Component trustworthiness in an enterprise software platform ecosystem

Anastasia Bengtsson[1 [0000−0001−6635−2509]], Petter Nielsen[1 [0000−0003−3723−6976]], and Magnus Li[1 [0000−0002−5308−9171]]

Department of Informatics, University of Oslo, Oslo, Norway
{anastabe,pnielsen,magl}@ifi.uio.no

Abstract. Enterprise software packages are increasingly designed as extendable software platforms. These platforms are characterised by modular architecture that allows third parties to innovate and create value through the development of complementary applications. The development process of complementary applications from scratch is resource-intensive. One way of optimising the development process is by using the component-based software engineering (CBSE) approach that focuses on software reuse and suggests building applications with reusable components. There is a considerable amount of literature on CBSE; however, there has been little discussion on how component-based software engineering can strengthen third-party application development in the context of an enterprise software platform ecosystem. Specifically, it is unclear how the challenge of component trustworthiness can be addressed in this context. To explore this, we conducted a design science research (DSR) study to answer the following question: *What are design principles pertaining to component trustworthiness for implementing a component repository that facilitates component reuse in an enterprise software platform ecosystem?* In our study, we have explored the potential for component reuse in the ecosystem of the global health software platform DHIS2 by designing and developing a prototype component repository. During the design and development process, two design principles were identified: *Principle of component trustworthiness* and *Principle of balanced certification*. These principles are to guide researchers and practitioners on how a component repository can be implemented in the context of an enterprise software platform ecosystem.

Keywords: enterprise software platform ecosystem · component certification · component trustworthiness · software reuse · component-based software engineering · DHIS2 · design science research

1 Introduction

In recent years, there has been an increase in the development of generic software packages that, contrary to bespoke solutions, are developed for the open market and a large group of customers with similar business needs [20]. Many of those generic packages are "branded as software platforms rather than products" [19, p. 2]. Software platform architecture comprises two main elements: a

stable core developed and governed by a platform owner and third-party complementary applications developed by outside parties as an extension of the core's interface and functionality [28]. Part of the attraction of platform architecture is that it reduces the resources required to deliver specific functionality to customers and users because the platform is only developed once and instantiated many times. However, developing complementary applications from scratch can be time-consuming, as well as counterproductive and resource-inefficient if different parties work on applications that are similar with regard to functionality. For example, a registration form or a navigation bar developed in the context of some specific application can be extracted and further reused in another application. This practice gave rise to a software development approach called component-based software engineering that emerged in the late 1990s under the umbrella of the maxim "buy, don't build" [29, p. 21]. Previous studies established that successful adoption of component-based software engineering can shorten development life-cycle, optimise software time to market, increase developers' productivity and the quality of the developed solution [5,17,18,26,29]. Although extensive research has been carried out on component-based software engineering (CBSE), less is known about CBSE processes taking place in the context of an enterprise software platform ecosystem and what implications it could have for the parties involved. Accordingly, we set out to explore how one can facilitate component reuse by designing and implementing a component repository. We aimed to optimise the application development process by providing a repository for reusable components developed within an enterprise software platform ecosystem, making them more easily discoverable. We sought to increase the trustworthiness of reusable components in the ecosystem through the implementation of component certification. We used the following research question to guide our research: *What are design principles pertaining to component trustworthiness for implementing a component repository that facilitates component reuse in an enterprise software platform ecosystem?* To address this, the first author has conducted a design science research project on DHIS2 under the supervision of the co-authors. DHIS2, a generic web-based health management system for routine management and generation of health information; it is currently being used in more than 70 low- and middle-income countries [2,6]. DHIS2's core is governed by the Health Information Systems Programme (HISP) at the University of Oslo, and the core development work is done by the DHIS2 core team. DHIS2 is an enterprise system that entails in-country ownership of its instances and data [6, para. 11]. DHIS2 provides RESTful Web API enabling web application development using various web technologies such as JavaScript, HTML5, and CSS [7, para. 3]. Web application development for DHIS2 is open to different actors, such as HISP groups. In this project, we got the opportunity to collaborate with the DHIS2 core team and HISP East Africa developers. We have designed and developed a prototype component repository, DHIS2 Shared Component Platform (SCP), and identified two design principles related to component trustworthiness.

This paper is divided into seven sections. Section 2 provides the literature review on digital platforms and component-based software engineering, while section 3 is concerned with the methodology employed for this study. Section 4 elaborates on SCP's design considerations with regard to component trustworthiness and certification. In section 5, we present the design principles and discuss them in light of the previous research.

2 Related research

2.1 Enterprise software platforms

Many successful companies around the world, such as Apple, Amazon, Facebook and Microsoft, are moving away from a traditional business model in favour of a platform business model, which creates value by providing means for transaction and interaction between distinct groups of users. Amazon brings buyers and sellers together, while gaming platforms like Steam and PS4 facilitate interactions between gamers, game developers, and video game publishers. The examples provided above are typically referred to as consumer platforms and represent the business-to-consumer market (B2C), and have been the focus of a considerable amount of literature in past years [12]. Recent years have witnessed a growing trend among vendors of enterprise software towards adoption of a platform business model, allowing third parties to engage in value co-creation efforts [9,12]. Contrary to B2C markets, far too little attention has been paid to value creation practices in business-to-business (B2B) markets [9,12]. Hein et al. [12] argue that the value creation process in B2B platforms is conducted in a more complex setting: platform owners have to orchestrate interactions with customers, which are not "private individuals but legal organisations" [12, p. 504]. Moreover, it is typically harder to meet their needs "due to their requirements as legal entities" [12, p. 516]. Accordingly, we aim at facilitating component reuse in the DHIS2 platform ecosystem and contribute to the research on value creation in a B2B market of enterprise software that uses a platform business model.

2.2 Digital platforms

Software platform's value creation model is characterised by the ability of an extensible core to provide a basis for innovation in the form of complementary applications developed by third-party developers (also referred to as complementors) [2,28]. Software platform ecosystem comprises the following elements: an extensible platform core developed and governed by platform owners, and third-party applications that extend the platform's core through its interfaces, known as boundary resources [2,11,19,28]. Platform owners use boundary resources to help shift the design capabilities to third-party developers enabling them to innovate, which is an important part of a successful platform ecosystem [11]. At the same time, boundary resources can also be used by platform owners to protect the platform from potentially malicious third parties [11].

Boundary resources can be classified into social boundary resources and technical boundary resources [3,10]. Technical boundary resources can be divided into two categories: application boundary resources and development boundary resources [2,3,11]. Application boundary resources, such as APIs, are the technologies necessary for establishing the interaction between the platform core and third-party applications, while development boundary resources, such as SDKs, are the technologies meant to support third-party development [2,3,10]. Social boundary resources such as guides and documentation share the supportive role with development boundary resources; however, they are typically knowledge-based [2,3].

2.3 Component-based software engineering

It is not uncommon for vastly different applications to have similarities in implemented functionality and features, for example, such as a feedback mechanism, a search and filter function, and navigation [2]. To optimise third-party application development process, one can develop specific functionality once and then make it available for further reuse as a reusable component for many other applications. A software component is a key element of component-based software engineering and is defined as "a unit of composition with contractually specified interfaces and context dependencies only. A software component can be deployed separately and is subject to composition by third parties" [27, p. 41]. A component is usually initially built as part of a specific application, then extracted as a standalone module and prepared for further reuse, typically through a process of generification that eliminates the component's application-specific aspects [18,29]. Afterwards, the component, along with its interface definition, dependency specification, metadata, description and other necessary information, is published to a component repository where it is stored and made available for other developers [5,18,29]. In the CBSE literature, this process is referred to as *CBSE for reuse* and the developers that create reusable component are known as component providers [13,25,27]. A reusable component may go through a process of *component certification* carried out by a third party known as component certifier to ensure some level of quality and trustworthiness [5,13,25,29]. Lastly, during a process referred to as *CBSE with reuse*, component users reuse components that they discovered based on their established requirements and required quality and trustworthiness [25].

2.4 Component trustworthiness and certification

Addressing component trustworthiness is currently one of the biggest challenges in CBSE [5,17,25], and the issue of component trustworthiness in third-party application development was brought to our attention by the DHIS2 core team, with whom we collaborated [2]. Trustworthiness is defined as the degree of confidence that the system behaves as expected [5,22,25]. The concept of trustworthiness is reflected by a computer system property dependability [25]; some use these two concepts interchangeably [22]. Dependability typically comprises

other properties such as availability, reliability, safety, and security; however, different systems prioritise different properties [5,25], and in CBSE, one has to relate system properties to component properties [5]. Component users face the challenge of component trustworthiness each time they utilise a third-party component with closed source code or when there are no available resources to analyse the component's source code, which makes it hard to identify unpredictable non-functional behaviour, malicious code, and "undocumented failure modes" [25, p. 454]. Component certification is a way to address component trustworthiness [25] and was chosen by the DHIS2 core team as an approach to establish component quality and increase component trustworthiness in the DHIS2 ecosystem. Certification is defined as a "formal demonstration that a system or component complies with its specified requirements and is accepted for operational use" [16, p. 63] and the party responsible for certification is referred to as a certifier [25]. The process of component certification is typically viewed as an assessment of reusable components according to a set of specific attributes or qualities defined in certification requirements [2,29]. There is still considerable uncertainty around different aspects of component certification due to the absence of standard techniques and methods; thus, certification is considered to be a major challenge in CBSE that requires further investigation and clarification [5,17,22,25,29]. Researchers agree on the fact that certification should be done by an independent third party [13,25], but it raises the question of who would be responsible if a certified component would not operate as expected [25]. Our empirical work in this study has shown that it is challenging to develop certification architecture and define certification requirements suitable for a large number of actors with divergent application development practices. Mohammad [22] suggests further research on certifier's credibility, certificate validity, and assurance of certified components immutability.

To conclude, this section has attempted to provide a brief summary of the literature on software platforms, specifically enterprise software platforms and explained software platforms' value creation model. Furthermore, we presented conceptual background on CBSE and the challenge of component trustworthiness.

3 Research approach

The first author took the lead on the research project conducted as part of a master's degree programme, while the co-authors acted as supervisors (for more details see [2]). The design principles we present in this paper were also documented in the first author's master's thesis. The design and development work on SCP was carried out together with two other master's students.

In order to address the research question of this paper, we employed DSR methodology that guided the design and development of DHIS2 Shared Component Platform. We have inductively identified two design principles from the process of the artefact's design. Data gathered through the interviews and focus groups alongside design and development work provided a foundation for

establishing the design principles, while CBSE, as a design theory, established and provided theoretical grounding. The evaluation activity in DSR was used to evaluate the application of the design principles in the developed artefact and assess the artefact's usefulness. In this section, we will explain the methodology and methods used for data collection and analysis in our study.

3.1 Research methodology

Design science research within the pragmatic research paradigm was chosen as a research methodology, as it is inherently a problem-solving paradigm that allows researchers and practitioners to participate in engaged research efforts and make a theoretical contribution to the research in the form of prescriptive knowledge [1,8,15]. Our research process was guided by the DSR process model [24] that consists of the following activities: "problem identification and motivation, definitions of the objectives for a solution, design and development, demonstration, evaluation, and communication" [4, p. 5]. We worked together with the DHIS2 practitioners engaged in third-party application development. The data gathered throughout our collaboration was used to define the objectives for our solution guiding the design and development activity, and to provide improvements to the developed artefact.

3.2 Data collection

The data collection methods in this study can be separated into two categories. The first category includes the methods we used to identify the problem, define objectives for our solution, and continuously improve the artefact during the design and development process. This category comprises two semi-structured interviews with HISP East Africa developers (5) and four focus groups with the DHIS2 core team (3). The goal of our interviews with HISP East Africa developers was to learn about their third-party application development and component reuse practices. For each interview, we developed an interview guide with a set of questions to ensure we covered our interview goal. Exploratory focus groups with the DHIS2 core team allowed us to engage in discussions related to different aspects of SCP.

The second category includes the methods for data collection as part of the formative intermediate evaluation conducted after the design and development process activity. Evaluation is an essential activity in DSR, used to establish the degree to which an artefact provides environmental utility for the end-user and to gain feedback for further improvements [30]. The evaluation criteria were chosen in accordance with a hierarchy of DSR goals for socio-technical systems solutions (Figure 3 [14, p. 8]) proposed by Hevner et al. In DSR evaluation, utilitarian goals with their corresponding evaluation criteria should be fulfilled before one can move on to the upper-level goals [14]; therefore, the following utilitarian criteria were chosen: accuracy, efficacy, performance, and usefulness. Additionally, we have included the criterion of openness pertaining to cognitive and aesthetic goals. The criterion of accuracy could be assessed quantitatively;

therefore, unit testing was chosen as an evaluation method. The criteria of openness and performance are concerned with the artefact's non-functional properties and were evaluated through an expert evaluation. The criteria of usefulness and efficacy, essential for determining artefact's utility, were assessed through a naturalistic empirical evaluation by users. The evaluators completed several tasks using the artefact and were asked to complete surveys containing both open-ended and close-ended questions. The evaluation participants were chosen using a combination of convenience sampling and purposive sampling [21].

3.3 Data analysis

We have employed thematic analysis [23] as a tool to analyse the data collected through this study. The data were first transcribed, and then we developed an initial set of codes to identify larger themes. For instance, a global theme *software reuse* comprises an organising theme *component* that consists of the following codes: *component granularity*, *component discovery*, *component specification*. A comment from one of the HISP developers, "in fact, we haven't really found many components from other HISP groups", is an example output of the *component discovery* code. Thematic analysis was conducted using a qualitative data analysis software package NVivo 12.

4 Design considerations

This section presents the design considerations that shaped the requirements and design of our artefact with regard to component trustworthiness and certification.

During one of our focus groups, the DHIS2 core team raised the problem of component trustworthiness, which was not addressed by any part of the DHIS2 platform ecosystem. They stated it would be good "to promote the ones [components] that have a certain level of quality, or that they have a certain level of maturity, and testing" (Core team developer, personal communication, October 2, 2020). Various ways of approaching this problem were discussed. In discussions of solving this problem by implementing component certification as part of SCP, the DHIS2 core team showed their interest in governing the certification process and expressed a preference for using automation in the certification process. Additional issues that came up during discussions with this DHIS2 core team were that we would have to establish certification requirements and how certification would be affected by different component versions.

When the possibility of using certification was brought up during the interviews with the HISP groups, they expressed an interest in having the certification requirements outlined: "What is the [certification] process, have you mapped out, let's say a checklist of what you go through to decide that package is [certified] or not?" (senior developer in HISP East Africa, personal communication, October 9, 2020). In the course of the design phase, it became clear to us that the DHIS2 core team, as a platform owner, has a preference towards React web development framework, which is not aligned with the current practices of some HISP

groups. This, in combination with the inherent power imbalance that would exist between the party that sets certification requirements and component providers that have to meet these requirements, made it clear that the certification process could have an impact on the governance balance in the DHIS2 platform ecosystem. We decided to implement certification by having a comma-separated values (CSV) file with a list of the identifiers of certified components, and the DHIS2 core team agreed that this approach was preferable among several options that were brought up. In comparison to the other considered solutions, this approach would place the lowest maintenance burden on certifiers because they would only govern the identifiers of the components and not the source code. The identifiers in this list would be represented by Node Package Manager (NPM) packages' names and versions, as all components in SCP are contained within NPM packages and stored in NPM Registry. Using NPM package names and versions as identifiers guarantees that the components are not modified after they were certified, as NPM packages in NPM Registry are immutable. We decided to host the list in a GitHub repository and to have component providers submit the component identifiers to this list using GitHub pull requests. This allowed us to use GitHub Actions for automation, and the pull request process would also be familiar to the practitioners we collaborated with, as they were already using GitHub. For performing actual certification checks, we developed a command-line interface (SCP CLI). SCP CLI is invoked by GitHub Actions workflow and can also be used by practitioners for the purpose of pre-certification, which is a verification that the components adhere to component quality standards established by certifiers. The DHIS2 core team was not specific on certification requirements. When determining the certification requirements, we concluded that in terms of functionality, certification could be classified as manual or automated; and more subjective or more objective (Figure 1). It should be noted that these are not binary values. We decided that for the SCP certification process, we would perform some automated checks but that the final decision to certify a component would require a manual qualitative assessment by the certifiers. The automated checks we implemented were a security check using `npm audit` and static code analysis using ESLint.

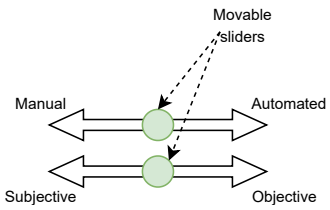

Fig. 1. Certification process classification.

While the role of certifier had not yet been assigned at the conclusion of our project, the plan was that this role would be assigned to the DHIS2 core

team. This means the DHIS2 core team would establish component quality metrics and develop certification requirements. Considering the fact that they also act as platform owners, they could pursue their own interests and attempt to control third-party application development practices in the ecosystem through component certification.

5 Discussion

In this section, we present two design principles established considering the empirical data gathered during this study and discuss their relevance in light of the existing research.

5.1 Principle of component trustworthiness

Empirical evidence obtained in this study showed the DHIS2 core team's interest in implementing component certification to promote the components with a certain level of quality. This supports findings in the literature we reviewed [5,25,17], which shows that component trustworthiness is an important but challenging aspect of CBSE, and the component certification is a way to address it. Therefore, we suggest that when one implements a component repository in an enterprise software platform ecosystem, one should consider implementing component certification. We formulate *Principle of component trustworthiness* as follows:

> **Design Principle (DP) of component trustworthiness:** To increase component trustworthiness in an enterprise software platform ecosystem, implement component certification because it is a means to establish component trustworthiness and increase component quality.

Component reuse in the DHIS2 ecosystem occurs within individual HISP groups, where different mechanisms for component quality assurance, such as code reviews and testing, can be used within the same organisation in order to increase component trustworthiness. However, when components are shared between the HISP groups, "inherently independent organisations with different policies" [2, p. 99], it can become more challenging or even not possible to control the quality of the components, which can result in reduced trust and consequently in limited component reuse [17]. Therefore, we emphasise the importance of component certification as a means to increase component trustworthiness in the context of an enterprise software platform ecosystem. In addition to an increase in trustworthiness, certification incentivises third-party developers to create components of sufficient quality to pass the certification process.

As mentioned in section 2.4, there is a need for further research on component certification due to the lack of standardised methods and procedures; therefore, we contribute with insights and recommendations pertaining to the implementation of component certification. In the literature, it has been proposed to utilise component versioning to establish component compatibility [26]. Additionally, a

component repository is expected to support versions and version updates [29]. We implemented a version-specific certification on immutable NPM packages because we considered it to be a safer solution than certification without taking into account versioning. While the DHIS2 core team did not initially support version-specific certification, the evaluation has shown that it is a preferable solution because certifiers cannot guarantee the quality of the components they have not evaluated: "Good approach. Lots of stuff break or gets worse over time. We can only verify what we have here and now" (Core team developer, personal communication, December 14, 2020).

Immutability of versions is similarly critical to prevent modifications to a component version after it has been certified and contributes to the future research proposed by Mohammad [22]. One major drawback of version-specific certification and immutability of the component versions is that certifying each new component version is more resource-intensive. One mitigation we considered was to certify components based on the **Major** and **Minor** facets of semantic version numbers. Still, such an approach would be vulnerable to malicious actors and incorrect usage of semantic versioning. The open nature of the SCP's certification process implemented using GitHub Actions workflow has received positive feedback from one of the DHIS2 core team developers during the artefact evaluation:

> I think it's quite nice. The PR [pull request] *acts as a discussion platform* [emphasis added] when we maintainers want the component [provider] to make changes also puts the whole thing [certification process] *open and public* [emphasis added]. (Core team developer, personal communication, December 14, 2020)

Transparency and openness of a certification process can be beneficial to component providers and users, as it allows them to observe how reusable components are being assessed and what certification metrics are in use, and even engage in discussion with certifiers. Lastly, we suggest that certification requirements should be communicated to component providers to give them an opportunity to develop components compliant with these requirements and adhere to a certain quality standard. Certification checks that require minimal human discretion can be automated, which can result in a more objective and rapid certification process. Pre-certification allows component providers to perform self-assessment of the components before they are published to a component repository and submitted for certification. To sum up, we suggest to consider implementing a transparent and open, and preferably automated certification process that operates on immutable component versions.

5.2 Principle of balanced certification

If the DHIS2 core team as a platform owner would take responsibility for component certification, it would allow them to use it as a control mechanism over component reuse practices and, consequently, control or influence third-party application development for DHIS2. Our recommendation is that the researchers

and practitioners who are to implement component certification and assign the role of certifier consider what implications their decisions could have on the balance of control and autonomy in an enterprise software platform ecosystem. Given this, we formulate *Principle of balanced certification* as follows:

> **Design Principle (DP) of balanced certification:** To increase the likelihood of component repository adoption and cultivate its growth in an enterprise software platform ecosystem, implement component certification with a balance between control and autonomy because the right governance balance is important for the ecosystem's growth and success.

With regard to the theory on governance and control in a software platform ecosystem presented by Tiwana [28], we highlight the importance of this principle by providing a critical perspective on the situation where platform owners assume governance of component certification. In a relationship between platform owners and third-party application developers, there are two types of decision right: *platform decision rights* and *app decision rights* [28]. Platform decision rights pertain to the decisions around the platform, while app decision rights pertain to third-party application development; it is worth noting that app decision rights have various degrees of decentralisation [28]. In order to centralise and extend their app decision rights, platform owners can utilise three formal control mechanisms, namely *gatekeeping*, *process control*, and *metrics* [28]. We have identified the following limitations of these mechanisms in our specific case, where control mechanisms are applied through a development boundary resource [28]:

1. *The three control mechanisms are predicated on a platform owner's ability to exercise direct control over the process of web application acceptance in a software platform's ecosystem.*
   If a component repository and associated certification system are implemented as a boundary resource, it is not mandatory for third-party application developers to utilise them. Platform owners can attempt to exercise direct control over components, for example, by allowing only certified components to be published in a repository; however, there is a risk that third-party application developers will not want to certify their components and not use the repository.
2. *Process control and metrics control mechanisms are based on a reward and penalisation system.*
   An attempt to enforce direct control through penalisation could impact a component repository and certification negatively, as third-party developers do not have to accept this control mechanism. On the other hand, we see the possibility for influencing component reuse and consequently third-party application development through the adoption of rewarding techniques, for example, by promoting certified components and making them more visible to component users.
3. *Process control and metrics control mechanisms require fair and objective judgement. Quality assessment metrics should be pre-defined and objective.*

This is in line with our recommendations to pre-define and to communicate certification requirements to component providers, as discussed in section 5.1.

It is worth noting that there is a difference between attempted control by platform owners and the possibility to realise such control, and, generally, for an attempt to be successful, two prerequisites must be met: the control mechanism should be accepted by third-party developers, and it should be fair and reasonable [28]. While it can be challenging for platform owners to extend their app decision rights through certification in a development boundary resource, in our opinion, there is an opportunity to carefully influence third-party application development practices through a reward system, fair and objective certification process with pre-defined and communicated certification requirements.

6 Conclusion

We have presented two design principles related to component trustworthiness and certification. Component reuse in an enterprise software platform ecosystem is similar to component markets and internet-wide component reuse, as the components are being reused between independent organisations, making it more challenging to assess component quality and trustworthiness. SCP's architectural design contributes to CBSE, more specifically to certification methods and procedures, which is not a widely understood area of CBSE. We suggested how version-based certification can be built within an enterprise software platform ecosystem handling a larger number of components without putting a high maintenance burden on certifiers. Moreover, we provided a critical perspective on situations where a platform owner takes responsibility for component certification and uses it as a mechanism to extend his app decision rights, building upon the existing research on software platform governance by Tiwana [28]. An attempt to curtail third-party developers' autonomy might hamper the adoption of a component repository, and if the component repository was attempting to improve component trustworthiness with component certification, such an attempt would also be impeded.

Lack of time and limited access to data due to the COVID-19 pandemic are the limitations of this study. Due to time constraints, a final DSR evaluation was not conducted, and the plans for the DHIS2 core team to take over the component certification process were not fulfilled.

A natural progression of this work is to analyse the role of development boundary resources with regard to governance and control in an enterprise software platform ecosystem.

7 Acknowledgements

We gratefully acknowledge HISP East Africa developers and the DHIS2 core team for their valuable contribution and willingness to participate in this project.

References

1. Baskerville, R.L., Kaul, M., Storey, V.C.: Genres of inquiry in design-science research: Justification and evaluation of knowledge production. MIS Quarterly **39**(3), 541–564 (2015), https://www.jstor.org/stable/26629620
2. Bengtsson, A.: Facilitating Software Reuse. Using Design Science Research To Develop Design Principles For Implementing A Component Repository. Master's thesis, University of Oslo (2021), http://urn.nb.no/URN:NBN:no-89800, publisher: Unpublished
3. Bianco, V.D., Myllarniemi, V., Komssi, M., Raatikainen, M.: The Role of Platform Boundary Resources in Software Ecosystems: A Case Study. In: 2014 IEEE/IFIP Conference on Software Architecture. pp. 11–20. IEEE, Sydney, Australia (Apr 2014). https://doi.org/10.1109/WICSA.2014.41, http://ieeexplore.ieee.org/document/6827094/
4. Brocke, J.v., Hevner, A., Maedche, A.: Introduction to Design Science Research, pp. 1–13 (09 2020). https://doi.org/10.1007/978-3-030-46781-4_1
5. Crnkovic, I., Larsson, M. (eds.): Building reliable component-based software systems. Artech House computing library, Artech House, Boston (2002)
6. About dhis2. https://www.dhis2.org/about (2021), last accessed 4 Aug 2021
7. Technology platform. https://dhis2.org/technology/ (2021), last accessed 12 Oct 2021
8. Dresch, A., Lacerda, D.P., Antunes Jr, J.A.V.: Design Science Research: A Method for Science and Technology Advancement. Springer International Publishing, Cham (2015). https://doi.org/10.1007/978-3-319-07374-3, http://link.springer.com/10.1007/978-3-319-07374-3
9. Foerderer, J., Kude, T., Schuetz, S.W., Heinzl, A.: Knowledge boundaries in enterprise software platform development: Antecedents and consequences for platform governance. Information Systems Journal **29**(1), 119–144 (Jan 2019). https://doi.org/10.1111/isj.12186, https://onlinelibrary.wiley.com/doi/10.1111/isj.12186
10. Ghazawneh, A.: Towards a boundary resources theory of software platforms. Jönköping International Business School, Jönköping (2012), diss. (sammanfattning) Jönköping : Högskolan i Jönköping, 2012
11. Ghazawneh, A., Henfridsson, O.: Balancing platform control and external contribution in third-party development: the boundary resources model: Control and contribution in third-party development. Information Systems Journal **23**(2), 173–192 (Mar 2013). https://doi.org/10.1111/j.1365-2575.2012.00406.x, http://doi.wiley.com/10.1111/j.1365-2575.2012.00406.x
12. Hein, A., Weking, J., Schreieck, M., Wiesche, M., Böhm, M., Krcmar, H.: Value co-creation practices in business-to-business platform ecosystems. Electronic Markets **29**(3), 503–518 (Sep 2019). https://doi.org/10.1007/s12525-019-00337-y, http://link.springer.com/10.1007/s12525-019-00337-y
13. Heineman, G.T., Councill, W.T. (eds.): Component-based software engineering: putting the pieces together. Addison-Wesley, Boston (2001)
14. Hevner, A., Prat, N., Comyn-Wattiau, I., Akoka, J.: A pragmatic approach for identifying and managing design science research goals and evaluation criteria. HAL p. 16 (2018)
15. Iivari, J., Venable, J.R.: Action research and design science research - Seemingly similar but decisively dissimilar. AISeL p. 13 (2009)

16. ISO/IEC 25010: ISO/IEC 25010:2011, systems and software engineering — systems and software quality requirements and evaluation (square) — system and software quality models (2011)
17. Jalender, B., Govardhan, D., Premchand, P.: Breaking the boundaries for software component reuse technology. International Journal of Computer Applications **13** (01 2010). https://doi.org/10.5120/1782-2458
18. Kotonya, Sommerville, Hall: Towards a classification model for component-based software engineering research. In: Proceedings of the 20th IEEE Instrumentation Technology Conference (Cat No 03CH37412) EURMIC-03. pp. 43–52. IEEE, Belek-Antalya, Turkey (2003). https://doi.org/10.1109/EURMIC.2003.1231566, http://ieeexplore.ieee.org/document/1231566/
19. Li, M.: Making Usable Generic Software - The Platform Appliances Approach (2019). https://doi.org/10.13140/RG.2.2.11381.83687, http://rgdoi.net/10.13140/RG.2.2.11381.83687, publisher: Unpublished
20. Li, M., Nielsen, P.: Making usable generic software. a matter of global or local design? (12 2019)
21. Lopez, V., Whitehead, D.: Sampling data and data collection in qualitative research, pp. 123–140 (01 2013)
22. Mohammad, M.S.: A formal component-based software engineering approach for developing trustworthy systems. Ph.D. thesis, Library and Archives Canada = Bibliothèque et Archives Canada, Ottawa (2011), iSBN: 9780494634172 OCLC: 769258225
23. Nowell, L.S., Norris, J.M., White, D.E., Moules, N.J.: Thematic Analysis: Striving to Meet the Trustworthiness Criteria. International Journal of Qualitative Methods **16**(1), 160940691773384 (Dec 2017). https://doi.org/10.1177/1609406917733847, http://journals.sagepub.com/doi/10.1177/1609406917733847
24. Peffers, K., Tuunanen, T., Rothenberger, M.A., Chatterjee, S.: A Design Science Research Methodology for Information Systems Research. Journal of Management Information Systems **24**(3), 45–77 (Dec 2008). https://doi.org/10.2753/MIS0742-1222240302, https://www.tandfonline.com/doi/full/10.2753/MIS0742-1222240302
25. Sommerville, I.: Software engineering. Pearson, Boston, 9th ed edn. (2011), oCLC: ocn462909026
26. Szyperski, C.: Component Software: Beyond Object-Oriented Programming. Addison-Wesley Longman Publishing Co., Inc., USA, 2nd edn. (2002)
27. Szyperski, C., Gruntz, D., Murer, S.: Component software: beyond object-oriented programming. Addison-Wesley Component software series, Addison-Wesley, London, 2nd ed edn. (2011), oCLC: 838151413
28. Tiwana, A.: Platform ecosystems: aligning architecture, governance, and strategy. MK, Amsterdam ; Waltham, MA (2014)
29. Tiwari, U., Kumar, S.: Component-Based Software Engineering: Methods and Metrics. Chapman and Hall/CRC (09 2020). https://doi.org/10.1201/9780429331749
30. Venable, J., Pries-Heje, J., Baskerville, R.: FEDS: a Framework for Evaluation in Design Science Research. European Journal of Information Systems **25**(1), 77–89 (Jan 2016). https://doi.org/10.1057/ejis.2014.36, https://www.tandfonline.com/doi/full/10.1057/ejis.2014.36