the learning process, e.g., a wiki-style web application or an Integrated Development Environment (IDE) extension.

Code Recipes, therefore, expose four features:

- Although the Recipes are structured around source code fragments, they are much more than just code. They encompass information that, besides providing technical solutions, includes comments and documentation sources that account for the learning process that other novice IoT developers followed and the decisions they made to reach a solution.
- Recipes are not constrained to a specific architecture, programming language or runtime environment. This means, first, that this proposal is aware of the heterogeneous nature of IoT environments, and second, that is suitable to be used in multiple scenarios with IoT novice developers.
- Recipes are not isolated from each other, they are cross-linked on the basis of three criteria: alternative versions, other language versions, and related recipes. This feature enables the sharing of diverse learning experiences with their commonalities and their divergences.
- Technical speaking, a structured representation (e.g., in JSON or XML) of the Code Recipes enables the implementation of various kind of tools that might handle them. For instance, a

Listing 1: Code Recipe Sample

Listing 1 describes a possible structure of a Code Recipes in JSON format. First, each recipe is described through an id (timestamp), its author name, publication date, name, description, and tags (lines 2 to 9). Then, the subsystems that the recipe integrates are specified in the endpoints fields (lines 10 and 11). Ingredients (line 12) correspond to the requirements of the recipe. They can be technical requirements, such as the deployment of a specific kind of web server, or data requirements, such as creating a developer account and issuing API client credentials. Dependencies (line 16) refers to requirements associated with the source code, which are fundamentally libraries and packages that must be installed.

Most importantly, Code Recipes include one or more code fragments that can be implemented in different programming languages and IDEs (lines 20 to 33). Each fragment has a set of parameters, which are values specific to each implementation of the recipe. Besides the source code, recipes include the documentation that their authors consulted, both for the whole recipe as well as for its fragments. They can be specified in the documentation_urls fields (lines 23 and 34). Finally, Code Recipes can be linked to each other in three ways (lines 36 to 38): alternative versions, that point to other recipes targeted at implementing the same integration; other language versions, that point to implementations of the same recipe in other programming languages; and related recipes, that correspond to other recipes that can be used as intermediate steps to implement the concerned recipe.

4 THE FITBIT OAUTH CODE RECIPE

With the use case described in Section 2 in mind, a Code Recipe was developed to illustrate how our approach might help novices to overcome integration issues through a collaborative approach. To develop this recipe, we took on the task of implementing a simple Java application to gather data from a Fitbit bracelet.

As mentioned before, no sample projects are provided in the Fitbit developers’ website. Therefore, the first endeavor was to find a sample project in which the OAuth authentication was implemented using Java. After googling “OAuth 2.0 Java Sample Code”, the second result took us to the documentation of the Google OAuth Client Library for Java (in the Code Recipes, this website would be included in the documentation_urls field). This website had setup instructions for Maven, the list of libraries that were required (in the Code Recipes would correspond to the dependencies field), the release notes of these libraries, and one sample code of the integration between a Java application and the DailyMotion API², using OAuth 2.0.

Once downloaded and imported the sample code into the IDE, the next step was to install and configure Maven, including the Project Object Model (POM) in which the dependencies of the project were defined. Later, when the Java project was already compilable, the next task was to identify which pieces of the code should be modified to achieve the integration with the Fitbit API (in the Code Recipes, these pieces are specified in the parameters field). Among the new data that had to be inserted into the code as parameters, there were the API key, the API secret, the Callback URL and the Scope. All of this data was obtained after completing the registration as a Fitbit developer (in the Code Recipes this registration accounts as an ingredient)

familiar with source code written by other developers that faced similar issues. Future work will concern: the development of a Code Recipes catalog; a web-based tool through which students can use them; and the subsequent evaluation in the context of the course.

REFERENCES

[1] Fulvio Corno, Luigi De Russis, and Dario Bonino. 2016. Educating Internet of Things Professionals: The Ambient Intelligence Course. IT Professional 18, 6 (Nov 2016), 50–57. https://doi.org/10.1109/MITP.2016.100


