The Journal of Systems & Software 169 (2020) 110706

Contents lists available at ScienceDirect

The Journal of Systems & Software

journal homepage: www.elsevier.com/locate/jss

In practice

Examining the reuse potentials of IoT application frameworks

Paraskevi Smiari [a], Stamatia Bibi [a,*], Daniel Feitosa [b]

[a] Department of Electrical and Computer Engineering, University of Western Macedonia, Greece
[b] Data Research Centre, University of Groningen, The Netherlands

ARTICLE INFO

Article history:
Received 5 November 2019
Received in revised form 5 June 2020
Accepted 22 June 2020
Available online 24 June 2020

Keywords:
IoT applications
Reusability
Black-box reuse
White-box reuse effort estimation

ABSTRACT

The major challenge that a developer confronts when building IoT systems is the management of a plethora of technologies implemented with various constraints, from different manufacturers, that at the end need to cooperate. In this paper we argue that developers can benefit from IoT frameworks by reusing their components so as to build in less time and effort IoT systems that can easily integrate new technologies. In order to explore the reuse opportunities offered by IoT frameworks we have performed a case study and analyzed 503 components reused by 35 IoT projects. We examined (a) the *types of functionality* that are most facilitated for reuse (b) the *reuse strategy* that is most adopted (c) the*quality of the reused components*. The results of the case study suggest that the main functionality reused is the one related to the Device Management layer and that Black-box reuse is the main type. Moreover, the quality of the reused components is improved compared to the rest of the components built from scratch.

1. Introduction

Over the last decade, the emerging paradigm of the Internet of Things (IoT) dominates the digital transformation (Zimmermann et al., 2015) landscape, providing benefits related to better quality of life and greater insight into businesses. The IoT describes the network of devices that are connected via the Internet, enabling the collection, exchange and analysis of generated information. According to Statista,[1] by 2025, the total amount of installed IoT devices will reach 75.44 billion worldwide, which corresponds to a fivefold increase compared to the amount of 2015. The next generation of mobile connection technology, 5G, is expected to boost the application of IoT in everyday life, further democratizing a range of services (Zanella et al., 2014) that include, among others, healthcare, education, manufacturing and home automation. IoT is considered as a system of systems that comprises physical things, various communication channels and a combination of complete software solutions, including data and operations. Hence, managing various hardware, software and network technologies, in order to implement IoT applications, has proven to be a major challenge for developers.

Given the aforementioned, developing IoT applications with the conventional platforms, is becoming increasingly difficult for developers that lack experience and knowledge (Zanella et al., 2014) in a domain that is still in its infancy. The main difficulty is that developers need to adapt their applications to multiple platforms and spent effort to learn and use platform APIs and information models (Bröring et al., 2017). To deal with this problem literature has suggested the use of frameworks (Aly et al., 2018; Ciccozzi and Spalazzese, 2016; Yelamarthi et al., 2017) that are platform independent and consist of highly modularized software building blocks (Kim et al., 2012; Serna et al., 2015). Typically, IoT frameworks present a set of common concepts (e.g., Devices, Gateways, Data Management) (Zimmermann et al., 2015) that can be reused by developers in order to save time and effort. In the recent years, Open Source Software (OSS) frameworks have become a key supplier of critical software components (Paschali et al., 2017), dominating the industry of IoT (Aly et al., 2018; Tanganelli et al., 2015). Currently there are several available OSS frameworks with big support from the community that can facilitate reuse in the context of IoT application development. However, as pointed by Chen et al. (2008), the reuse of OSS components encompasses challenges with respect to component selection, component integration, and system maintenance (Chen et al., 2008). Thus, such endeavor involves three major tasks that the reuser needs to perform:

(a) **Identify the reusable asset:** In this step the reuser has to identify the most suitable asset that implements the type of functionality she wants to integrate in the new system (Paschali et al., 2017). According to Schwittek and Eicker, it is of paramount importance to define the reused functionality based on the application domain (Schwittek and Eicker, 2013) where it will be

* Correspondence to: Department of Electrical and Computer Engineering, University of Western Macedonia, Karamanli & Ligeris, Kozani, 50100, Greece.
E-mail addresses: psmiari@uowm.gr (P. Smiari), sbibi@uowm.gr (S. Bibi), d.feitosa@rug.nl (D. Feitosa).

[1] https://www.statista.com/statistics/976045/iot-revenue-forecast-worldwide/.

https://doi.org/10.1016/j.jss.2020.110706
0164-1212/

integrated and the associated requirement (Raemaekers et al., 2012) that it is intended to fulfill. In the IoT application domain, the reusable functionalities can be mapped to the layers of the architecture (Zimmermann et al., 2015) of a typical IoT system. According to Zimmerman (Zimmermann et al., 2015), an IoT solution comprises of many ***devices*** that ***communicate*** through a network and generate data that are ***integrated*** to be ***managed*** in the IoT cloud platform and further ***processed*** to be ***presented*** to the end-user. All of the above functionalities that are supported by IoT systems require the relevant ***identification and access management*** mechanisms. It is important for the reuser to examine the IoT reuse options based on the types of functionality that the reusable components offer. For example, it is expected that components related to Communication Protocols may be more easily available compared to components related to Devices virtualization. Taking into consideration the availability of the reusable resources, the reuser may optimize the reuse process.

(b) **Integrate the reusable asset:** In this step the developer has to decide upon the strategy for integrating the reusable asset to the new system. There are several cases where the assets are integrated in the new system simply as they are, indicating ***Black-box reuse*** (Frakes and Terry, 1996). In other cases, new functionalities must be implemented on the reused components for integrating them, indicating ***White-box reuse*** (Frakes and Terry, 1996). In the latter case it is important to have an approximation of the effort required to integrate the reused component, which can be measured as the effort required to apply the changes. In IoT development, due to the large scale of operations when building applications (Tanganelli et al., 2015), it is important to know from the beginning the reuse strategy for integrating components based on the types of functionalities that can be reused. For example, it is highly possible to retrieve Device Management components (i.e., Operational components) that can be used as is, whereas Presentation components (i.e., Building Charts components) may require a lot of customization and potentially deemed prohibitive.

(c) **Evaluate the reusable asset:** In this step the reuser needs to ensure that the quality of the reusable components does not compromise the overall quality of the application (Bibi et al., 2010; Lim, 1994). This is particularly important for IoT application components, since they are expected to possess various quality characteristics that are related to their ability to be (a) ***extendible*** (Smiari and Bibi, 2018) so as to handle the variability of heterogeneous devices and technologies, (b) ***flexible*** (Kim et al., 2012) so as to easily adapt to changes caused by the external environment and implement new requirements, (c) ***reusable*** (Lazarescu, 2014) so as to allow further reuse in future IoT applications, saving time and effort, and (d) ***functional*** (Bröring et al., 2017) so as to offer several functionalities through their public APIs. We clarify that other quality attributes, e.g., understandability and effectiveness, may also be important (Bansiya and Davis, 2002). For this purpose, it is important to examine the quality of the reused components to ensure that they do not deteriorate the quality of the new system.

In this study we examine the reuse opportunities offered by OSS IoT frameworks in order to assist practitioners in performing the aforementioned tasks for deciding upon the following:

- What functionalities can be reused from IoT frameworks?
- Which reuse strategy should be adopted with respect to the functionality reused?
- What is the quality of the reused components with respect to the functionality reused?

To answer the aforementioned concerns, we have performed a case study on 503 components, originating from 7 different IoT projects coming from the Eclipse IoT framework that has been reused by 35 IoT projects. Eclipse IoT framework was selected because it is frequently used in research (Aly et al., 2018; Kovatsch et al., 2014; Smiari et al., 2019), has a strong support from industrial players like Bosch and QIVICON, and is often reused for building commercial products (e.g., Mixtile Hub, Coqon). In order to draw conclusions regarding reuse opportunities from IoT frameworks, we have downloaded and analyzed the source code of the applications that reused Eclipse IoT frameworks and examined the components that were reused.

The rest of this paper is organized as follows. Section 2 discusses the related work. Section 3 describes the case study design whereas Section 4 presents the results obtained from the case study. Section 5 discusses the results of this case study and Section 6 addresses the threats to validity of the research performed. Finally, Section 7 concludes the paper and presents ideas for future work.

2. Related work

2.1. Software reuse

Software reuse is a widely known and used technique for the creation of a new software product that is based on the adoption of existing software components. Reusable components can bring many benefits in the software development process some of them being the reduction of cost and effort (Wangoo and Singh, 2018) and the increase of productivity (Lim, 1994). Research activities focusing on the advantages of software reuse, have also pointed out that it has a positive impact in quality (Arvanitou et al., 2016) and maintainability (Lim, 1994) of the software. Software reuse is also known in literature for the improvement of software system flexibility (Jatain et al., 2013). This is achieved through the separation of the stable parts of systems from the specification of their composition (Jatain et al., 2013). Improved flexibility has also been identified with the usage of frameworks as argued by Wang et al. (1999) who proposed a framework which aims at assembling components in distributed systems. Many research efforts are found in literature examining various aspects of software artifact reusability (Jatain et al., 2013). We will describe these efforts by categorizing them based upon the tasks of the software reuse process that we referred to in Section 1: (a) the reusable component identification (b) the reuse strategy adopted regarding the integration of the component to the target system (c) the evaluation of the reused components.

The first step, when it comes to the reusability of components, is the identification of the appropriate reusable software component. Research has focused on clustering techniques in order to identify components of similar functionality and reusability. Such techniques have been thoroughly discussed by Jatain et al. in Jatain et al. (2013) as well as Saied et al. (2018) who managed to spot reusable components on third party libraries through usage pattern mining. Automated mechanisms that identify reusable components by searching routines with a specific input have also been proposed by Podgurski and Pierce (1993). The authors created a method that identifies reusable components through behavior sampling by giving a randomly chosen sample. When it comes to selecting the most appropriate candidate for reuse research has shown that familiarity with the domain or architecture (Torchiano and Morisio, 2004) is a major factor. Other techniques to prioritize and select the most suitable candidate, besides interoperability (Tran et al., 1997), is to take into account the functional and non-functional requirements (Tran et al., 1997) of the specific domain. By evaluating reusable components through metrics that indicate the acceptance of a component by the developers (Papamichail et al., 2018), we can effectively estimate the reusability, leading to a more suitable selection prior to integrating it.

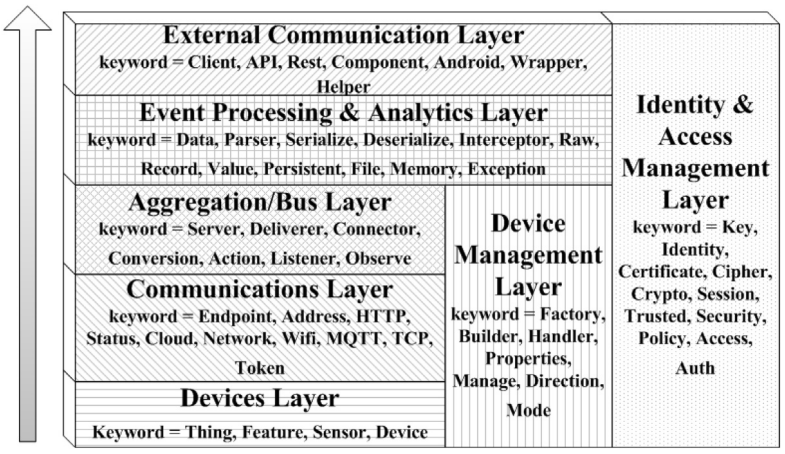


Fig. 1. IoT reference architecture (Fremantle, 2015).

The next step, after identifying the suitable reusable component, is to integrate it into the target software system. Ye et al. (2000) has mentioned several limitations when it comes to integrating reusable components in a system. These limitations can be extracted from the environment we want to integrate the component in and such can be the implementation language, the performance, or the quality. It is observed that in service oriented architectures the potential in reusability is high because developers often choose to learn an existing service than to implement it from scratch, thus making it necessary for the existence of an integration framework (Zhu, 2005). Such framework is presented by Yu et al. aiming at integrating user interface components with the help of a declarative composition language called XPIL (Yu et al., 2007) which is based on XML. Although, it is often required to parameterize the reusable components in order to integrate them in the existing environment (Gupta et al., 2010), research has shown that, by continuously testing components and recommending those that have the lowest failure rate (Kessel and Atkinson, 2018), it can enable the integration of reusable components especially in an environment that supports continuous integration.

The third step we need to consider, when creating reusable components, is to evaluate the reused component in terms of quality. A wide range of studies have been performed for assessing the quality of a certain reusable artifact based on structural properties (e.g., encapsulation, coupling and cohesion). Bansiya and Davis (2002) proposed QMOOD, a hierarchical quality model, for assessing the quality of object-oriented artifacts that relates structural properties to high-level quality attributes (e.g., reusability, flexibility, etc.). Prakash et al. used a suite of metrics (Prakash et al., 2012), that he divided into procedural and object oriented, based on the classical metrics suite proposed by Chidamber and Kemerer (1994). These metrics were also used by Padhy et al. (2018) in order to create prediction methods to spot whether or not components are qualified as reusable. Additionally, reusability indices (Ampatzoglou et al., 2018) have been introduced that are based on both non-structural and structural metrics. In this context data mining techniques (Prakash et al., 2012; Wangoo and Singh, 2018) have been adopted for evaluating the quality of the reusable assets by taking into consideration several software quality metrics, some of them being the metrics suite provided by Chidamber and Kemerer (1994).

In this study we have selected to adopt the QMOOD model (Bansiya and Davis, 2002) to assess the quality of the reusable components since it is a model that has been thoroughly used in literature to investigate software quality (Couto et al., 2018; O'Keeffe and Cinnéide, 2006; Osbeck et al., 2011) and software reusability (Ampatzoglou et al., 2011; Ani et al., 2017). Additionally, another motivation for selecting QMOOD (Bansiya and Davis, 2002), is the fact that it provides 6 indices, Reusability, Extendibility, Flexibility, Functionality, Understandability and Effectiveness that assess the quality aspects that are of interest in the case of component reuse in the context of IoT (Bröring et al., 2017; Kim et al., 2012; Lazarescu, 2014; Smiari and Bibi, 2018).

2.2. Designing IoT applications

In this section we will initially present the typical architectural design of IoT applications and then proceed with describing the efforts performed for facilitating reuse in the IoT context. Several studies can be found in literature that propose reference architectures for developing modular and reusable software services for IoT systems (Yelamarthi et al., 2017; Zimmermann et al., 2015). According to Fremantle (2015) an IoT architecture consists of seven layers (see Fig. 1).

The *Devices* layer, represents a plethora of heterogeneous devices that are connected to the internet usually as digital twins stored to the cloud. The *Communication* layer, supports the connectivity of devices through several communication protocols. The *Aggregation/Bus* layer works as a gateway for the different devices and enables the communication between devices as well as bridging and transforming between protocols. The *Event Processing & Analytics* layer, represents the data retrieved from the bus layer and stored to the database for analyzing events. The *External Communication* layer, indicates the communication achieved outside of devices with the usage of processing models. The *Device Management* layer includes all the functions available that handle devices. Lastly the *Identity & Access Management* layer, which is responsible for controlling the access provided, managing different identities and managing the security of the devices and the environment in general.

The similarities between the different IoT platform architecture models are appointed by Guth et al. (2018). Guth et al. concluded that the various layers of the different architectures can be mapped to a single abstract reference architecture. As an example the architecture proposed by Cisco (The Internet of Things Reference Model Whitepaper, 2014) offers similar functionalities and presents common layers to Fremantle (2015) as the "Communication layer" in Fremantle's architecture (Fremantle, 2015) can be mapped to the "Connectivity" layer in Cisco (The

Internet of Things Reference Model Whitepaper, 2014). Another similar architecture is the one proposed by Krčo (Krčo et al., 2014) who suggests layers such as the "Device" layer. The IoT architecture proposed by Yelamarthi et al. (2017) presents a "Sensors" layer corresponding to the "Devices layer" in Fremantle (2015). Additionally, the "External Communications layer" (Fremantle, 2015) can be found in the "User interface devices" layer in the IoT architecture proposed by Yelamarthi et al. (2017).

Several researchers adopted the architecture presented in Fremantle (2015) to build IoT platforms. Specifically, Zamfir et al. (2016) proposed a platform for prototyping IoT applications in the healthcare monitoring sector. Additionally, Neagu et al. (2016) focused on creating cloud solutions for IoT in the healthcare sector, introducing a Sensing as a Service platform. Levina et al. (2017) adopted the IoT reference architecture presented in Fremantle (2015) for developing an intelligent transportation system. The reference architecture proposed by Fremantle was also used as a ground rule for implementing IoT platforms for Smart Homes by Pessoa and Duarte-Figueiredo (2017). The architecture was chosen due to its extensibility and modularity while the main goal was to increase specific aspects of smart homes, such as security.

Reuse potentials in the IoT domain, with respect to the different types of IoT services have been explored by a few researchers. Katasonov et al. introduced a middleware platform called UBIWARE (Katasonov et al., 2008) for supporting the communication between IoT services. This platform consists of a selection of tools that enable the implementation of agents and adaptors. The middleware is based on the usage of reusable Java components, which enable communication and collaboration of services in heterogeneous environments. Additionally, Lazarescu (2014), introduced a platform focused on establishing wireless communications in the context of IoT for achieving low cost and long-term environmental monitoring. The authors defined generic requirements gathered from different IoT environmental monitoring applications and managed to structure a platform which can be reused in a wide range of similar long-term environmental monitoring applications. Cicozzi et al. promoted the reusability of design artifacts in IoT systems through Model Driven Engineering and self-adaptive systems proposing the MDE4IoT framework (Ciccozzi and Spalazzese, 2016). A model-based system was also introduced by Shani and Broodney who proposed a solution to reuse already existing IoT models (Shani and Broodney, 2015). To achieve that the authors adopted semantic mediation models to spot commonalities between models. Chatuverdi et al. examined the reusability of data management functionality focusing on reusing streaming dataflows (Chaturvedi et al., 2007). The authors proposed dataflow reuse algorithms that identify common tasks that can be reused in order to create a merged dataflow containing all the streams. Smirek et al. examine possible reusability at the user interface level (Smirek et al., 2016) by comparing two frameworks, one of which being part of the Eclipse IoT family. The authors examined whether the functionalities of the two frameworks allow them to create a more abstract and customizable user interface.

Despite the fact that several researchers proposed models and frameworks that can support the reuse in the specific layers of the IoT applications architecture, the reuse from existing open source frameworks has not been yet addressed. In this paper we examine the reuse potentials of open source components that are already available and can support the development of platform independent IoT applications.

3. Case study design evaluation

In this section, we present the design of the case study performed to assess the reusability potentials of IoT frameworks. We report the details of this case study based on the guidelines of Runeson and Höst (2009). Our work comprises an embedded multiple-case study, where the involved contexts are the different IoT functionality types, the cases are the IoT projects and the units of analysis are their reused components. Thus, in Section 3.1, we present the research objectives of the study. In Sections 3.2 and 3.3, we describe the IoT frameworks that participate in this study and the data collection processes. Finally, in Sections 3.4 and 3.5 we provide an overview of the data and the data analysis process.

3.1. Research objectives and questions

The overall goal of this case study is to examine the reuse potentials of IoT application frameworks with respect to (a) the type of functionality that the reused component implements, (b) whether customization is required for integrating the reused component to the target system, and (c) the quality of the reused components compared to the native ones. To ease the design and reporting of the case study, we split the aforementioned goal into three research questions based on the analysis perspectives that we introduced in Section 1.

(RQ1) Which types of functionality offer the most components in the context of IoT application development?

This research question aims at identifying the types of functionalities implemented in IoT frameworks that offer the larger pool of reusable components. The classification scheme adopted to assess the types of functionalities was inspired by the architecture proposed by Fremantle (2015). We choose to adopt this architecture model since it has been used in numerous research activities (Neagu et al., 2016; Pessoa and Duarte-Figueiredo, 2017; Zamfir et al., 2016). Also we believe that the specific reference architecture presents commonalities with the majority of the rest of the proposed architectures (Krčo et al., 2014; The Internet of Things Reference Model Whitepaper, 2014; Yelamarthi et al., 2017) across literature, thus making it a respectable choice. This architecture defines seven core types of functionalities according to the corresponding layers: the *Devices* layer, the *Communication* layer, the *Aggregation/Bus* layer, the *Event Processing & Analytics* layer, the *External Communication* layer, the *Device Management* layer and the *Identity & Access Management* layer (Fremantle, 2015; Zimmermann et al., 2015). A more detailed presentation of the different functionalities offered by each layer can be found in Section 2 and Fig. 1.

The answer to this research question will provide both researchers and practitioners with an overview of the reuse potentials offered by each type of functionality. On the one hand, practitioners will be aware of the types of components that can be more easily found and reused in the context of IoT applications. On the other hand, researchers will be able to identify types of implemented functionalities that are currently sparsely reused, examine the associated rationale and propose solutions, e.g., to prioritize maintenance and development of new reusable components.

(RQ2) Do the reused software components require customization?

This research question aims to identify whether or not the reused components are customized when integrated to the target system and, if customized, to analyze the measured effort. The term customization, in software component reuse, refers to the modification of the source code of the reused components so as to fulfill the purpose of the target system where they are

employed (Frakes and Terry, 1996). This research question is further decomposed as follows.

RQ2.1 Which reuse strategy (i.e. White-box reuse or Black-box reuse) is adopted when integrating the reused IoT components to the target application? To answer this RQ we discriminate between white-box reuse and black-box reuse (Heinemann et al., 2011). *White-box* reuse according to Ravichandran and Rothenberger (2003) and Frakes and Terry (1996) allows for the customization of the reused artifact while *Black-box* reuse aims to the direct integration of the artifact based on its API using software components "as is", with no code modification.

The analysis in this question will provide an overview of how many of the reused components where customized (white-box reuse) and how many were not (black-box reuse) per type of functionality. The answer to this RQ is useful to both researchers and practitioners since it will appoint the types of functionalities that can be reused "as is". Such IoT functionalities can be universally standardized in order to allow platforms interworking, a fact that currently is very important when it comes to IoT application development.

RQ2.2 What is the customization effort required to integrate the reused components in the case of White-box reuse?

This research question captures the customization effort required for tailoring the functionality of the software component to fit in the target system when White-box reuse strategy is adopted. We clarify that Black-box reuse was not considered in this RQ since it requires no modifications (Gaffney and Durek, 1989). According to Frakes and Terry (1996), the customization effort is the cost associated with the new code, implemented for incorporating the reused code into the target system. The integration effort, in our case, is measured as the number of source lines of code (SLOC) for implementing new functionalities related to the reused artifacts. In particular, the effort is calculated as the SLOC added, deleted or modified in order to customize the original component that exists in the IoT development framework to the target system. The use of SLOC is a common practice when measuring the effort required to implement software within the context of software reuse (Gui and Scott, 2006; Prieto-Diaz and Freeman, 1987).

The output of this research question will provide insights to practitioners on the effort required, when adopting White-box reuse for each type of IoT functionality reused.

(RQ3) Do reused components present higher quality compared to the native components in the context of IoT application development?

This research question aims at investigating if the quality of the reused components is higher compared to the quality of the native components, in which the reused ones are introduced. The quality of components measured by assessing six indices of the QMOOD model (Bansiya and Davis, 2002), Reusability, Flexibility, Extendibility, Functionality, Understandability and Effectiveness. The QMOOD model was selected since it has widely been used in literature (Ani et al., 2017; Couto et al., 2018; O'Keeffe and Cinnéide, 2006) and it includes a variety of quality aspects that are appointed as important when evaluating IoT applications (Kim et al., 2012; Smiari and Bibi, 2018). The output of this research question will be a comparison between the quality of the reused and the native components per type of functionality offered in IoT applications.

The answer to this question will help practitioners evaluate the reused components in terms of quality and guide maintenance activities in the long-term by prioritizing the maintenance of the ones with lower quality (compared to the native components). Furthermore, researchers can easily identify components of different functionality types that are in need for applying methods and tools that improve quality.

3.2. Case selection and units of analysis

This section presents the details of the projects that were selected for examining the reuse potentials in IoT applications. In this embedded multiple-case study, the involved contexts are the different IoT functionality types, the cases are the Eclipse IoT projects and the units of analysis are their reused components.

The selected projects are part of the Eclipse IoT framework. Eclipse IoT[2] offers 35 open source projects which implement protocols, services and gateways among others. We addressed reuse opportunities from Eclipse IoT for a variety of reasons:

- Eclipse IoT projects provide all the benefits of an open source project, that are big embracement from the community (Von Krogh et al., 2003) and cost efficiency (Morgan and Finnegan, 2007) which renders it a primary choice when it comes to reusing components for building IoT solutions (Bröring et al., 2017; Shani and Broodney, 2015).
- The initiative of Eclipse IoT is supported by many commercial companies (Bosch, Sierra Wireless, Aloxy, Othermo, IBM and Red Hat) that have adopted these frameworks. This fact shows that this framework is expected to further attract developers to IoT technologies.
- Eclipse IoT projects are consistently used for empirical research the last years in the context of building cloud services (Kovatsch et al., 2014), communication protocols (Tanganelli et al., 2015), enabling solutions for the vehicle domain (Höttger et al., 2018), providing solutions for manufacturing systems (Dorofeev et al., 2017).

In this study we aim to analyze ***components*** of IoT projects that have already been reused by other projects and implement ***the types of functionalities*** presented in Fig. 1. Therefore, in order to retrieve the reused components, we considered the following steps.

(a) Initially tracked all the Eclipse IoT projects as mentioned in the official website of eclipse IoT[2] (35 projects)

(b) Searched for Eclipse IoT projects built with Maven, finding 15 of them. We applied this criterion since it provides an easy mechanism to facilitate reuse between different projects. Moreover, Maven offers detailed reuse information between software components, i.e., the number of times each package has been reused and the name of the component that reuses it.

(c) Examined if the 15 Eclipse IoT projects have been reused (by inspecting the information provided by Maven). From these 15 projects, seven of them have been actually reused. For each component in these seven Eclipse IoT projects, named the "source" components, we recorded:

- The name of the project and the name of the component from Eclipse IoT projects ***that has been reused***, as the "source component" (in total, 503 components from the seven projects were analyzed).
- The name of the project and the name of the component ***that has reused*** components from Eclipse IoT projects, the "target" component.

Therefore, only the Eclipse IoT projects that have been reused by other projects were further analyzed, as we were not able to acquire reuse information for the rest of the Eclipse IoT projects. The seven projects from which we could extract reuse information are: Californium, Ditto, Kura, Leshan, Milo, Paho and

[2] https://iot.eclipse.org/, https://www.eclipse.org/californium/, https://www.eclipse.org/ditto/, https://www.eclipse.org/kura/, https://www.eclipse.org/leshan/, https://projects.eclipse.org/projects/iot.milo, https://www.eclipse.org/paho/, https://www.eclipse.org/smarthome/.

Table 1
Eclipse IoT reused projects.

Project	Description	#Classes	LoC	Releases	# Forks	# Contributors	# Classes reused by other projects
Californium (2015)	Devices & services communication	662	141.118	29	247	49	75
Ditto (2017)	Handles devices heterogeneity	3.361	420.944	14	41	21	25
Kura (2014)	Communication with IoT hardware	1.965	317.578	42	215	38	36
Leshan (2015)	M2M communication	367	48.849	27	260	25	27
Milo (2017)	Data transfer	1.478	184.368	21	165	18	74
Paho (2013)	Communication of IoT applications	344	68.626	14	534	41	26
SmartHome (2014)	Support of IoT smart home devices	2.197	363.735	4	845	178	240

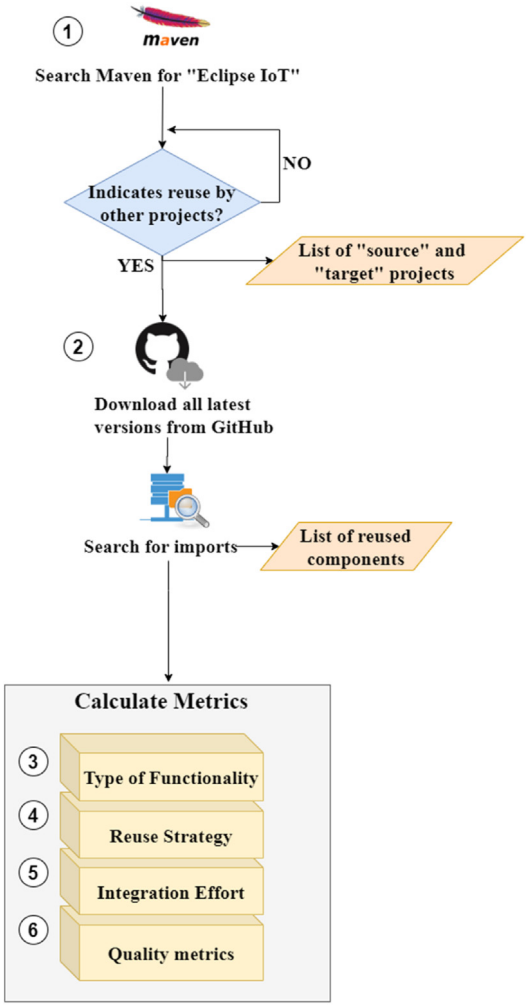


Fig. 2. Case selection process.

SmartHome. The rest of the Eclipse IoT projects did not present any reuse instances and therefore excluded from the analysis.

Table 1 presents a summary of the seven IoT projects whose components are reused by other projects. Appendix presents a summary of the projects that reuse Eclipse IoT.

3.3. Data collection

The dataset used in this study consists of 503 rows, i.e., one for each reused component (a component is considered to be a class). For every component, we recorded 3 sets of metrics: related to Reuse, Effort and Quality. We also synthesized these metrics at the project level in order to answer the three research questions.

Table 2 presents the metrics considered within the scope of this study and their description.

In order to collect the data for this study we followed the process summarized in the next steps, also depicted in Fig. 2.

Step 1: We searched the Maven[3] repository for the seven Eclipse IoT projects that were reused (see Table 1). These projects are considered as the "source" of the reused components. Then we recorded the projects that reused the seven Eclipse IoT projects, as reported in Maven. These projects are considered as the "targets" of the reused components. The output of this step is a list of the *"source"* and *"target"* projects.

Step 2: In this step, we downloaded both *source* and *target* projects from GitHub. In order to retrieve the reused classes, we searched all *"import"* directives in the target projects that included the path to classes of the *source* projects. The "*import*" directives were considered to be reuse indicators of particular classes of the source projects. The output of this step is the list of the reused components (classes) from the source projects.

Step 3: In this step, we classified the reused components based on the ***Type of Functionality*** they offer. The functionality of each component was mapped to one of the seven architectural layers introduced by Fremantle (2015). The process for mapping the functionality of each component was based on the keywords presented in Fig. 1. The keywords were selected based on the words (and synonyms) included in the names of the layers of well-known IoT architectures (Krčo et al., 2014; The Internet of Things Reference Model Whitepaper, 2014; Yelamarthi et al., 2017) and the semantics behind every "source" component to further verify the type of functionality it is meant to implement. For example, the class MqttMessage is part of the implementation of the MQTT messaging protocol offered by the project Paho. Since MQTT is a protocol that relates to communications this class will be categorized as part of the *Communication* layer. The first author of the study parsed the source code of the components and manually classified each reused component to a type of functionality based on the keywords of Fig. 1. In 70% of the cases the classification was easy since the name of the component was very representative of its functionality (and usually included as term or sub-term one of the keywords of Fig. 1). A one-third sample of these cases was inspected by the other authors to ensure consistency. In 30% of the cases the classification output was not clear. These cases were further discussed among the authors for reaching a consensus. In 5% of these cases there was further discussion and disagreements that were resolved by internal author voting.

Step 4: In this step, we classified the ***Reuse Strategy*** into Black-box reuse or White-box reuse (Ravichandran and Rothenberger, 2003). By Black-box reuse we mean an instantiation of a new object from the reused class or the utilization of a parameter or function of a reused class that has not been subject to changes (Ravichandran and Rothenberger, 2003). White-box reuse refers to the extension of the reused class by adding new functionalities or implementing existing definitions (Ravichandran and

[3] https://mvnrepository.com/.

Table 2
IoT source selection metrics.

Type of metric	Metric	Description
Reuse metrics	Development with Reuse (Schwittek and Eicker, 2013)	Shows if a component has been built with reuse (Yes/No)
	Type of Functionality (Fremantle, 2015)	Devices layer Communication layer Aggregation/Bus layer Event Processing & Analytics layer External Communication layer Device Management layer Identity & Access management layer
	Reuse Strategy (Ravichandran and Rothenberger, 2003)	Black-box White-box
Effort metrics	Integration Effort (Gui and Scott, 2006; Prieto-Diaz and Freeman, 1987)	= Added SLOC + modified SLOC + deleted SLOC in the components reused by the "target" projects. If the "source" component is reused more than once then the integration effort is calculated as the average Integration Effort for all the cases that it has been reused.
Quality metrics (Bansiya and Davis, 2002)[a]	Functionality (Bansiya and Davis, 2002)	0.12 * CAM + 0.22 * NOP + 0.22 * CIS + 0.22 * DSC + 0.22 * NOH
	Extendibility (Bansiya and Davis, 2002)	0.5 * ANA-0.5 * DCC + 0.5 * MFA + 0.5 * NOP
	Reusability (Bansiya and Davis, 2002)	−0.25 * DCC + 0.25 * CAM + 0.5 * CIS + 0.5 * DSC
	Flexibility (Bansiya and Davis, 2002)	0.25 * DAM − 0.25 * DCC + 0.5 * MOA + 0.5 * NOP
	Understandability (Bansiya and Davis, 2002)	−0.33 * ANA + 0.33 * DAM − 0.33 * DCC + 0.33 * CAM − 0.33 * NOP − 0.33 * NOM − 0.33 * DSC
	Effectiveness (Bansiya and Davis, 2002)	0.2 * ANA + 0.2 * DAM + 0.2 * MOA + 0.2 * MFA + 0.2 * NOP
Synthesized metrics	Reused components	Total number of components reused per Type of Functionality (Is calculated as the total number of times the different values of Type of Functionality are observed for the components where Development with Reuse = "Yes")
	Native components	Total number of native components per Type of Functionality (Is calculated as the total number of times the different values of Type of Functionality are observed for the components where Development with Reuse = "No")
	Black-box reuse	Total number of components reused in the form of Black-box Reuse per type of functionality (Is calculated as the total number of times the different values of Type of Functionality are observed for the components where Development with Reuse = "Yes" and Reuse strategy = "Black-box")
	White-box reuse	Total number of components reused in the form of White-box Reuse per Type of Functionality (Is calculated as the total number of times the different values of Type of Functionality are observed for the components where Development with Reuse = "Yes" and Reuse Strategy = "White-box")
	Reuse Frequency (Frakes and Terry, 1996)	= E/L E = Number of classes that are reused from external sources, in the target system L = Total number of classes in the target system

[a]Metrics and indices coming from QMOOD model
CAM = Cohesion Among Methods of Class
NOP = Number of Polymorphic Methods
CIS = Class Interface Size
DSC = Design Size in Classes
NOH = Number of Hierarchies,
ANA = Average Number of Ancestors
DCC = Direct Class Coupling
MFA = Measure of Functional Abstraction
DAM = Data Access Metric
NOM = Number of Methods
MOA = Measure of Aggregation (Bansiya and Davis, 2002).

Rothenberger, 2003). To define the type of reuse we analyzed the project's dependency tree to identify the system files (.class) that exist in compiled packages projects that are downloaded from the Maven repository. The identification of the original Eclipse IoT reused components relied on the naming of the system class. We searched each one of these classes in the source code of the "target" projects, and when we identified class extensions in a project, we marked them as White-box reuse. We also searched the target system for the keywords "extends" or "implements" or "override" in methods that could indicate the addition/implementation/modification of a class. The remaining Eclipse IoT classes in the "target" projects were considered to be Black-box reuse (since they were not extended/modified in target source code).

Step 5: Next, in the case of White-box reuse we recorded the ***Integration Effort*** required to customize the reuse component to the target system. The Integration Effort is measured as the source lines of code (SLOC) developed for integrating the reused component. For this purpose, we recorded the SLOC developed in the case of White-box reuse. In particular, we used a diff tool[4] (for comparing source code delta) to assess the number of lines added, modified and deleted between the original component

[4] https://www.jetbrains.com/help/idea/comparing-files-and-folders.html.

and the reused one. In most of the cases of White-box reuse, functionalities were added by either extending a parent class, implementing an interface or overriding a method.

Step 6: As a final step, we recorded the ***Quality*** of both the reused components and the native components, i.e., classes of the target system that were built from scratch. In this step, we analyzed the dependency tree of each "target" project to identify if a class has been reused or it is built from scratch (native). First, we marked as reused all systems classes (.class files) that exist in the compiled packages of the "target" projects that are downloaded from the Maven repository. The identification of the original Eclipse IoT reused components relied on the naming of the system class and the extended classes relied on the process described in Step 4. There were also other native components reused from Maven repository, which did not originate from the Eclipse IoT project that were excluded from the analysis. All other classes of the built projects (i.e., other than reused ones) are tagged as native. The quality of the components was assessed by utilizing the QMOOD model (Bansiya and Davis, 2002) and Percerons[5] tool. In particular, we calculated the following quality indices:

- ***Functionality*** indicates the level at which a class provides operations to other classes.
- ***Extendibility*** indicates the level at which code can be expanded to accommodate new requirements.
- ***Reusability*** indicates the level at which code can be applied to different contexts.
- ***Flexibility*** indicates the level at which code can be altered to be adapted into different contexts.
- ***Understandability*** indicates the level at which code can be comprehended
- ***Effectiveness*** indicates the level at which code can perform specific operations successfully

3.4. Data analysis

The data analysis of this case study includes the calculation of (a) the frequency and descriptive statistics of reuse properties across the different types of IoT functionality and (b) the application of Significance tests for checking whether the differences across reused functionality types are significant. In the case of applying Significance tests we initially performed a Shapiro–Wilk test in order to investigate whether our data are normally distributed. Since the test indicated that our data are not normally distributed, we then used non-parametric tests (Kruskal–Wallis H test and a Mann–Whitney U test) considering the fact that our data set satisfies the following assumptions:

- the dependent variables are either continuous or ordinal
- the independent variables consist of two or more categorical and independent groups.
  - In the case where the groups are only two the Mann–Whitney U test is performed
  - Otherwise the Kruskal–Wallis H test is performed
- we have independence of observations, since our data consist of different projects that reuse different components

For **RQ1**, we provide the descriptive statistics (Mean, Min, Max, St. deviation) and performed a Kruskal–Wallis H test (x^2 (df), p) to check the difference between the level of reuse across different functionality types. The dependent variables consist of the *No of Reused components* and the *No of Native components* whereas the independent variable is the *Type of functionality*. Additionally, we also calculated the corresponding statistics for the components that are native (they are built from scratch) in the "target" projects. Among the variables that are of interest in this question is the Reuse Frequency (see Table 2) that is an indication of the level of reuse in the "target" system.

Concerning **RQ2**, we provide the descriptive statistics (Mean, Min, Max, St. deviation) related to the total number of times where the two different reuse strategies are observed (Black-box reuse, White-box reuse). We also provide the corresponding frequency of the two types of reuse strategy in the form of a bar chart. We then perform a Mann–Whitney U test (U, p) to check whether the difference between the two reuse strategies is significant with respect to the functionality type of the reused component. The dependent variable is associated with the *Type of Functionality* while the independent variable consists of the *Black-box* and *White-box* type of reuse. Additionally, in the case where White-box reuse is observed we also provide the related descriptive statistics (Mean, Min, Max, St. deviation) regarding the Integration Effort required to customize the component to the "target" class and perform a Kruskal–Wallis H test (x^2 (df), p) to check the difference between the effort across different functionality types. In this case, the dependent variable is attributed as the *Integration Effort* whereas the independent variable is considered to be the *Type of Functionality*.

Similarly, in **RQ3**, we provide the descriptive statistics (Mean, Min, Max, St. deviation) related to the quality attributes (Extendibility, Flexibility, Reusability, Functionality, Understandability, Effectiveness) of both the reused components and the native ones of the "target" projects. Then we perform a Mann–Whitney U test (U, p) to check whether the difference between the quality in the reuse and the native components is significant. The dependent variables refer to the Reusability, Functionality, Flexibility, Extendibility, Understandability and Effectiveness while the independent variable consists of the *Reused* and the *Native components*.

In Table 3 we provide an overview of the data analysis methods employed per research question.

3.5. Data overview

In this section, we present an overview of the data used in the study. First, we categorized the data into reused and native components. Next, we calculated the descriptive statistics that give us an overview of the data as presented in detail in Table 4 and Fig. 3. When focusing in *White-box reuse* the average integration effort that is required is 214.18 lines of code. Additionally, the Reusability appeared to have a higher average value in the reused components compared to the other metrics, whereas Understandability appeared to have the lowest. As for the native components, Flexibility, Extendibility and Understandability appear to have higher mean values compared to the reused components.

The total components from the "target" projects that were found in each *Type of Functionality* are presented in Fig. 3 and each bar is split into the total percentage of native and reused components that were found in each type respectively. Although in total multiple components target the *External Communication* layer (27.1%), the *Communication* layer (22.6%) and the *Event Processing & Analytics* layer (22.4%), we observed that the *Devices* and *Device Management* layer appear to be reused more than the rest of the layers, with 70% and 11% of the components being reused respectively. This finding is intuitive since, when referring to the Reused components per *Type of Functionality* (see Fig. 3), selecting components that will enable building and handling diverse devices without the need to rewrite functionality is of high

[5] https://extreme.se.uom.gr/.

Table 3
Data analysis overview.

RQ	Variable	Analysis	Calculations
RQ1	Dependent variables: ***No of Reused components, No of native components***	Descriptive statistics	Mean, Min, Max, Std. Dev, Reuse frequency
	Independent variable: ***Type of functionality***	Kruskal–Wallis H Test	$x^2\,(df)\,,\ p$
RQ2.1	Dependent variables: ***Type of functionality***	Descriptive statistics	Mean, Min, Max, Std. Dev
	Independent variable: ***Black-box reuse, White-box reuse***	Bar Chart Mann–Whitney U Test	Frequencies U, p
RQ2.2	Dependent variables: ***Integration effort***	Descriptive statistics	Integration effort, Min, Max, Std. Dev
	Independent variable: ***Type of functionality***	Kruskal–Wallis H test	$x^2\,(df)\,,\ p$
RQ3	Dependent variables: ***Reusability, Functionality, Flexibility, Extendibility, Understandability, Effectiveness***	Descriptive statistics	Mean, Min, Max, Std. Dev
	Independent variable: ***Reused components, Native components***	Mann–Whitney U test	U, p

Table 4
Data overview.

	Reused								Native			
	White box				Black box							
	Min	Max	Mean	St. Dev	Min	Max	Mean	St. Dev	Min	Max	Mean	St. Dev
Integration effort	27	1421	214.18	203.27	–	–	–	–	–	–	–	–
Reusability	0.25	90.73	13.23	12.99	0.25	67.78	8.75	10.10	0.25	422.79	3.98	9.64
Flexibility	−0.25	1.00	0.27	0.16	−0.25	2.24	0.32	0.39	−0.25	114.75	0.60	1.84
Functionality	0.32	42.07	6.46	6.31	0.1	36.35	4.17	5.00	0.10	209.68	1.90	4.75
Extendibility	−0.32	1.51	0.49	0.36	−0.54	3.5	0.43	0.49	−25.06	116.5	0.89	2.15
Understandability	−61.71	−0.33	−9.30	8.68	−45.32	−0.33	−5.25	6.20	−157.29	0.00	−3.60	6.01
Effectiveness	−0.20	0.87	0.55	0.15	−0.2	2.0	0.41	0.32	−0.20	46.40	0.52	0.89

importance in IoT projects and thus, the *Devices* (70%) and the *Device Management* layers (11%) appear to have more instances.

Furthermore, the *Devices* layer appears to be the lowest in total components, which is understandable since this category refers to lower level devices and sensors and it would not be a point of interest in some of the projects. However, there are more reused components (70%) than native components (30%). On the contrary, the *Communication*, the *Event Processing & Analytics* and the *External Communication* layers appear to have the lowest amount of reused components (2%) compared to the native ones, which suggests that these components are much more often rewritten.

4. Results

In this section, we present and interpret the results of this case study, organized by research question and based on the data analysis presented in Section 3.4.

4.1. RQ1 – Which types of functionality offer the most components in the context of IoT application development?

To address this research question, we provide the descriptive statistics of the reused components per type of functionality (see Table 5) and examine the reuse potentials, per type of implemented functionality, of the reused components. Table 5 presents the summary statistics for the seven types of functionality offered by the Eclipse IoT projects. It can be observed that most reused components (see Mean — column 2) offer functionality related to the *Device Management* layer. Such functionality includes standard operations for handling IoT devices, e.g., configuration, remote management, provisioning, maintenance and monitoring. External Communication components are also reused frequently, offering functionality related to the communication of IoT devices with the applications that handle the derived data, e.g., wrappers, APIs and clients.

In terms of highest reuse frequency (see the rightmost column in Table 5), we notice that *Device* layer is the most recurrent type. The components that implement functionality related to this layer usually represent the different kinds of hardware equipment that can be used in the context of IoT, from the more abstract form ("thing") to the more specialized form (e.g., sensors, actuators and mobile devices). Moreover, we find components that represent device properties like operational range, lifespan, etc. Overall, it seems that practitioners seek to reuse high-level IoT functionality such as the representation and management of devices. This can be interpreted intuitively, as these functionalities are context-free and are more likely to be reused (Dorofeev et al., 2017). Therefore, we summarize that device functionalities share commonalities that can be reused and additionally present the same management requirements.

The least reused components are related to the *Identity & Access Management* layer and to the *Communication* layer. We observe that practitioners tend to reuse fewer of such components probably because they are considered to be safety-critical for IoT applications that need to avoid third-party attacks (Kim et al., 2012), ensuring the smooth operation of the devices. Moreover, components that implement context-aware functionalities (e.g., communications and aggregation) are highly dependent on the protocol that a device (and its manufacturer) can support and, thus, are less reused. Also, such functionalities are commonly implemented from scratch (Ciccozzi and Spalazzese, 2016), as they need to be configured based on the specific requirements and deployment details of a particular IoT application and its restrictions in terms of hardware and communication protocols. To investigate if the aforementioned differences are statistically significant, we performed a Kruskal–Wallis H test, which suggested

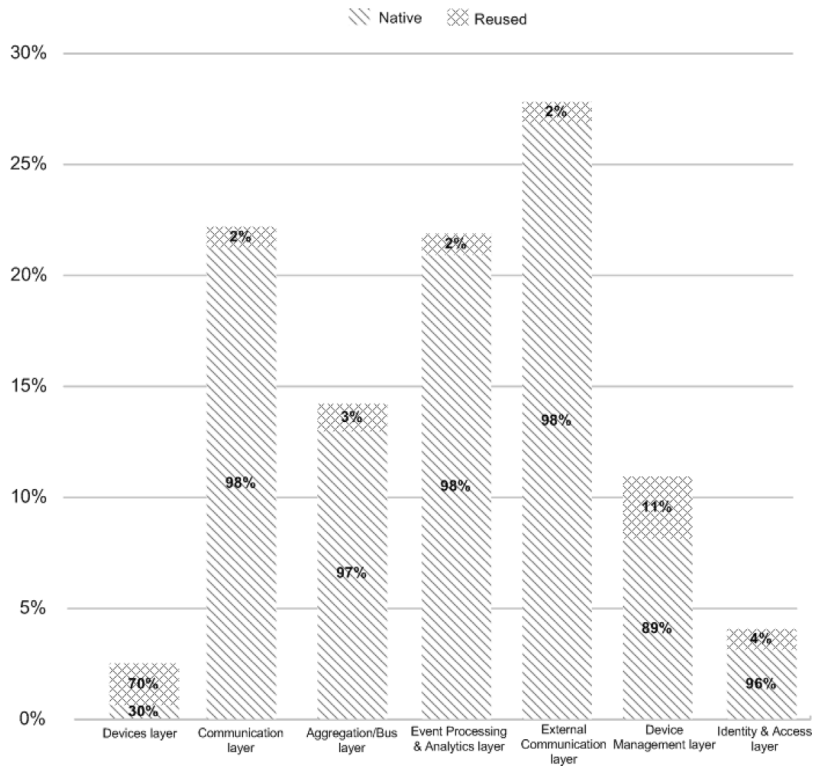


Fig. 3. Type of functionality of reused and native components bar chart.

Table 5
Reuse per type of functionality.

Type of functionality	Reused components				Native components				Reuse frequency
	Mean	Min	Max	St. Dev	Mean	Min	Max	St. Dev	
Devices	6.81	1	45	8.63	7.66	1	18	9.07	0.88
Communication	4.43	1	27	4.27	66.63	1	1315	160.54	0.06
Aggregation/Bus	5.16	1	20	4.21	64.57	1	1203	142.33	0.08
Event Processing & Analytics	5.05	1	39	5.16	63.49	1	2897	266.96	0.08
External Communication	7.07	1	98	11.4	48.75	1	1130	96.47	0.15
Device Management	9.31	1	93	10.3	27.89	2	181	27.33	0.33
Identity & Access Management	2.95	1	9	2.14	31.50	1	190	42.02	0.09
Kruskal–Wallis H test	x^2 (6) =		*38.59*			p =		*<0.001*	

that there are significant differences in the reusable components of the different types of functionality (x^2 (6) = 38.59, p = <0.001).

4.2. RQ2 – Do the reused software components require customization?

In this section, we first present and interpret the results regarding the reuse strategy that is mostly adopted, differentiating between Black-box reuse and White-box reuse. Next, we focus on White-box reuse and examine the integration effort required to customize the reused component to the target class.

4.2.1. RQ2.1 - Which reuse strategy (i.e., White-box reuse or Black-box reuse) is adopted when integrating the reused IoT components to the target application?

In Fig. 4, we present the type of Reuse Strategy employed per type of functionality. The results suggest that most of the components are integrated in the "target" project in the form of Black-box reuse (72.6%). As we can observe in Fig. 4 and Table 6, the *Device Management* components are, on average, mostly reused "as is" (i.e., Black-box reuse). Moreover, White-box reuse was required in 27.4% of the components. To investigate if the aforementioned differences are statistically significant, we performed a Mann–Whitney U test, which suggested that there are significant differences in the different types of reuse in terms of the different types of functionality for all the types of functionality.

Black-box reuse is the dominant form in third-party IoT components (i.e., not developed in-house), which can be interpreted in two ways. On the one hand, practitioners may not be willing to devote effort and time to understand the reused components and, therefore, they select components that can be reused as is. This interpretation is in accordance with related work that also appoints the dominance of Black-Box reuse in the case of reusing third-party, open source software components (Haefliger et al., 2008). On the other hand, the reused components offer core functionality (i.e., *Device Manage*ment layer), which is the least likely to require changes. For example, general purpose components like the ones representing devices, communication protocols, management operations or standard cryptography algorithms are the least likely to require changes.

4.2.2. RQ2.2 - What is the customization effort required to integrate the reused components in the case of White-box reuse?

To investigate the effort required for the integration of the components with respect to the different types of reused functionality, we calculated the descriptive statistics and performed a

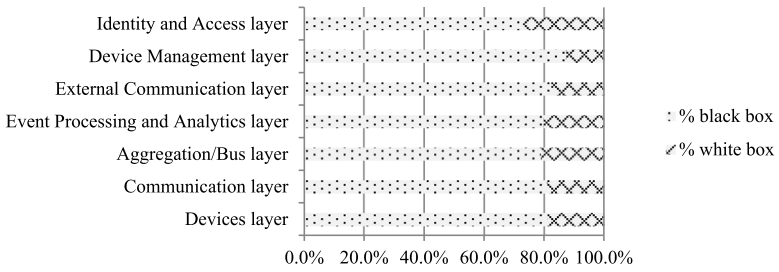


Fig. 4. Reuse strategy per type of functionality.

Table 6
Type of reuse per type of functionality.

Type of functionality	Black-box reuse				White-box reuse			
	Mean	Min	Max	St.Dev	Mean	Min	Max	St.Dev
Devices	6.67	1	45	8.26	1.55	1	4	1.13
Mann–Whitney U test	$U =$	69.500			$p =$	0.001		
Communication	4.34	1	26	4.15	1.02	1	2	0.16
Mann–Whitney U test	$U =$	871.000			$p =$	<0.001		
Aggregation/Bus	4.72	1	19	3.89	1.26	1	3	0.51
Mann–Whitney U test	$U =$	2718.000			$p =$	<0.001		
Event processing & Analytics	4.66	1	38	4.75	1.18	1	4	0.51
Mann–Whitney U test	$U =$	1927.500			$p =$	<0.001		
External Communication	6.70	1	96	11.07	1.43	1	5	0.82
Mann–Whitney U test	$U =$	1671.500			$p =$	<0.001		
Device management	8.71	1	91	9.90	1.27	1	5	0.60
Mann–Whitney U test	$U =$	10 610.500			$p =$	<0.001		
Identity & Access management	2.70	1	9	2.15	1	1	1	0.00
Mann–Whitney U test	$U =$	12.000			$p =$	0.012		

Kruskal–Wallis H test for the variable *Integration Effort*, but only for the cases where the reuse type is White-box. The summarized results are presented in Table 7.

We observed that the type of functionality that requires the greatest effort on average, in the case of White-box reuse, regards the *Device Management* layer. This result can be interpreted intuitively since components in this layer are responsible for supporting specific smart devices. As IoT frameworks focus on accommodating a substantial range of devices (Zanella et al., 2014), it is expected that such reusable components (from the framework) are quite generic and would require a significant amount of code to fully characterize the smart devices.

Additionally, we found that components implementing functionality related to the *Identity & Access Management* layer are the ones that required the least integration effort. This finding suggests that the implemented standards are most often sufficient (e.g., in terms of reliability and security) for the target applications, which is understandable as the authorization standards and protocols are becoming widespread across various industries and application domains (Leiba, 2012), also enabling more complete off-the-shelf components.

4.3. RQ3 – Do reused components present higher quality compared to the native components in the context of IoT application development?

In Table 8 we present the results regarding the quality of the reused components. For this research question, we examined the offered functionalities by assessing the six quality indicators defined in the QMOOD model (Bansiya and Davis, 2002), i.e., Reusability, Flexibility, Functionality, Extendibility, Understandability and Effectiveness. Overall, it is observed that reused components present higher quality in terms of Reusability and Functionality. This is expected since the reused components come from the Eclipse IoT development framework that is built to facilitate reuse and offer abstractions that will help developers adopt functionalities that are typical to the particular application domain such (Kovatsch et al., 2014; Tanganelli et al., 2015).

Table 7
Effort required to integrate the reused components per type of functionality.

Type of functionality	Integration effort	Std. Dev	Min	Max
Devices	159.42	111.74	60.00	394.00
Communication	192.84	178.17	27.00	944.00
Aggregation/Bus	162.53	100.29	27.00	781.00
Event Processing & Analytics	127.86	91.86	44.00	553.00
External Communication	223.53	214.06	27.00	968.00
Device Management	264.26	244.76	42.00	1421.00
Identity & Access Management	83.00	25.85	61.00	119.00
Kruskal–Wallis H test	x^2 (6) =	41.826	$p =$	<0.001

In terms of Flexibility, Extendibility and Effectiveness, the reused components present similar values to native components, suggesting that reusing IoT components does not jeopardize the quality of the target system. This result is also verified by Feitosa et al. (2020) who argue that reused components present similar maintainability compared to native ones. In terms of Understandability, the reused components appear to more comprised than native ones. This result is expected since the metric used for the calculation of Understandability considers that inheritance and polymorphism have a negative effect on Understandability (Bansiya and Davis, 2002). Since the components under study support the representation of abstractions to a great degree, the reusable code is considered to be less understandable. However, we argue that this result can be counterbalanced by the fact that the reusable components, and the projects hosting them, usually provide a very detailed documentation that is considered to affect implicitly the Understandability (Paschali et al., 2017).

Table 8
Quality of the eclipse IoT projects and destination components per type of functionality offered.

Layer		Reusability		Flexibility		Functionality		Extendibility		Understandability		Effectiveness	
		Reused	Native	Reused	Native	Reused	Native	Reused	Native	Reused	Native	Reused	Native
Devices	Mean	6.2	2.0	0.2	0.08	2.9	0.9	1.8	0.5	−4.3	−1.8	0.5	0.1
	Min	3.5	0.8	0.2	0.0	1.6	0.3	0.5	0.0	−4.6	−2.6	0.4	0.0
	Max	6.8	3.6	0.3	0.3	3.2	1.6	0.9	1.5	−3.2	−1.0	0.6	0.2
	St.dev	1.2	1.4	0.01	0.1	0.6	0.6	0.2	0.9	0.6	1.1	0.05	0.1
	$U =$	1.000		5.500		1.000		10.000		0.000		0.000	
	$p =$	0.007		0.029		0.007		0.334		0.011		0.011	
Communication	Mean	6.1	4.4	0.6	0.6	2.8	2.1	0.3	1.2	−3.8	−4.2	0.3	0.7
	Min	0.3	0.3	−0.3	−0.3	0.1	0.1	0.0	−25.1	−17.5	−153.5	−0.2	−0.2
	Max	34.0	138.0	2.2	114.8	17.2	101.4	3.0	116.5	−0.3	0.0	2.0	46.4
	St.dev	7.3	7.9	0.5	2.8	3.6	4.0	0.6	3.1	3.8	6.7	0.4	1.3
	$U =$	76 908.500		100 500.500		81 414.500		74 121.000		91 997.000		75 676.000	
	$p =$	<0.001		0.629		0.001		<0.001		0.704		0.003	
Aggregation/ Bus	Mean	9.2	4.5	0.4	0.4	4.5	2.3	0.4	1.5	−5.7	−4.3	0.5	0.7
	Min	0.3	0.3	−0.3	−0.2	0.3	0.1	0.0	−0.5	−26.7	−51.2	−0.2	−0.2
	Max	39.9	282.0	2.2	36.2	19.4	154.7	3.5	36.5	−0.3	0.0	1.6	14.4
	St.dev	10.5	10.8	0.4	2.4	5.3	5.8	0.6	2.8	5.9	5.4	0.3	1.2
	$U =$	1210.500		2481.500		1230.500		3201.000		17 767.000		21 522.500	
	$p =$	<0.001		0.035		0.001		0.205		0.006		0.345	
Event Processing & Analytics	Mean	9.8	3.4	0.3	0.5	4.7	1.6	0.5	1.1	−4.0	−3.6	0.4	0.6
	Min	0.3	0.3	−0.3	−0.3	0.1	0.1	0.0	0.0	−14.8	−58.7	−0.2	−0.2
	Max	34.0	75.5	2.2	11.3	17.2	34.5	2.0	11.5	−0.3	0.0	1.4	6.2
	St.dev	9.1	5.4	0.4	0.8	4.5	2.4	0.4	1.4	3.3	4.1	0.3	0.6
	$U =$	34 767.500		63 182.500		34 580.500		63 405.500		47 513.000		47 701.500	
	$p =$	<0.001		0.029		<0.001		0.029		0.135		0.143	
External Communication	Mean	7.6	4.0	0.2	0.6	3.6	1.9	0.3	0.5	−4.4	−3.2	0.3	0.4
	Min	0.3	0.3	−0.3	−0.3	0.1	0.1	−0.3	−0.6	−45.3	−157.3	−0.2	−0.2
	Max	67.8	422.8	1.3	38.8	36.4	209.7	1.5	40.0	−0.3	0.0	0.6	15.8
	St.dev	12.1	11.7	0.4	1.4	6.1	5.6	0.4	1.6	8.5	6.5	0.2	0.7
	$U =$	103 420.000		114 031.000		102 754.000		140 076.500		99 368.000		105 025.000	
	$p =$	<0.001		<0.001		<0.001		0.105		0.137		0.350	
Device Management	Mean	14.4	3.3	0.3	0.1	7.0	1.5	0.6	0.3	−10.0	−4.8	0.6	0.2
	Min	1.1	0.3	−0.2	−0.3	0.6	0.1	−0.5	0.0	−61.7	−12.6	0.4	−0.2
	Max	90.7	12.0	1.0	0.8	42.1	5.3	1.5	1.5	−1.7	−1.0	0.9	0.6
	St.dev	13.3	2.7	0.1	0.3	6.4	1.2	0.3	0.4	8.8	4.0	0.1	0.2
	$U =$	299.000		721.500		286.000		723.000		605.000		94.500	
	$p =$	<0.001		0.001		<0.001		0.001		0.001		<0.001	
Identity & Access	Mean	7.0	3.5	0.5	0.5	3.2	1.6	0.7	0.5	−5.0	−3.3	0.5	0.4
	Min	1.1	0.3	−0.3	−0.3	0.5	0.1	0.0	0.0	−16.8	−31.7	−0.2	−0.2
	Max	23.0	31.0	2.2	3.3	10.6	13.7	1.0	4.0	−0.3	0.0	2.0	2.0
	St.dev	6.2	4.9	0.5	0.5	2.9	2.2	0.4	0.7	4.4	4.2	0.5	0.3
	$U =$	1515.500		2563.500		1652.000		3240.000		2108.500		2899.500	
	$p =$	<0.001		0.017		<0.001		0.641		0.018		0.911	

5. Discussion

In this section, we illustrate the relevance of our findings through an example use case involving a reuse decision making process in the context of an IoT application development. Additionally, we interpret the results obtained by this case study and elaborate on implications to both researchers and practitioners.

5.1. Applicability of empirical findings

In Section 1, we described three major tasks of the reuse process, i.e., (a) identify the reusable asset, (b) integrate the reusable asset and (c) evaluate the reusable asset. In this section, we elaborate on an example decision making process, presenting the data that would guide it. To that end, in Table 9 we rank the different functionality types for each metric considered in this study, which are grouped according to the three major tasks. For example, when identifying reusable assets (wondering "what to reuse?"), one may look into the reuse frequency metric (column *RF*) and check the top three reused functionalities, finding that components related to the *Devices*, the *Device Management* and the *External Communications* layers are more commonly reused. To help the reader further, we provide examples of popular components on the right side of Table 9.

In our example, the reuser intends to develop a small-scale Smart Home application that will manage a set of smart lamps manufactured by different companies. The goal is to create a holistic solution exploiting the existing reusable components that are freely available, which will be configured to meet the needs of a Smart Home.

Based on Table 9 and the architecture model of Fig. 1, the developer should start by reusing components from the *Devices* layer. The *Devices* layer implements the lower level components of the architecture representing the devices around which the developer will build a centralized solution. These components present the highest reuse frequency and also possess the highest Extendibility. Additional components implementing *Device* functionality may need extra coding since these components are often integrated via White-Box reuse in the target system. However, the expected necessary effort is not that high.

The next step is to implement functionality related to the management of the devices (such as turn the lights on/off, change their color, or update their firmware) by reusing components

Table 9
IoT component reuse ranking.

Functionality	Task | Relevant question | Metrics											Example of popular reused component
	Identification What to reuse?		Integration Does it need customization?			Evaluation What is their quality?						
	RF	#Reused components	White-Box	Black-Box	Integration effort	Reusability	Flexibility	Functionality	Extendibility	Understandability	Effectiveness	
Devices	**1**	**3**	**1**	**3**	**3**	6	7	6	**1**	**2**	**2**	Thing
Communication	7	6	6	4	5	7	**1**	7	6	**1**	6	NetworkService
Aggregation/Bus	6	4	4	6	4	**3**	**3**	**3**	5	6	**3**	MQTTActionListener
Event Processing & Analytics	5	5	5	5	**2**	**2**	4	**2**	4	4	5	MemoryPersistence
External Communication	**3**	**2**	**2**	**2**	6	4	6	4	7	**3**	7	ChartProvider
Device Management	**2**	**1**	**3**	**1**	7	**1**	5	**1**	**3**	7	**1**	OnOffStatus
Identity & Access Management	4	7	7	7	**1**	5	**2**	5	**2**	5	4	AuthorizationModelFactory

from the *Device Management* layer. These components are top-ranked in terms of Reusability, Functionality, Effectiveness and reuse frequency, making them the perfect candidates to manage all the functionalities performed by the IoT devices. Moreover, most of these components are integrated via Black-box reuse. However, if additional code is required (i.e., White-box reuse), the developer should be aware that this integration can be costly since they depicted the maximum effort.

In order to facilitate the *Communication* of the smart lamps, some basic components can be reused for the protocol that each device supports. These components are highly ranked in terms of Flexibility and Understandability, allowing for the implementation of a uniform communication solution that will accommodate the diversity of protocols successfully. However, these components are not reused frequently and, thus, the reuser may need to search carefully to identify the appropriate components for reuse. Furthermore, a special focus is given to reusing *Aggregation/Bus* components since they are responsible for bridging the different devices. The smart lamps are from different manufacturers and, therefore, we need to write extra native code to provide a more uniform solution. Fortunately, these components appear to be highly ranked for most quality attributes.

The next step is to handle all the data retrieved from our smart lamps (e.g., energy consumption). Components related to *Event Processing & Analytics*, such as general purpose data storage and retrieval procedures, can be reused in our example. However, since we want to satisfy specific use cases like identifying peaks of energy consumption, we would choose to expand these components by writing our own algorithms and automations from scratch. The same goes for the *Identity & Access Management* layer, in which we reuse components for already existing solutions (e.g., authentication), but we also expand them by writing native code in order to have an application that will be configured based on our specific needs (e.g., customized security).

In the end, we structure the entire solution by reusing components related to the *External Communication* layer, providing functionalities related to data visualization, such as charts. We also need to implement native code, as we want to satisfy certain use cases that cannot be fulfilled through reuse, such as combining UI components to create a better user experience. This layer appears to have quite high Understandability, which is what we are looking for when creating a system that can be easily configured to meet the users' needs.

5.2. Implications to researchers and practitioners

The results of this study provide useful information and guidance to *practitioners* on planning the reuse of components in the context of IoT application development. In particular, some take away messages that we can provide based on our case study are:

- When identifying reuse opportunities, Engineers of IoT can greatly benefit from reusing a core set of general purpose components, which in our case regard the *Event Processing & Analytics* layer and the *Devices* layer. These components, in their majority, can be reused as is without requiring any integration effort.
- Regarding the integration of reused components, we highlight that although components from the *Device Management* layer are heavily reused (and necessary), they may require a substantial integration effort, which should be factored in the design and implementation phased of an IoT application.
- Regarding the evaluation of reusable assets, we mainly found that reuse in the context of IoT development seems not to jeopardize the overall quality of the IoT system while also improving its reusability and functionality.

Based on the results of this case study, we encourage researchers to:

- Further explore the reuse of components in the context of IoT application development by examining other Open Source projects. Researchers can investigate whether the same type of components, as appointed by this study, have been systematically reused.
- Introduce a process for systematic, planned reuse of IoT components. Such a process would define clear procedures for: (a) identifying and sorting the reusable components, (b) integrating the reused components into the new applications and (c) maintaining these components.

6. Threats to validity

To conclude this section, we refer to the threats to validity of this case study divided by construct, internal, reliability and external aspects (Runeson and Höst, 2009). Construct validity defines how effectively a test or experiment measures up to its

Table A.1
Projects that reuse Eclipse IoT projects.

Project	Description	#Classes	LoC	Releases	# Forks	# Contributors	# Reused classes
Apache Camel	Integration systems framework	18.461	2.176.350	158	4.000	605	49
Eclipse Hono	IoT devices connector service	648	119.809	43	94	32	10
Eclipse Leshan	M2M communication	367	48.849	27	260	25	67
Mule CoAP connector	CoAP capable applications	13	3.001	1	2	1	5
Rhiot	IoT messaging platform	443	39.398	5	24	14	44
IoTDM	Middleware for IoT data management	245	40.801	20	2	17	15
PerfCake	Performance testing framework	273	48.736	24	31	19	2
GSN	Middleware for sensor networks	443	86.804	13	41	11	1
SiteWhere	IoT platform	1.195	150.579	39	298	13	4
Bosch IoT Things	Examples for Bosch IoT Things service	21	4.101	1	34	22	12
Eclipse Kura Addons	Communication addons for Eclipse Kura	26	3.703	16	3	2	10
Dentrassi Camel	Component for providing OPC UA client and server	36	4.266	2	5	2	18
DFKI COS BaSys Platform	Service platform	1.142	265.678	6	0	5	3
PLC4J Driver OPC UA	Set of libraries that enables communication with PLC	866	90.312	16	88	37	2
Apache ActiveMQ	High performance message broker	4.650	913.479	66	1.200	91	3
Apache Stratos	PaaS framework	1.438	189.727	53	111	48	4
Spring Integration	Support for Enterprise Integration Patterns	2.950	405.617	204	831	141	10
WSO2 Axis2 Transports	Framework for axis2 based transports	361	43.481	62	144	77	7
WSO2 MB	WSO2 message broker	292	49.481	26	74	29	40
MQTT Notification Plugin	MQTT Notification Plugin for Jenkins	2	380	0	14	4	1
Applozic Android SDK	Android Chat SDK	327	64.660	66	304	19	2
AWS AppSync SDK	Android AWS AppSync SDK	220	45.786	34	41	17	2
CA Mobile API Gateway	Android CA Mobile API Gateway SDK	531	97.305	24	17	15	3
AirMap	Android Airspace service for drones SDK	145	24.497	25	5	3	1
Labstack Android	Android library for labstack platform	4	81	0	0	1	2
TNT4J	Track and trace API	164	31.107	217	12	7	2
Qiscus SDK Android	Android Chat SDK	176	30.791	177	75	13	1
Ibis Adapter Framework	Stateless integration framework	1.108	198.028	204	41	27	3
Vert.x MQTT	MQTT server and client	58	9.401	34	56	15	12
Moquette	MQTT lightweight broker	130	14.917	14	636	35	9
Eclipse Kapua	Platform for managing IoT gateways	2.733	247.650	28	138	27	11
Joynr	Web-based communication framework	1.232	146.817	142	33	17	3
FROST-Server	Server for OGC Sensor Things API	464	64.117	20	31	11	4
MQTT	Load testing for MQTT broker	25	3.685	6	1	7	7
OpenHab2	Home automation platform	3.384	394.788	90	2600	535	787

claims. Internal validity is related to the examination of causal relations examining whether an experimental environment is adequate to support the claim. External validity examines whether the results of a study can be generalized to other cases. Reliability is associated to the reproducibility of the study, i.e., the ability of other researchers to repeat the same process, collect data and reach the same results.

A possible threat to ***construct validity*** is related to the metrics that are used to answer our research questions. Since, to our knowledge, there are not any studies available in literature examining the reuse potentials from IoT development frameworks, we selected the metrics based on general purpose studies examining software reuse metrics (Frakes and Terry, 1996), models (Schwittek and Eicker, 2013) and strategies, Ravichandran and Rothenberger (2003). Each metric selected in this study is based on the relevant reference from existing literature. With respect to the synthesized metrics (Reused components, Native components, Black-box reuse, White-box reuse) we selected to use these metrics in order to be able to map the reused components to the type of IoT functionality that they serve. Regarding the effort metrics, we believe that the lines of code are indicators of the effort required to integrate the reused components. This metric has been also adopted in Gui and Scott (2006) and Prieto-Diaz and Freeman (1987) for assessing the reuse effort. Moreover, since the scope of the study is to compare customization effort between reusable components that offer different functionalities and not to provide a typical reuse effort estimation model, we believe that this metric is an indicative for comparing the difference between the customization effort of reusable components. Nevertheless, we acknowledge that there are other metrics that can also be used. Concerning the type of functionality, we based our categorization on the reference architecture proposed by Fremantle (2015). The specific architecture was chosen (a) considering its high adoption by other research practitioners in literature (Neagu et al., 2016; Pessoa and Duarte-Figueiredo, 2017; Zamfir et al., 2016) and (b) considering its high coverage in IoT architecture aspects (Krčo et al., 2014; The Internet of Things Reference Model Whitepaper, 2014; Yelamarthi et al., 2017). We acknowledge though, that the choice of the reference architecture model may alter the observed results. For the quality assessment of the reused components, we have used QMOOD model (Bansiya and Davis, 2002), which is an established quality model that has been rigorously validated and reused in literature (Ani et al., 2017; Couto et al., 2018; O'Keeffe and Cinnéide, 2006). However, we acknowledge that other quality models could lead to variations in the observed results.

Regarding ***internal validity***, the proposed study attempted to form an association between (a) the reuse strategy and the type of functionality implemented by the reused components and (b) the quality characteristics (such as flexibility, reusability, functionality, extendibility, understandability and effectiveness) and the type of functionality implemented by the reused components. We cannot claim that this association forms a causal relationship between the type of functionality of the reused components and the reuse strategy adopted or the quality of the components. The results just indicate trends and common practice when it comes to selecting and integrating components in the context of IoT application development. Some of these trends can be verified intuitively while others may be surprising.

With regard to ***reliability***, we believe that the followed research process ensures the replication of our study. The process

that has been followed in this study has been thoroughly documented in the case study design, provided in Section 3. In addition, the extraction of the data and the associated structural metrics was performed with the help of the publicly available tools (Maven repository, Github, Percerons) and therefore any interested researcher can repeat the analysis and derive the same results. We acknowledge though potential researchers' bias during the data collection due to the manual classification of components into types of functionality performed by the first author. The first author of the study parsed the source code of the components and classified each reused component to a type of functionality based on the keywords of Fig. 1. In most of the cases the classification was easy since the name of the components clearly indicated its function. For these case, one-third of the sample were inspected by the other authors to ensure consistency. Additionally, the classification was not straightforward in 30% of the cases, in which scenario all authors discussed to reach consensus.

Finally, we acknowledge that the ***external validity***, is threatened by the fact that the data set examines a subset of IoT reuse opportunities coming from the 7 projects supported by Eclipse IoT framework. Since we wanted to measure actual reuse opportunities, as observed in practice with the help of Maven repository statistics, we were forced to select only frameworks whose components were reused by other projects. In Maven Repository we were able to retrieve other reused frameworks, such as Microsoft Azure IoT, Amazon SDKs but we made the choice not to analyze frameworks that are platform specific, compromising the interoperability of the applications. Therefore, concerning the generalizability, we can say that a replication of the analysis performed in this study to other IoT frameworks would be useful for further generalizing the results regarding reuse opportunities in the IoT development context.

7. Conclusions

In this paper, we explored the reuse opportunities stemming from 7 popular projects coming from the Eclipse IoT framework for building IoT applications. We performed a case study and investigated (a) the types of functionalities that can be reused; (b) the reuse strategy that is adopted and the effort required for integrating the reused components with respect to the type of functionality that they implement and (c) the quality of the reused components with respect to the type of functionality that they implement. We analyzed 503 reused components integrated in 35 target IoT applications. The results of this case study suggest that: the main reused functionality is related to the *Device Management* layer; the main reuse strategy is Black-box reuse; and the effort for integrating the reused components can range from 27 lines of code to 1421 lines of code. The quality of the reused components is slightly higher, in terms of *Reusability* and *Functionality*, compared to components built from scratch in most cases. As future work we intend to further explore reuse opportunities within IoT frameworks by examining other open source frameworks, retrieving candidate components and comparing them.

CRediT authorship contribution statement

Paraskevi Smiari: Conceptualization, Methodology, Software, Formal analysis, Data curation, Writing - original draft, Writing - review & editing. **Stamatia Bibi:** Conceptualization, Methodology, Software, Writing - original draft, Writing - review & editing. **Daniel Feitosa:** Conceptualization, Methodology, Writing - original draft, Writing - review & editing.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgment

This research was co-funded by the European Union and Greek national funds through the Operational Program Competitiveness, Entrepreneurship, and Innovation, grant number T1EDK-04873

Appendix. Projects that reuse Eclipse IoT projects

See Table A.1.

References

Aly, M., Khomh, F., Yacout, S., 2018. Kubernetes or OpenShift? Which Technology Best Suits Eclipse Hono IoT Deployments. In: 2018 IEEE 11th Conference on Service-Oriented Computing and Applications (SOCA). IEEE.

Ampatzoglou, A., Bibi, S., Chatzigeorgiou, A., Avgeriou, P., Stamelos, I., 2018. Reusability index: A measure for assessing software assets reusability. In: International Conference on Software Reuse. Springer, Cham.

Ampatzoglou, A., Kritikos, A., Kakarontzas, G., Stamelos, I., 2011. An empirical investigation on the reusability of design patterns and software packages. J. Syst. Softw. 84 (12), 2265–2283.

Ani, Z.C., Basri, S., Sarlan, A., 2017. A reusability assessment of UCP-based effort estimation framework using object-oriented approach. J. Telecommun. Electron. Comput. Eng. (JTEC) 9 (3–5), 111–114.

Arvanitou, E.M., Ampatzoglou, A., Chatzigeorgiou, A., Avgeriou, P., 2016. Software metrics fluctuation: a property for assisting the metric selection process. Inf. Softw. Technol. 72, 110–124.

Bansiya, J., Davis, C.G., 2002. A hierarchical model for object-oriented design quality assessment. IEEE Trans. Softw. Eng. 28 (1), 4–17.

Bibi, S., Stamelos, I., Gerolimos, G., Kollias, V., 2010. BBN based approach for improving the software development process of an SME—a case study. J. Softw. Maint. Evol.: Res. Pract. 22 (2).

Bröring, A., Schmid, S., Schindhelm, C.K., Khelil, A., Kaebisch, S., Kramer, D., Phouc, D.L., Mitic, J., Anicic, D., Teniente, E., 2017. Enabling IoT ecosystems through platform interoperability. IEEE Softw. 34 (1), 54–61.

Chaturvedi, S., Tyagi, S., Simmhan, Y., 2007. Collaborative Reuse of streaming dataflows in IoT applications. In: 2017 IEEE 13th International Conference on E-Science (E-Science). pp. 403–412.

Chen, W., Li, J., Ma, J., Conradi, R., Ji, J., Liu, C., 2008. An empirical study on software development with open source components in the chinese software industry. Softw. Process: Improv. Pract. 13 (1), 89–100.

Chidamber, S.R., Kemerer, C.F., 1994. A metrics suite for object oriented design. IEEE Trans. Softw. Eng. 20 (6), 476–493.

Ciccozzi, F., Spalazzese, R., 2016. MDE4IoT: supporting the internet of things with model-driven engineering. In: International Symposium on Intelligent and Distributed Computing. Springer, Cham.

Couto, C.M.S., Rocha, H., Terra, R., 2018. A quality-oriented approach to recommend move method refactorings. In: Proceedings of the 17th Brazilian Symposium on Software Quality. pp. 11–20.

Dorofeev, K., Cheng, C.H., Guedes, M., Ferreira, P., Profanter, S., Zoitl, A., 2017. Device adapter concept towards enabling plug & produce production environments. In: 2017 22nd IEEE International Conference on Emerging Technologies and Factory Automation (ETFA). IEEE.

Feitosa, D., Ampatzoglou, A., Gkortzis, A., Bibi, S., Chatzigeorgiou, A., 2020. CODE reuse in practice: Benefiting or Harming Technical Debt. J. Syst. Softw. 110618.

Frakes, W., Terry, C., 1996. Software reuse: metrics and models. ACM Comput. Surv. 28 (2), 415–435.

Fremantle, P., 2015. A Reference Architecture for the Internet of Things. WSO2 White paper.

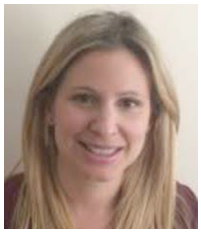
Gaffney, Jr., J.E., Durek, T.A., 1989. Software reuse—key to enhanced productivity: some quantitative models. Inf. Softw. Technol. 31 (5), 258–267.

Gui, G., Scott, P.D., 2006. Coupling and cohesion measures for evaluation of component reusability. In: Proceedings of the 2006 International Workshop on Mining Software Repositories. ACM, pp. 18–21.

Gupta, A., Cruzes, D., Shull, F., Conradi, R., Rønneberg, H., Landre, E., 2010. An examination of change profiles in reusable and non-reusable software systems. J. Softw. Maint. Evol.: Res. Pract. 22 (5), 359–380.

Guth, J., Breitenbücher, U., Falkenthal, M., Fremantle, P., Kopp, O., Leymann, F., Reinfurt, L., 2018. A detailed analysis of IoT platform architectures: concepts, similarities, and differences. In: Internet of Everything. Springer, Singapore, pp. 81–101.
Haefliger, S., Von Krogh, G., Spaeth, S., 2008. Code reuse in open source software. Manage. Sci. 54 (1), 180–193.
Heinemann, L., Deissenboeck, F., Gleirscher, M., Hummel, B., Irlbeck, M., 2011. On the extent and nature of software reuse in open source java projects. In: International Conference on Software Reuse. Springer, Berlin, Heidelberg, pp. 207–222.
Höttger, R., Ozcelikors, M., Heisig, P., Krawczyk, L., Cuadra, P., Wolff, C., 2018. Combining Eclipse IoT Technologies for a RPI3-Rover Along with Eclipse Kuksa. Software Engineering (Workshops).
Jatain, A., Nagpal, A., Gaur, D., 2013. Agglomerative Hierarchical approach for clustering components of similar Reusability. Int. J. Comput. Appl. 68 (2).
Katasonov, A., Kaykova, O., Khriyenko, O., Nikitin, S., Terziyan, V.Y., 2008. Smart semantic Middleware for the Internet of Things. Icinco-Icso 8, 169–178.
Kessel, M., Atkinson, C., 2018. Integrating reuse into the rapid, continuous software engineering cycle through test-driven search. In: 2018 IEEE/ACM 4th International Workshop on Rapid Continuous Software Engineering (RCoSE). IEEE.
Kim, J.E., Boulos, G., Yackovich, J., Barth, T., Beckel, C., Mosse, D., 2012. Seamless integration of heterogeneous devices and access control in smart homes. In: Intelligent Environments (IE), 2012 8th International Conference on IEEE. pp. 206–213.
Kovatsch, M., Lanter, M., Shelby, Z., 2014). Californium: Scalable cloud services for the internet of things with coap. In: 2014 International Conference on the Internet of Things (IOT). IEEE.
Krčo, S., Pokrić, B., Carrez, F., 2014. Designing IoT architecture (s): A European perspective. In: 2014 IEEE World Forum on Internet of Things (WF-IoT). pp. 79–84.
Lazarescu, M.T., 2014. Internet of things low-cost long-term environmental monitoring with reusable wireless sensor network platform. In: Internet of Things. Springer, Cham, pp. 169–196.
Leiba, Barry, 2012. Oauth web authorization protocol. IEEE Internet Comput. 16 (1), 74–77.
Levina, A.I., Dubgorn, A.S., Iliashenko, O.Y., 2017. Internet of things within the service architecture of intelligent transport systems. In: 2017 European Conference on Electrical Engineering and Computer Science (EECS) IEEE. pp. 351–355.
Lim, W.C., 1994. Effects of reuse on quality, productivity, and economics. IEEE Softw. 11 (5), 23–30.
Morgan, L., Finnegan, P., 2007. Benefits and drawbacks of open source software: an exploratory study of secondary software firms. In: IFIP International Conference on Open Source Systems. Springer, Boston, MA.
Neagu, G., Florian, V., Stanciu, A., Preda, S., 2016. Sensing as a service approach in health monitoring. In: 2016 15th RoEduNet Conference: Networking in Education and Research. IEEE, pp. 1–5.
O'Keeffe, M., Cinnéide, M.O., 2006. Search-based software maintenance. In: Conference on Software Maintenance and Reengineering (CSMR'06). IEEE, p. 10.
Osbeck, J., Virani, S., Fuentes, O., Roden, P., 2011. Investigation of automatic prediction of software quality. In: 2011 Annual Meeting of the North American Fuzzy Information Processing Society. IEEE, pp. 1–6.
Padhy, N., Singh, R.P., Satapathy, S.C., 2018. Software reusability metrics estimation: Algorithms, models and optimization techniques. Comput. Electr. Eng. 69, 653–668.
Papamichail, M., Diamantopoulos, T., Chrysovergis, I., Samlidis, P., Symeonidis, A., 2018. User-perceived reusability estimation based on analysis of software repositories. In: 2018 IEEE Workshop on Machine Learning Techniques for Software Quality Evaluation (MaLTeSQuE). IEEE.
Paschali, M.E., Ampatzoglou, A., Bibi, S., Chatzigeorgiou, A., Stamelos, I., 2017. Reusability of open source software across domains: A case study. J. Syst. Softw. 134, 211–227.
Pessoa, T.Q., Duarte-Figueiredo, F., 2017. NodePI: An integrated platform for smart homes. In: 2017 IEEE 9th Latin-American Conference on Communications (LATINCOM). IEEE, pp. 1–6.
Podgurski, A., Pierce, L., 1993. Retrieving reusable software by sampling behavior. ACM Trans. Softw. Eng. Methodol. 2, 286–303.
Prakash, B.A., Ashoka, D.V., Aradhya, V.M., 2012. Application of data mining techniques for software reuse process. Proc. Technol. 4, 384–389.
Prieto-Diaz, R., Freeman, P., 1987. Classifying software for reusability. IEEE Softw. 4 (1), 6.
Raemaekers, S., Deursen, A.V., Visser, J., 2012. An analysis of dependence on Third-party libraries in Open Source and proprietary systems. In: 6th International Workshop on Software Quality and Maintainability (SQM' 12).
Ravichandran, T., Rothenberger, M.A., 2003. Software reuse strategies and component markets. Commun. ACM 46 (8), 109–114.
Runeson, P., Höst, M., 2009. Guidelines for conducting and reporting case study research in software engineering. Empir. Softw. Eng. 14 (2), 131.
Saied, M.A., Ouni, A., Sahraoui, H., Kula, R.G., Inoue, K., Lo, D., 2018. Improving reusability of software libraries through usage pattern mining. J. Syst. Softw. 145, 164–179.
Schwittek, W., Eicker, S., 2013. A study on Third Party Component reuse in Java Enterprise Open Source Software. In: 16th International Symposium on Component-Based Software Engineering (CBSE' 13). ACM, pp. 75–80.
Serna, M.A., Sreenan, C.J., Fedor, S., 2015. A visual programming framework for wireless sensor networks in smart home applications. In: 2015 IEEE Tenth International Conference on Intelligent Sensors, Sensor Networks and Information Processing (ISSNIP). pp. 1–6.
Shani, U., Broodney, H., 2015. Reuse in model-based systems engineering. In: 2015 Annual IEEE Systems Conference (SysCon) Proceedings. IEEE.
Smiari, P., Bibi, S., 2018. A smart city application modeling framework: A case study on Re-engineering a smart retail platform. In: 2018 44th Euromicro Conference on Software Engineering and Advanced Applications (SEAA) IEEE. pp. 111–118.
Smiari, P., Bibi, S., Feitosa, D., 2019. Examining the Reusability of smart home applications: A case study on Eclipse Smart home. In: International Conference on Software and Systems Reuse. Springer, Cham, pp. 232–247.
Smirek, L., Zimmermann, G., Beigl, M., 2016. Just a smart home or your smart home–a framework for personalized user interfaces based on eclipse smart home and universal remote console. Procedia Comput. Sci. 98, 107–116.
Tanganelli, G., Vallati, C., Mingozzi, E., 2015. CoAPthon: Easy development of CoAP-based IoT applications with Python. In: 2015 IEEE 2nd World Forum on Internet of Things (WF-IoT). IEEE.
The Internet of Things Reference Model Whitepaper, 2014. The Internet of Things Reference Model - Whitepaper. Cisco.
Torchiano, M., Morisio, M., 2004. Overlooked aspects of COTS-based development. IEEE Softw. 21 (2), 88–93.
Tran, V., Liu, D.B., Hummel, B., 1997. Component-based systems development: challenges and lessons learned. In: Proceedings Eighth IEEE International Workshop on Software Technology and Engineering Practice Incorporating Computer Aided Software Engineering. IEEE.
Von Krogh, G., Spaeth, S., Lakhani, K.R., 2003. Community, joining, and specialization in open source software innovation: a case study. Res. Policy 32 (7), 1217–1241.
Wang, G., Ungar, L., Klawitter, D., 1999. Component assembly for Object-Oriented Distributed Systems. IEEE Comput. 71–78.
Wangoo, D.P., Singh, A., 2018. A classification based predictive cost model for measuring Reusability Level of Open Source software.
Ye, Y., Fischer, G., Reeves, B., 2000. Integrating active information delivery and reuse repository systems. In: ACM SIGSOFT Software Engineering Notes. Vol. 25. ACM, No. 6.
Yelamarthi, K., Aman, Md S., Abdelgawad, A., 2017. An application-driven modular IoT architecture. In: Wireless Communications and Mobile Computing 2017.
Yu, J., Benatallah, B., Saint-Paul, R., Casati, F., Daniel, F., Matera, M., 2007. A framework for rapid integration of presentation components. In: Proceedings of the 16th International Conference on World Wide Web. ACM.
Zamfir, M., Florian, V., Stanciu, A., Neagu, G., Preda, Ş., Militaru, G., 2016. Towards a platform for prototyping IoT health monitoring services. In: International Conference on Exploring Services Science. Springer, Cham, pp. 522–533.
Zanella, A., Bui, N., Castellani, A., Vangelista, L., Zorzi, M., 2014. Internet of things for smart cities. IEEE Internet Things J. 1 (1), 22–32.
Zhu, H., 2005. Building reusable components with service-oriented architectures. In: IRI-2005 IEEE International Conference on Information Reuse and Integration, Conf, 2005. IEEE.
Zimmermann, A., Schmidt, R., Sandkuhl, K., Wißotzki, M., Jugel, D., Möhring, M., 2015. Digital enterprise architecture-transformation for the internet of things. In: 2015 IEEE 19th International Enterprise Distributed Object Computing Workshop. IEEE.

Paraskevi Smiari is a Ph.D. candidate at the Department of Electrical and Computer Engineering at the University of Western Macedonia, Kozani, Greece. She holds a BSc in Informatics and Telecommunications Engineering (2016) from the same University. Her research interests include software process and architecture models, IoT application development and software reuse and quality. She currently works as an Android Developer in the medical device industry.



Dr. Stamatia Bibi is an Assistant Professor of software engineering in the Department of Electrical and Computer Engineering at the University of Western Macedonia, Kozani, Greece. She holds a BSc in Informatics (2002) and a Ph.D. (2008) in software engineering from the Aristotle University of Thessaloniki, Greece. Her interests include process models, cost estimation, quality assessment, and cloud computing. She currently has 40 publications among journal, conference papers and book chapters.

Dr. Daniel Feitosa is an Assistant Professor in the Faculty Campus Fryslân and the Chief Data Scientist at the Data Research Centre of the University of Groningen. He is also an associated researcher in the group of Software Engineering and Architecture of the University of Groningen. He holds a BSc degree (2010) and MSc (2013) in Computer Science from the University of São Paulo, Brazil, and was awarded his Ph.D. degree (2019) in Software Engineering by the University of Groningen. He currently has 20 publications among journal, conference papers and book chapters. His main research interests are in software architecture, software patterns and data analytics.