UNIVERSIDADE FEDERAL DE MINAS GERAIS
Instituto de Ciências Exatas
Programa de Pós-Graduação em Ciência da Computação

Rafael Torres Souza

Assessing Reusable Web Applications: The Django Ecosystem Case

Belo Horizonte
2023

Rafael Torres Souza

Assessing Reusable Web Applications: The Django Ecosystem Case

Final Version

Thesis presented to the Graduate Program in Computer Science of the Federal University of Minas Gerais in partial fulfillment of the requirements for the degree of Master in Computer Science.

Advisor: André Cavalcante Hora

Belo Horizonte
2023

Souza, Rafael Torres.

S729a Assessing reusable web applications: [recurso eletrônico] the Django Ecosystem case / Rafael Torres Souza- 2023.

1 recurso online (60 f. il., color.) : pdf.

Orientador: André Cavalcante Hora.

Dissertação (Mestrado) - Universidade Federal de Minas Gerais, Instituto de Ciências Exatas, Departamento de Ciências da Computação.
Referências: f. 54-60

1. Computação – Teses. 2. Engenharia de software – Teses. 3. Website – Desenvolvimento - Teses. 4. Django (Recurso eletrônico). I. Hora, André Cavalcante.
II. Universidade Federal de Minas Gerais, Instituto de Ciências Exatas, Departamento de Computação. III.Título.

CDU 519.6*32(043)

Ficha catalográfica elaborada pela bibliotecária Irénquer Vismeg Lucas Cruz
CRB 6/819 - Universidade Federal de Minas Gerais – ICEx

UNIVERSIDADE FEDERAL DE MINAS GERAIS
INSTITUTO DE CIÊNCIAS EXATAS
DEPARTAMENTO DE CIÊNCIA DA COMPUTAÇÃO
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

FOLHA DE APROVAÇÃO

ASSESSING REUSABLE WEB APPLICATIONS: THE DJANGO ECOSYSTEM CASE

RAFAEL TORRES SOUZA

Dissertação defendida e aprovada pela banca examinadora constituída pelos Senhores:

Prof. André Cavalcante Hora - Orientador
Departamento de Ciência da Computação - UFMG

Prof. Eduardo Magno Lages Figueiredo
Departamento de Ciência da Computação - UFMG

Prof. Marco Túlio de Oliveira Valente
Departamento de Ciência da Computação - UFMG

Belo Horizonte, 14 de dezembro de 2023.

Documento assinado eletronicamente por **Andre Cavalcante Hora**, **Professor do Magistério**

Superior, em 01/04/2024, às 10:49, conforme horário oficial de Brasília, com fundamento no art. 5º do Decreto nº 10.543, de 13 de novembro de 2020.

Documento assinado eletronicamente por **Eduardo Magno Lages Figueiredo**, **Professor do Magistério Superior**, em 01/04/2024, às 12:06, conforme horário oficial de Brasília, com fundamento no art. 5º do Decreto nº 10.543, de 13 de novembro de 2020.

Documento assinado eletronicamente por **Marco Tulio de Oliveira Valente**, **Professor do Magistério Superior**, em 01/04/2024, às 13:23, conforme horário oficial de Brasília, com fundamento no art. 5º do Decreto nº 10.543, de 13 de novembro de 2020.

A autenticidade deste documento pode ser conferida no site https://sei.ufmg.br/sei/controlador_externo.php?acao=documento_conferir&id_orgao_acesso_externo=0, informando o código verificador **3148951** e o código CRC **144A6334**.

Referência: Processo nº 23072.219105/2024-78 SEI nº 3148951

To Kecia, Zelia, and Fernando.

Acknowledgments

First of all, I would like to thank my family, who have always supported me and cheered for my success. It wouldn't be possible to get here without them.

My extreme gratitude to all the friendships I've made, with whom I shared the good and the bad moments from this journey since my undergraduate. In particular, my friends from the "Mestres do Método", who have always been by my side and who motivate and inspire me so much.

My thanks to the university UFMG, who expanded my horizons by providing me with so many experiences and putting me in touch with so many incredible people. Special gratitude to both staff and professors from the Computer Science Department - especially my advisor and the professors who joined my dissertation's examination - for all the valuable lessons and knowledge sharing during these years.

Lastly, my gratitude to all whom I shared - even in quick moments - the yearnings, frustrations, and joys of this path.

"*I knew exactly what to do, but in a much more real sense, I had no idea what to do.*"
(Michael Scott)

Resumo

Os aplicativos web modernos são construídos utilizando web frameworks, que oferecem benefícios como reutilização de recursos e melhoria de produtividade. Ao utilizar esses frameworks, os desenvolvedores podem contar com várias funcionalidades comumente requeridas por aplicações do mundo real, como autenticação, gerenciamento de sessão e roteamento de URL. Entretanto, apesar dos benefícios, eles não atendem a todas as necessidades de desenvolvimento. É esperado que nem todos os requisitos sejam satisfeitos pelo framework selecionado, portanto, os desenvolvedores podem precisar implementar suas soluções localmente ou contar com bibliotecas externas.

O framework web Django em particular possui um rico ecossistema de pacotes disponíveis na plataforma Django Packages. Esses pacotes reutilizáveis oferecem diversas funcionalidades como autenticação moderna, armazenamento em cache, e integrações com sistemas externos. Devido à estrutura de aplicações construídas com o Django framework, os desenvolvedores podem reutilizar aplicativos prontos para uso. No entanto, atualmente, não está claro quais soluções estão disponíveis para serem reutilizadas pelos desenvolvedores, ou quão ativo e confiável esse ecossistema realmente é. Estas informações seriam importantes para caracterizar o ecossistema Django, ajudando os desenvolvedores a entender melhor os aspectos positivos e negativos da adoção deste framework.

Para preencher essa lacuna, propomos um estudo empírico para avaliar o ecossistema Django de aplicações reutilizáveis. Analisamos 487 aplicações reutilizáveis para entender seu domínio alvo, nível de atividade, testes e compatibilidade. Descobrimos que as aplicações reutilizáveis estão concentradas principalmente em três categorias: ferramentas de desenvolvedor (27,8%), interfaces de usuário (17,1%) e persistência (15,9%). Detectamos também que essas categorias possuem características distintas. As ferramentas de desenvolvedor têm a maior concentração de aplicativos jovens e inativos. Em contrapartida, os aplicativos de persistência são os mais antigos, enquanto os de segurança são os mais ativos. Além disso, aplicações de desempenho possuem proporcionalmente mais testes. As versões do Python mais suportadas são 3.7 a 3.11, enquanto as versões do Django mais suportadas são 3.2 e 4.0. Finalmente, com base em nossas descobertas, apontamos implicações tanto para profissionais da área quanto para pesquisadores.

Palavras-chave: ecossistemas de software; desenvolvimento web; evolução de software; estudo empírico; Django web framework.

Abstract

Modern web applications are built on top of web frameworks, which offer benefits such as feature reuse and productivity improvement. By relying on web frameworks, developers can safely deal with many needs of real-world web applications, like authentication, session management, and URL routing. Despite the benefits of relying on web frameworks, they do not address all the needs to create web applications. It is expected that not all requirements are satisfied by the selected web framework, therefore, developers may need to implement their local solutions or rely on external libraries for reusability purposes.

In particular, the Django web framework provides a rich software ecosystem of reusable packages that are available in the Django Packages platform. These reusable packages support developers with advanced development aspects, such as modern authentication, caching, and deployment. Due to the Django native structure, developers can reuse out-of-the-box applications (known as reusable apps). However, currently, it is not clear what solutions are available to be reused by developers, or how active and reliable this ecosystem really is. This information would be important to characterize the Django ecosystem, helping developers to better understand the positive and negative aspects of adopting this framework.

To overcome this gap, in this research, we propose an empirical study to assess the Django ecosystem of reusable applications. We analyze 487 reusable applications to understand their target domain, level of activity, tests, and compatibility. We find that reusable applications are mostly concentrated in three categories: developer tools (27.8%), user interface (17.1%), and persistence (15.9%). We also detect that these categories have distinct characteristics. Developer tools have the highest concentration of young and inactive applications. In contrast, persistence applications are the oldest ones, while security ones are the most active. Moreover, performance applications have proportionally more tests. The most supported Python versions are 3.7 to 3.11, while the most supported Django versions are 3.2 and 4.0. Finally, based on our findings, we provide implications for both practitioners and researchers.

Keywords: software ecosystem; web development; software evolution; empirical study; Django web framework.

List of Figures

List of Tables

Contents

Chapter 1

Introduction

1.1 Motivation

Modern web applications are built on top of web frameworks, which offer benefits such as feature reuse and productivity improvement, decreasing development costs [46, 34, 50, 41]. By relying on web frameworks, developers can safely handle many needs of real-world web applications, like authentication, session management, URL routing, and static file management, to name a few. Nowadays, there are several popular web frameworks to support web development in most programming languages, for example, Django for Python, Spring Boot for Java, Laravel for PHP, ASP.NET for C#, and React for JavaScript.

Despite the benefits of web frameworks, they do not address all requirements to create web applications. It is expected that not all requirements are satisfied by the selected web framework, thus, developers may need to implement their local solutions or rely on external packages for reusability purposes [52, 55]. For example, web frameworks may provide basic features to handle user authentication, but they may not provide advanced solutions to authenticate the user via social networks or ensure authentication security via captchas. In fact, with the popularity of some web frameworks, those external packages may create an essential software ecosystem of reusable packages that support building web applications with more productivity and quality.

In particular, the Django web framework [20] is among the most used web frameworks nowadays[1] and is adopted by top tech companies and organizations, such as Instagram, Pinterest, and Mozilla.[2] In short, Django allows developers to work on both the front-end and back-end of the web application, this way, developers can build full-stack applications or focus on the back-end by exposing features to the front-end via microservices, for example.

Django provides a rich software ecosystem of reusable packages that is available in the Django Packages platform [22]. The reusable packages support developers with

[1] https://survey.stackoverflow.co/2022/#most-popular-technologies-webframe
[2] https://www.djangoproject.com/start/overview

advanced development aspects, such as authentication, caching, and deployment. Moreover, due to the Django native structure, developers can reuse out-of-the-box applications (known as *reusable applications*) [19]. For example, a blog application may rely on reusable apps to support post comments and captcha verification instead of developing its own local solution from scratch. Figure 1.1 presents a usage example overview of a captcha[3] reusable application: after installing the reusable app, the captcha verification can be directly reused in the project under development. Reusable apps frequently include their own user interfaces (UI) that can be directly adopted in the project front-end. In the captcha example, the captcha UI can be fully reused out-of-the-box.

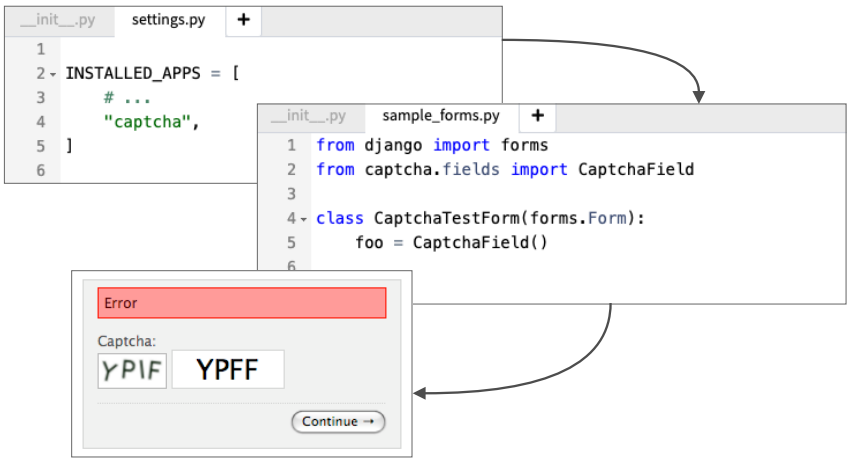


Figure 1.1: Example of a captcha verification reusable app (*mbi/django-simple-captcha*).

In this research, we propose to explore the Django ecosystem due to two major reasons. *First*, despite its relevance, to the best of our knowledge, the Django ecosystem is under-studied. While there are studies covering other software ecosystems, like Android [30, 16, 38], Apache [3, 17, 4], Spring [40, 41], and Eclipse [45, 11], unfortunately, we find no research in the context of the Django ecosystem. Currently, it is not clear what solutions are available to be reused by developers in this important ecosystem, or how active and reliable this ecosystem really is. This information would be important to characterize the Django ecosystem, helping developers to better understand the positive and negative aspects of adopting this framework. Moreover, this would provide the basis for the development of novel tools to overcome possible limitations. *Second*, the literature has largely explored reuse via APIs [51, 56], mostly at the class and method level [10, 49]. However, Django promotes a distinct level of reusability, that is, developers can reuse fully working applications (*i.e.,* the reusable applications) in their projects. We believe that this particular way of reuse brings an interesting and unique case to be further explored.

[3] https://github.com/mbi/django-simple-captcha

1.2 Proposed Work

To overcome this gap, in this dissertation, we propose an empirical study to assess the Django ecosystem. Particularly, we explore the reusable applications and analyze their target domain, level of activity, frequency of tests, and compatibility. For this purpose, we qualitatively and quantitatively analyze 487 reusable applications and propose four research questions:

- *RQ1: What reusable applications are available for developers in the Django ecosystem?* Reusable applications are mostly concentrated in three categories: *developer tools* (27.8%), *user interface* (17.1%), and *persistence* (15.9%). We also find reusable applications related to *security* (9.7%), *communication* (5%), and *performance* (3.7%).

- *RQ2: How active are the reusable applications?* The reusable applications are 7.7 years old and have 10.17 months without commits (on the median). Reusable applications are deprecated due to multiple issues faced by open-source projects, such as *usurped by competitor*, *lack of time of the main contributor*, *based on outdated technologies*, and *lack of interest of the main contributor*. We also find that reusable applications are deprecated because they are *included in main Django framework*, thus, they are not needed anymore as a separate app and are directly available in Django.

- *RQ3: How tested are the reusable applications?* On the median, reusable applications have 0.45 test methods per non-test method. This means that half of the reusable applications have less than one test for each non-test method. However, we find some variation according to the categories, for example, *persistence* (0.51) apps have proportionally more tests than *user interface* ones (0.35).

- *RQ4: How compatible are the reusable applications?* We find that 80% of the reusable applications include the supported Python versions, 64% include the Django versions, and 61% include both Python and Django versions. Upgrading to recent versions of Django and Python may not be supported in some reusable applications. For example, among the reusable applications that support Python 3.5, we find that only 75.44% support Python 3.6 and 64.36% support Python 3.11. However, compatibility issues become less problematic in Django versions 3.2 to 4.1 and in Python versions 3.7 to 3.11.

Overall, we find that the categories of reusable applications have distinct characteristics. *Developer tools* have the highest concentration of young and inactive applications.

In contrast, *persistence* applications are the oldest ones, while *security* ones are the most active. Moreover, *performance* applications have proportionally more tests. We also detect compatibility issues faced by reusable applications that could possibly impact client web applications.

1.3 Contributions

The contributions of this dissertation are twofold: (i) we provide one of the first empirical studies to explore the Django ecosystem both qualitatively and quantitatively and (ii) we propose and discuss implications for practitioners and researchers.

Based on our findings, we discuss six implications for researchers and practitioners. We elaborate on the (1) novel empirical data about the Django ecosystem, which contributes to the software ecosystem literature [30, 16, 38, 3, 17, 4, 40, 41, 45, 11]. Next, we detail three possible problems faced by the Django ecosystem: (2) the low activity level of reusable applications, (3) the possible low frequency of tests in reusable applications, and (4) the compatibility issues of reusable applications. We also discuss (5) the interest of the Django community in three types of reusable applications: tooling, UI, and persistence. We conclude by discussing (6) the fact that relevant reusable applications may be merged in the Django framework.

1.4 Outline of the Dissertation

This dissertation is structured as follows.

- **Chapter 2** introduces the Django framework and the reusable applications.
- **Chapter 3** presents the study design and assesses the research questions.
- **Chapter 4** Presents the results for the proposed research questions. It also presents threats to validity related to these questions.
- **Chapter 5** discusses implications for practitioners and researchers.
- **Chapter 6** concludes this dissertation by presenting an overview and suggestions for future work.

Chapter 2

Background and Related Work

This chapter starts by introducing the Django Framework in Section 2.1. Section 2.2 details the framework's ecosystem and how reusable applications can be integrated into a project. Section 2.3 explores the reuse of out-of-the-box components on other frameworks. Section 2.4 presents the related work, and Section 2.5 presents the final remarks.

2.1 The Django Framework in a Nutshell

Django [20] is an open-source web framework for the Python programming language. It has a large set of built-in features so developers can focus on development tasks [20]. These features include authentication, caching, sending emails, sessions, data validation, pagination, and serialization, to name a few.[1] For example, the framework includes an extensible user authentication system that handles user accounts, groups, permissions, and cookie-based user sessions.[2] Indeed, Django's built-in features possibly contributed to its popularity, which is nowadays the most adopted web framework in Python.

In Django, developers can reuse out-of-the-box applications, which are known as *reusable applications* [19]. Figure 2.1 presents an example of a reusable application that adds a fully working text editor[3] in a Django web application. After installing the reusable application, the text editor can be directly reused in the project. Notice that a reusable application is different from the traditional API reuse at the class and method level [10, 49]. Reusable applications are fully working applications and they frequently include their own user interfaces, with their own model, view, and template layers.

[1] https://docs.djangoproject.com
[2] https://docs.djangoproject.com/en/4.2/topics/auth
[3] https://github.com/summernote/django-summernote

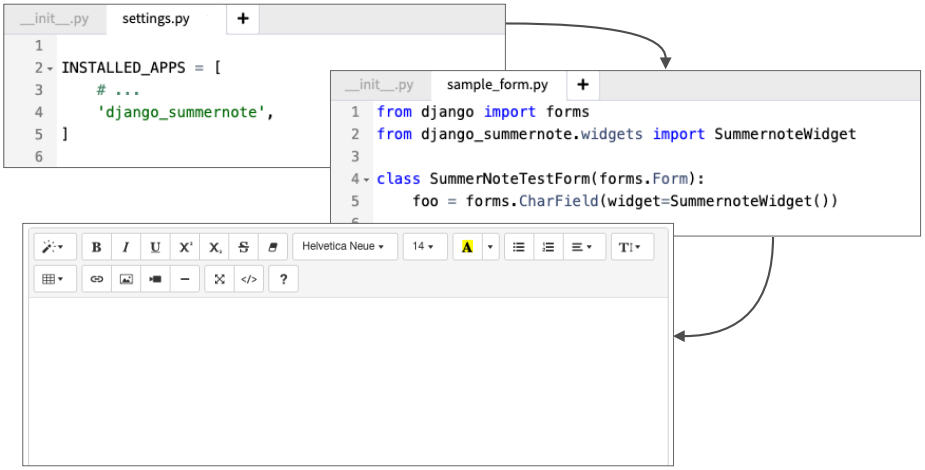


Figure 2.1: Example of a text editor reusable app (*summernote/django-summernote*).

2.2 The Django Ecosystem of Reusable Applications

Django provides a large corpus of reusable applications created by the open-source community. The Django Packages platform [22] provides a solution to group and access this ecosystem of reusable applications. It is a directory of reusable applications, frameworks, and projects built with the Django framework itself. The Django Packages platform lists over 4K reusable applications from diverse categories, like authentication, messaging, and security, to name a few. Figure 2.2 presents an overview of the development process in Django. Web applications are built using the Django framework and can be empowered by reusable applications, which are also built using the Django framework itself.

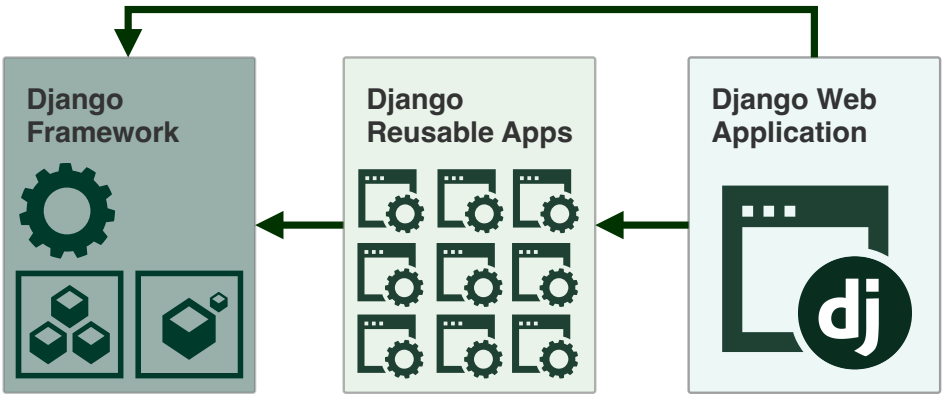


Figure 2.2: Web applications are built using the Django framework and can be empowered by reusable applications, which are also built using the Django framework.

Some reusable applications get enough popularity and notoriety over time to be considered a top choice for their context. For example, the Django Rest Framework[4]

[4] https://www.django-rest-framework.org

is a solution to build web APIs with more than 25K stars on GitHub and is currently being used by companies like Heroku, Mozilla, and RedHat. Other examples of popular reusable applications include ChatterBot[5] (a machine learning, conversational dialog engine for creating chatbots), django-crispy-forms[6] (an app to handle forms), Silk[7] (a live profiling and inspection tool), and Django Filter[8] (a generic system for filtering from URL parameters).

2.3 Reusable Applications in Other Frameworks

Just like the reusable applications in Django, there are other frameworks that provide mechanisms for reuse that are wider than just reusing minor code artifacts like methods and interfaces. As an example, many frameworks focused on front-end development allow users to reuse out-of-the-box user interface components. The import of ready-to-use components speeds up the development and also helps the standardization of the application's design. In addition, users are not only able to export their components to be easily reused across their projects, but they can also rely on libraries that deliver the basic components of famous front-end toolkits and design systems, such as Bootstrap[9] and Material Design.[10] Some of those libraries can be seen in Table 2.1.

Framework	Component Libraries
React[11]	React Bootstrap,[12] MUI,[13] Chakra[14]
Angular[15]	Angular Material,[16] Nebular,[17] PrimeNG[18]
Vue[19]	BootstrapVue,[20] Quasar,[21] Buefy[22]

Table 2.1: Front-end frameworks and some of its popular components libraries.

[5] https://github.com/gunthercox/ChatterBot
[6] https://github.com/django-crispy-forms/django-crispy-forms
[7] https://github.com/jazzband/django-silk
[8] https://github.com/carltongibson/django-filter
[9] https://getbootstrap.com
[10] https://m3.material.io
[11] https://react.dev
[12] https://react-bootstrap.github.io
[13] https://mui.com
[14] https://chakra-ui.com
[15] https://angular.io
[16] https://material.angular.io
[17] https://akveo.github.io/nebular
[18] https://primeng.org
[19] https://vuejs.org
[20] https://bootstrap-vue.org

These components deliver functionalities as ready-to-use to the developers, requiring only the installation of the package and the import of the reference in the main code. The already defined standardized style saves cost and time if the application does not have any specific requirement in this context. Figure 2.3 shows an example of a code using both *React* framework and components from the *React Bootstrap* library. The library provides ready-to-use components such as list items and a progress bar, with an already-defined style.

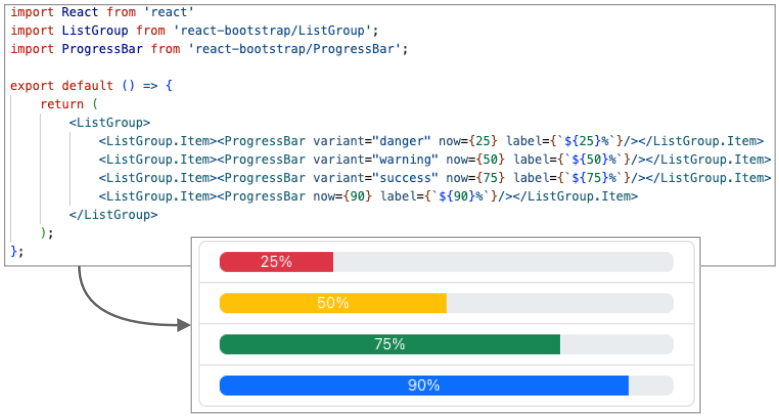


Figure 2.3: Example of reusable components with the *React* framework.

We recall that Django reusable applications are working applications and they may include multiple layers, such as user interfaces, models, views, and templates. Therefore, in some sense, Django reusable applications are more powerful than the reusable applications for React, Angular, and Vue, which mostly focus on user interfaces.

2.4 Related Work

2.4.1 Software Ecosystem

Software ecosystems (SECOs) have been a subject with increasing momentum since early 2000, especially after the work from Messerschmitt *et al.* [42], which defines it as a "*collection of software products that have some given degree of symbiotic relationships*". Since then, the definition of what constitutes a SECO has been discussed in many studies. While some studies focus on the units that are compounding the software ecosystem and

[21] https://quasar.dev
[22] https://buefy.org

the technical relations between them [42, 32] (*e.g.,* shared dependencies and common technology), other studies highlight the set of actors that are related to maintenance [39, 7].

The evolution of the ecosystem itself across time is being analyzed in different contexts [14, 25]. Constantinou and Mens [14] study the Ruby ecosystem over a nine-year period to find growth in many aspects of the ecosystem until early 2014, followed by an increased abandonment rate for both contributors and projects. German *et al.* [25] explores the evolution characteristics of the project GNU R and finds that the ecosystem of user-contributed packages has been growing at a faster rate than the R core packages.

SECOs have also been the subject of analysis in several contexts. One example is the evolution and maintenance inside an ecosystem [11, 3], especially in the mobile context [30, 16, 38]. Bavota *et al.* [3] analyze the evolution of dependencies between projects in the Apache ecosystem and find that developers working in this ecosystem are more attempted to upgrade a dependency when the release involves a substantial number of bug fixes, but tend to avoid it when there are many changes in the API interfaces. Businge *et al.* [11] analyze third-party plugins from the Eclipse ecosystem and find that their success rate of them is related to their dependency on stable and supported Eclipse APIs.

Regarding mobile ecosystems, Huang *et al.* [30] analyze the dependencies on Android applications regarding the upgrade of those dependencies, and find problems hindering easy updates such as deprecated functions, changed data structures, and dependencies between different libraries. Derr *et al.* [16] investigates the library outdatedness problem on Android by conducting a large-scale library updatability analysis of apps and finds that more than 80% of the libraries could be upgraded by at least one version without modifying the application code. Liu *et al.* [38] analyze analytics libraries used in the Android ecosystem, specifically the personal information collected by these libraries, and find that some apps leak users' personal information through analytics libraries.

Many other aspects are analyzed in the context of SECOs, such as technical debt [17] and the usage and maintenance of code samples [40, 41]. Digkas *et al.* [17] explore the evolution of systems from the Apache ecosystem to calculate the trends of the technical debt to find that technical debt normalized to the size of the system tends to decrease over time. Menezes *et al.* [40, 41] provide insights into how code samples are maintained and used by developers in both Android and SpringBoot ecosystems.

Despite the relevance of the Django framework and its software ecosystem to building web applications, to the best of our knowledge, this particular ecosystem is not so well explored in the literature as other ecosystems such as the ones from Android, Eclipse, and Apache. Thus, our findings contribute to the software ecosystem literature by shedding light on multiple aspects of the Django ecosystem.

2.4.2 Framework, Library, and API Reuse

Reuse in software development is a field of study that has been extensively explored at different levels, including frameworks, libraries, and APIs. For example, API evolution [29, 28] and breaking changes [9, 57, 8] that are introduced along the evolution is an important research topic. In this context, Hora *et al.* [29] propose an exploratory study aimed at observing API evolution and its impact in the Pharo software ecosystem to provide a basis to better understand how such impact can be alleviated. Brito *et al.* [8] analyze the evolution of Java libraries and detect breaking changes, finding that most of the breaking changes are motivated by the implementation of new features, improvement of maintainability, and the desire to make the APIs simpler. Xavier *et al.* [57] measure the number of breaking changes on Java libraries and their impact on clients, finding that the frequency of breaking changes increases over time. Brito *et al.* [9] proposes a tool to identify API breaking and non-breaking changes between two versions of Java libraries and report usage scenarios of this tool with real-world Java libraries.

Many other aspects related to software reuse are explored by literature, such as updatability [35, 44] and migration [54, 33, 2]. For instance, Kula *et al.* [35] provide an empirical study regarding the update of library dependencies from Java projects and found that the majority of the systems keep their outdated dependencies. Mirhosseini and Parnin [44] analyze the usage of tools like badges and automated pull requests to warn the developers about stale dependencies on open-source projects and evaluate that these initiatives improve the upgrade of dependencies when compared to other projects. Barbosa and Hora [2] study the migration of testing frameworks in Python and find that most systems are migrating from unittest to pytest. Teyton *et al.* [54] mine library migrations from several open-source projects and provides recommendations for dependencies replacements based on static analysis of source code. Kabinna *et al.* [33] study logging library migrations in the Apache ecosystem through manual analysis of issues and find that the main reasons for logging library migration include the increase in flexibility and performance and reduction of effort spent on code maintenance.

Our study contributes to the corpus of studies in the context of software reuse by exploring the reusable applications of the Django framework.

2.4.3 Activity of Open-Source Projects

Another research topic close to ours includes studies that explore the activities of open-source projects (OSS). The usage of open-source software in the industry has been increasing in the last few decades due to several factors, such as the easiness of contributing and sharing projects with the current platforms. Today, OSS development is not only maintained by dispersed groups of loosely organized individuals but also by companies. Naggle [47] analyzes contributions to the Linux project and runs an estimation framework to estimate the increased productivity returns from the use of OSS, for both contributors and non-contributors. The author finds that contributors gain a productivity benefit from the use of OSS that is up to 100% higher than non-contributors.

Considering the benefits and especially the dependency upon the current open-source software, their survival is extremely important. In this context, some studies measure the deprecation and the survival of OSS projects [37, 23, 1, 43, 12]. Lin *et al.* [37] examine the developer turnover inside OSS from different organizations relating it to the moment of start contributing and to the types of contributions. The authors find that developers have higher chances to continue contributing by starting contributing to the project earlier and focusing on modifying code - instead of creating new code or creating documentation. Foucault *et al.* [23] study the source code of open-source projects to characterize patterns of developer turnover and to determine the effects of turnover on software quality. Avelino *et al.* [1] select popular OSS projects, recover the abandoned and the surviving ones, and conduct a survey with the developers who maintained the surviving projects. The authors find that the developer's usage of the systems is the main motivation to contribute, and lack of time and difficulty in obtaining write access to the code are the main barriers. Calefato *et al.* [12] analyze the inactivity of core developers in OSS organizations through a method that identifies the inactive periods by analyzing their frequency of contributions. Miller *et al.* [43] conduct surveys to identify the reasons and predictive factors behind developer disengagement in OSS. The authors find that factors such as the popularity of the project play a key role and that transitions (*e.g.,* switching jobs) are a common reason to stop contributing. We contribute to this research line by exploring the activity level of the reusable applications provided by the Django community.

2.5 Final Remarks

In this chapter, we discussed the technical background related to this work. We detailed the Django framework and its key characteristics and discussed the relation between the framework and its ecosystem. We analyzed how packages from the Django Packages platform can be integrated into a project built with Django framework. We also discussed how frameworks of other programming languages relate to external reusable applications. Finally, we presented the related work.

Chapter 3

Study Design

This chapter introduces the study design. Section 3.1 presents an overview of the entire process. Section 3.2 presents the steps to collect the packages. Section 3.3 presents the steps to filter the relevant reusable applications among the packages. Section 3.4 details the extraction of code metrics and compatibility data for each one of the reusable apps. Finally, Section 3.5 assesses each one of the research questions and Section 3.6 presents the final remarks.

3.1 Overview

In this study, we aim to explore real-world reusable applications. Therefore, we rely on the Django Packages platform [22] to collect the applications. Figure 3.1 presents an overview of the data collection.

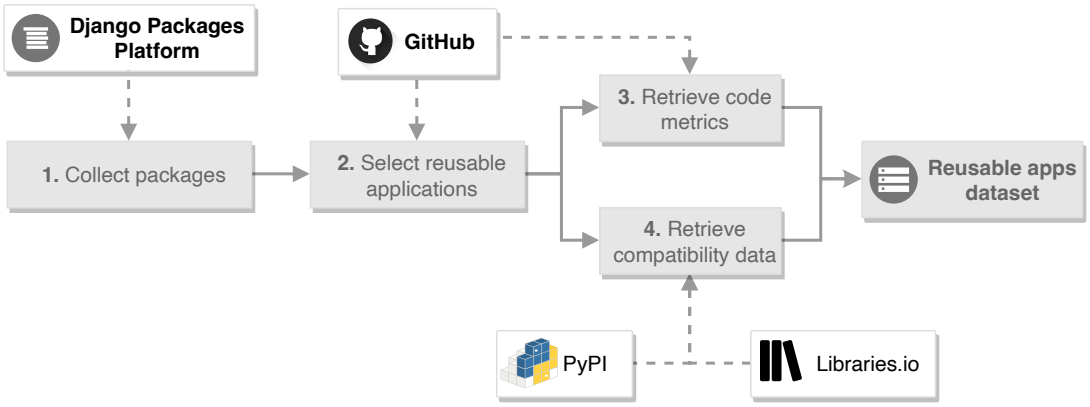


Figure 3.1: Overview of the data collection.

First, we collect a list of all Django packages available in the Django Packages platform. Then, we select the packages that are relevant reusable applications. Next, for each selected reusable application, we extract code metrics from GitHub and retrieve compatibility data from the Python Package Index (PyPi) [48] and Libraries.io [36]. Fi-

nally, we have our dataset of Django reusable applications, which is publicly available at Zenodo.[1] The following sections detail each step.

3.2 Collecting Packages

The first step is to collect the packages available in the Django Packages platform. Overall, the Django Packages platform provides over 4K packages that can be reused by developers. As we will analyze the source code of the selected packages, we filter out packages that do not provide information about the code repository in GitHub. Among all packages, we found 4,060 with valid references to their GitHub repositories.

However, not all packages are necessarily reusable applications. They can be other reusable components to help the Django development in some way. Therefore, we need to carefully select the packages that are indeed reusable applications.

3.3 Selecting Reusable Applications

In this step, we select the packages that are reusable applications. Here, it is important to notice that we are interested in analyzing relevant and real-world reusable applications. For this purpose, we apply multiple automated and manual filtering, as follows.

First, we rely on the categorization proposed by the Django Packages platform itself to select packages that are classified as applications. Thus, we find 3,323 package applications. Second, we clone the repository of those packages and access their documentation to ensure they have clear documentation and can be used as a reusable application by developers. Specifically, we verify whether the documentation files have some reference to the keyword INSTALLED_APPS, which is the recommended (and official) configuration to import and use reusable applications in Django. The files considered as part of documentation are the *readme* and the files with the extension *rst* or *md* in root folders such as *doc* and *docs*. For example, Figure 3.2 presents part of the documentation of the *torchbox/django-recaptcha* reusable application, which clearly presents how to import it via the INSTALLED_APPS configuration. After applying this filtering step, we are left

[1] https://doi.org/10.5281/zenodo.8003371

with 2,365 applications.

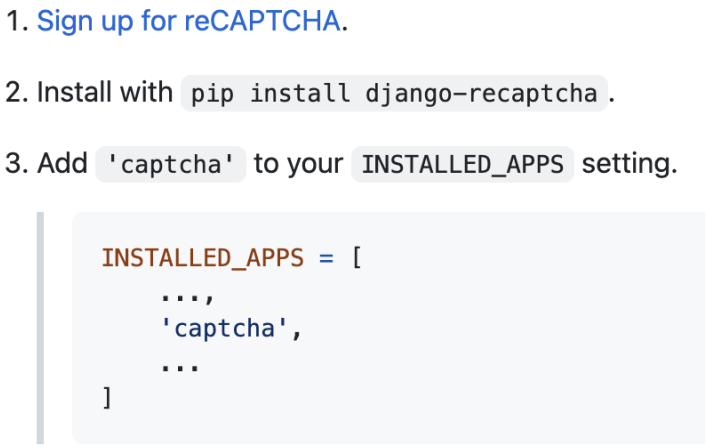


Figure 3.2: Example of a INSTALLED_APPS configuration (*torchbox/django-recaptcha*).

Next, we detect that a few packages are in fact content management systems (CMS). Therefore, we decided to filter out those packages because they are not in the scope of our research. This way, we are left with 2,078 applications. We also apply some filtering to ensure relevant and real-world applications. We select the packages with at least 100 stars to avoid less popular projects, like toy projects and proofs of concept, for example. This filtering is commonly adopted in the software mining literature to remove less popular projects [6, 53]. Here, we also manually inspected the official documentation of each package to ensure it is a reusable application. After this step, we select 547 packages that are reusable applications.

Lastly, we filter out reusable applications that are self-identified as deprecated to avoid assessing abandoned applications. One way to detect the application as deprecated is when one of the authors marks the repository as archived in GitHub, meaning it is no longer actively maintained. The other way is when the repository is not marked as archived but in the description of the repository (*i.e.,* the *readme* file) the authors explicitly mention it is deprecated or no longer active. After manually inspecting the documentation of the applications, we find 487 reusable applications that are not deprecated.

Figure 3.3 summarizes all automated and manual filtering applied to collect reusable applications and ensure they are in fact relevant ones that can be reused by developers in their projects. After all the filtering steps, we select 487 reusable applications that are further explored in this research.

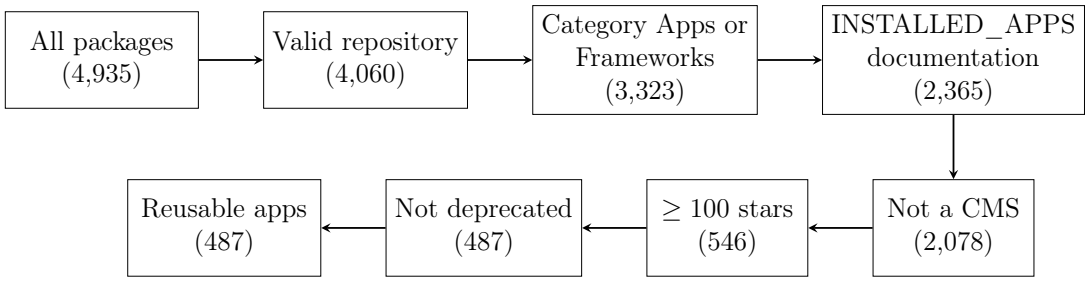


Figure 3.3: Summary of the filtering process to collect the reusable applications.

3.4 Retrieving Code Metrics and Compatibility Data

We clone the 487 selected reusable applications to compute both source file-level and commit-level metrics. The source files are analyzed to extract size metrics (such as the number of lines of code and the number of methods and functions) and test metrics (such as the number of test methods and non-test methods). We also analyze the control version system to extract commit-level metrics such as the number of commits and the date of the latest commit.

Finally, we extract compatibility data of the reusable applications to explore their support for Python and Django. For this purpose, we rely mainly on Pypi, [2] a software repository for the Python programming language. In the case we could not detect the compatibility data in PyPi, we try to extract from Libraries.io,[3] a platform that monitors package releases and indexes data from millions of packages in general. These repositories (Pypi and Libraries.io) allow developers to inform the supported versions of their applications. Fortunately, the Django Packages platform links each application to its respective entry in Pypi (and Pypi links each application to Libraries.io). We rely on this information to retrieve the supported Python and Django versions of the reusable applications. Commonly, versions follow the *semantic versioning*,[4] that is, they are written in the format X.Y.Z, where X indicates major changes, Y indicates minor changes, and Z indicates patches. In this study, we keep the versions in the format X.Y to avoid any noise caused by patch numbers. Thus, versions with patch numbers are mapped to their respective major/minor versions. For example, version 3.10.1 is categorized as 3.10.

[2] https://pypi.org
[3] https://libraries.io
[4] https://semver.org

3.5 Research Questions

3.5.1 RQ1: What reusable applications are available for developers in the Django ecosystem?

In this research question, we explore what reusable applications are provided in the Django ecosystem to be used by client projects. For this purpose, we randomly select 216 reusable applications (*i.e.,* 95% confidence level and 5% confidence interval). For each reusable application, we manually inspect the *readme* file, and, in cases in which the *readme* was unclear, we also explore the official documentation of the reusable application.

We adopt thematic analysis [15] to classify the domain of the reusable applications, with the following steps: (1) initial reading of the readme file/documentation, (2) generating a first code for each application, (3) searching for themes among the proposed codes, (4) reviewing the themes to find opportunities for merging, and (5) defining and naming the final themes. The first three steps were done by the author of this dissertation, while steps 4 and 5 were done together with the advisor until consensus was achieved.

Rationale: We aim to better understand what reusable applications are available in the wild. So far, it is unclear what types of reusable applications are provided to complement the basic features available in Django. For example, the over-concentration of a certain type of reusable application may indicate some limitations of Django that are being overcome by the community.

3.5.2 RQ2: How active are the reusable applications?

This research question is divided into two parts. First, we analyze how active reusable applications are by assessing two metrics: (1) age and (2) lack of activity. We compute age by assessing the number of days between the first and last commit. Lack of activity is computed by assessing the number of days since the last commit.

Another important factor related to activity is deprecation. During the process of collecting reusable applications, we filter out 59 applications that are explicitly deprecated. This high number of deprecated apps motivated us to better understand the reasons behind it. Among the 59 reusable applications, we find that 33 are either marked as archived in GitHub or had a short notice about the deprecation in the documentation,

but did not provide any detailed reason. Two reusable applications (*umap-project/django-leaflet-storage* and *Bearle/django-private-chat*) are defined as deprecated due to a change to a new repository, *i.e.,* the project is still under development, but in another repository. The remaining 24 applications do provide a detailed reason for the deprecation. Thus, in the second part, we explore the reasons behind the deprecation of these 24 reusable applications. Coelho and Valente [13] categorized reasons for the failure of open-source projects and grouped them into three categories: reasons related to the development team (*e.g., lack of time*), reasons related to project characteristics (*e.g., the project is obsolete* or *outdated technologies*), and reasons related to the environment (*e.g., usurpation by competition*). We also rely on thematic analysis [15] to classify the deprecation rationales and take into account the nomenclature adopted in prior work [13] for comparison purposes.

Rationale: Measuring the activity level can be seen as a proxy of how much developers can trust reusable applications. Reusable applications that are abandoned (that is, not explicitly deprecated, but with a long period without any code change) are probably not well suited for updates and also do not have a good perspective in terms of new functionalities and bug fixes. Moreover, the analysis of the reasons for deprecation can bring insights into why reusable applications are discontinued as compared to general open-source projects [13].

3.5.3 RQ3: How tested are the reusable applications?

This research question provides an overview of the presence of tests in reusable applications. For each reusable application, we analyze the number of test methods divided by the total number of non-test methods. We analyze the ratio of tests to avoid any inaccurate assumption guided by absolute numbers, *e.g.,* well-tested small projects with the same number of tests as larger projects that lack tests. A test method is a method with the prefix `test_` and that is located in a path with the substring *test*. All other methods outside the paths with substring *test* are considered non-test methods.

Rationale: Developers working with the Django framework are likely to include these reusable applications in the build of their applications, thus, they need to trust that the applications are working well. Tests are considered one important quality practice to ensure software applications are well prepared for production usage [24, 5], thus, measuring the presence of tests in the reusable applications can bring insights into their trustfulness.

3.5.4 RQ4: How compatible are the reusable applications?

This research question is divided into two parts. First, we analyze the Python and Django versions supported by the reusable applications. We extract the versions of the reusable applications from their entry in PyPi or Libraries.io, as detailed in Section 3.4. In the second part, we explore the compatibility of the reusable applications. We consider versions that are available in the official Django documentation (see Table 3.1) [21]. That is, the Django versions 2.2, 3.1, 3.2, 4.0, and 4.1 and the Python versions from 3.5 to 3.11.

Django Versions	Python Versions
2.2	3.5, 3.6, 3.7, 3.8 (added in 2.2.8), 3.9 (added in 2.2.17)
3.1	3.6, 3.7, 3.8, 3.9 (added in 3.1.3)
3.2	3.6, 3.7, 3.8, 3.9, 3.10 (added in 3.2.9)
4.0	3.8, 3.9, 3.10
4.1	3.8, 3.9, 3.10, 3.11 (added in 4.1.3)

Table 3.1: Relation of which Python versions are available for each Django framework version, extracted from the Django documentation [21].

Rationale: The supported versions in terms of both programming language and framework are fundamental for developers when deciding whether a dependency (*e.g.,* a reusable application) will be adopted or not. In this context, compatibility of the reusable applications to Django and Python may directly affect the evolution of their dependent web applications. For example, suppose that a developer decides to build a web application on top of Django 4.0 and relies on a specific reusable application that supports Django 4.0 but not Django 4.1. Due to the version compatibility of the dependency, the developer will not be able to upgrade the web application to rely on Django 4.2. Indeed, version update is also important in the aspect of security, since third-party outdated dependencies are more likely to have security issues [16, 30, 31].

3.6 Final Remarks

In this chapter, we detailed the process designed to build the dataset of 487 Django reusable applications. We presented each step of the data collection which raised 4,935 packages, each filter applied - so as the motivations behind it - and how external data such as code metrics and compatibility data were collected. Finally, we assessed each of the research questions and presented the rationale for each of them.

Chapter 4

Results

In this chapter, we present the results for each one of the proposed research questions. Section 4.1 presents the domain of the reusable applications, so as the distribution of the selected sample among these domains. Section 4.2 brings the level of activity of the reusable applications and the reasons behind the deprecations. Section 4.3 presents the level of tests of the reusable applications. Section 4.4 analyzes both Python and Django versions supported by the reusable applications and explores their compatibility. Then, we present the threats to validity in Section 4.5 and a summary of the findings as final remarks in Section 4.6.

4.1 RQ1: What reusable applications are available for developers in the Django ecosystem?

Figure 4.1 presents the results of our manual classification of the reusable applications. The most frequent category of reusable applications is *developer tools* (27.8%), followed by *user interface* (17.1%) and *persistence* (15.9%). We also find reusable applications related to *security* (9.7%), *communication* (5%), and *performance* (3.7%). Next, we detail and present examples for each category.

Developer Tools. The most frequent category includes reusable applications to support developers in their development tasks, like debugging, deployment, and caching, to name a few. For example, the application *jazzband/django-debug-toolbar* provides panels that display debug information, while *ehmatthes/django-simple-deploy* offers a solution to deploy Django applications. The application *mwarkentin/django-watchman* exposes a status endpoint for backend services like databases and caching, while *jazzband/django-hosts* provides dynamic and static host resolving.

User Interface. The second most frequent category contains reusable applications related to the creation of user interfaces, like forms, editors, menus, navigation, and pagi-

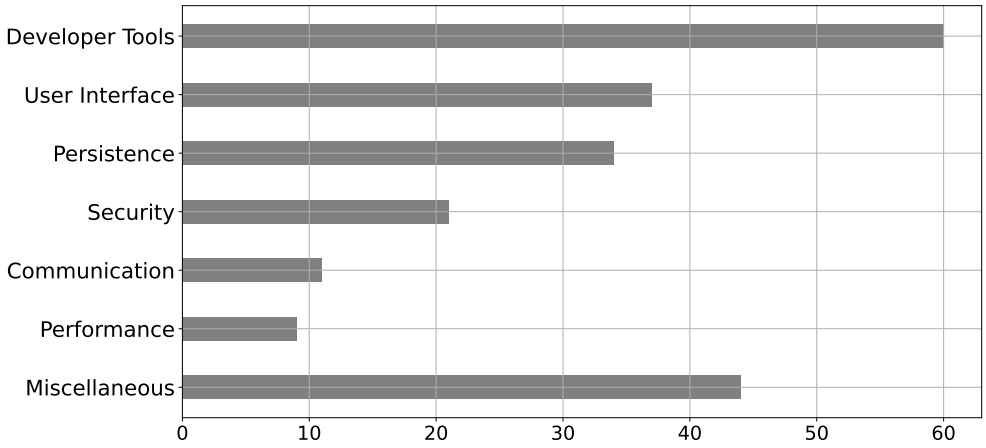


Figure 4.1: Categories of reusable applications.

nation. The application *jazzband/django-tinymce* contains a widget to render form fields as a rich-text editor. The application *rossp/django-menu* provides a basic structure for building multiple navigation menus, while *shtalinberg/django-el-pagination* provides tools for endless and lazy pagination. The application *fabiocaccamo/django-admin-interface* is a customizable interface for Django's administration painel [18].

Persistence. The third most frequent category includes reusable applications to deal with persistence. To name a few examples, the application *scholrly/neo4django* provides integration between Neo4j[1] databases and Django models, while *incuna/django-pgcrypto-fields* supports data encryption and decryption using a PostgreSQL[2] extension. The application *jazzband/django-recurrence* provides fields for working with recurring dates, while *ulule/django-linguist* offers a persistent way to manage translations for Django models.

Security. This category contains reusable applications that deal with security concerns, like authentication, authorization, and security vulnerabilities. For instance, the application *django-auth-ldap/django-auth-ldap* provides a backend that authenticates against Lightweight Directory Access Protocol (LDAP), while *merixstudio/django-trench* provides endpoint APIs to support multi-factor authentication. The application *dmpayton/django-admin-honeypot* presents a fake admin login page to log and notify admins of attempted unauthorized access, while *jazzband/django-axes* allows tracking suspicious login attempts.

Communication This category includes reusable applications related to all sorts of communication services, like messaging, notification, and email. For instance, the application *jazzband/django-newsletter* manages multiple mass-mailing lists, *pinax/pinax-messages* provides a user-to-user threaded messaging, and *v1k45/django-notify-x* provides a notification system.

Performance. It contains reusable applications related to increasing performance, managing tasks, or improving automated processes. The application *peterbe/django-fancy-*

[1]Neo4j: https://neo4j.com
[2]PostgreSQL: https://www.postgresql.org

cache provides a decorator to set caching for requests, and *opus10/django-pgpubsub* presents a framework for building asynchronous and distributed message processing network. The application *chrisspen/django-chroniker* allows managing cron jobs via the Django administration section, and *rq/django-rq* provides integration with the Redis[3] queue.

Miscellaneous. Finally, this category includes reusable applications with very specific goals that do not clearly fit in any of the previous categories. For example, the application *django-getpaid/django-getpaid* is a payment processing framework, *adamcharnock/django-tz-detect* allows the Django application to detect the user's timezone, and *tomwalker/django_quiz* provides a configurable quiz application for Django.

> **RQ1 Summary:** Reusable applications are mostly concentrated in three categories: *developer tools* (27.8%), *user interface* (17.1%) and *persistence* (15.9%). We also find reusable applications related to *security* (9.7%), *communication* (5%), and *performance* (3.7%). This highlights that the overall interest of the community is mostly in tooling, UI, and persistence.

4.2 RQ2: How active are the reusable applications?

This research question has two parts. First, we analyze how active are reusable applications by assessing two metrics: age and lack of activity. Second, we explore the reasons some reusable applications are deprecated over time.

4.2.1 Activity Analysis

Figure 4.2 presents the age distribution (in years) of the reusable applications, according to the categories defined in RQ1. On the median, we note that the reusable applications have 7.7 years. Overall, the categories with the oldest applications are *performance* (8.99 years), *persistence* (8.69 years), and *user interface* (8.5 years), while the category with the youngest ones is *developer tools* (5.75 years).

Figure 4.3 presents a complementary view of the age distribution. It classifies each reusable application as young (age is up to the first quartile), medium (age is between

[3] https://redis.io

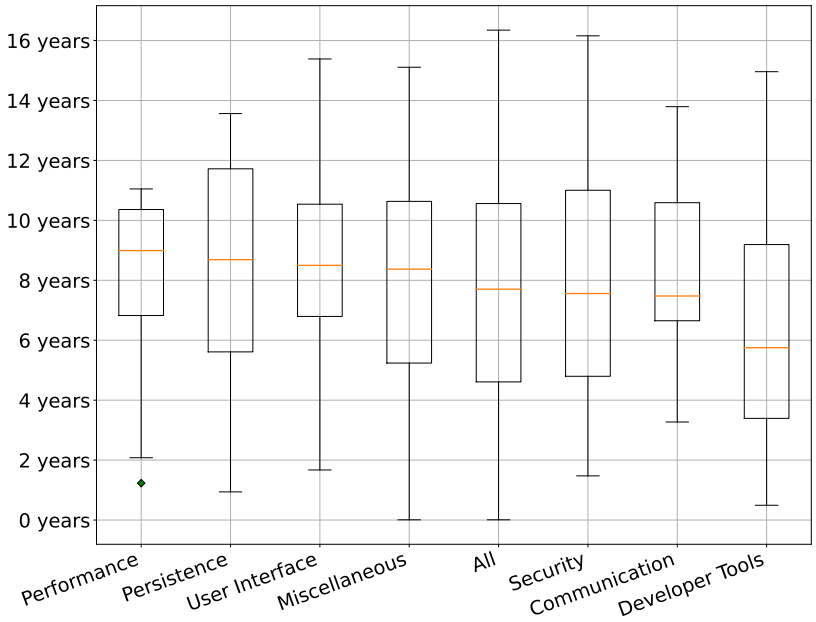


Figure 4.2: Age distribution of the reusable applications (in years).

the first and the third quartile), and old (age is greater than the third quartile). Here, we notice that *developer tools* have the highest concentration of young applications (33.3%), while *persistence* has the highest concentration of old applications (44.1%).

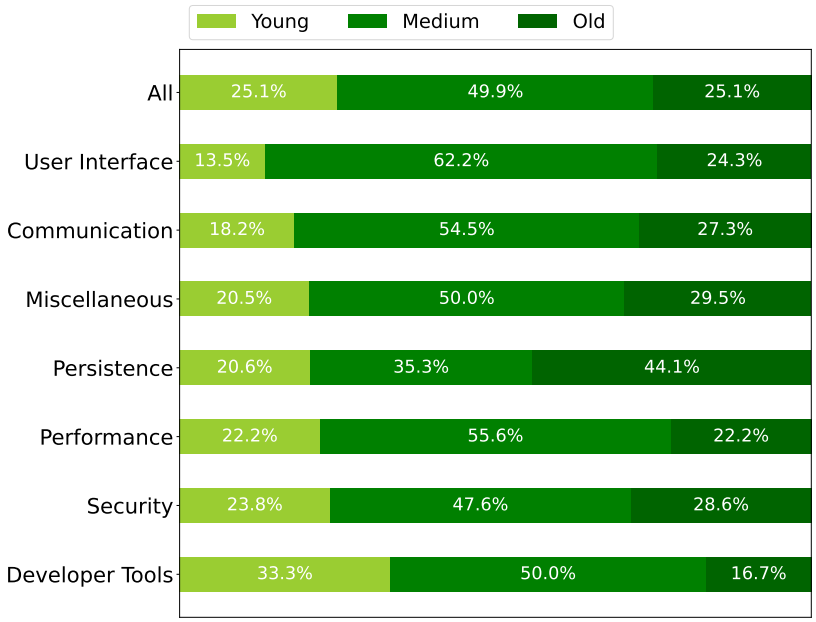


Figure 4.3: Summary of the age of the reusable applications.

Next, we explore the lack of activity in the reusable applications. This metric is defined as the number of months since the last commit. Figure 4.4 presents the lack of activity distribution. On the median, the reusable applications have 10.17 months without commits. Overall, the categories with the most months without commits are *miscellaneous* (17 months), *communication* (15.37 months), and *user interface* (11.97 months). On the

other hand, *security* has the applications with the least months without commits (7.83 months).

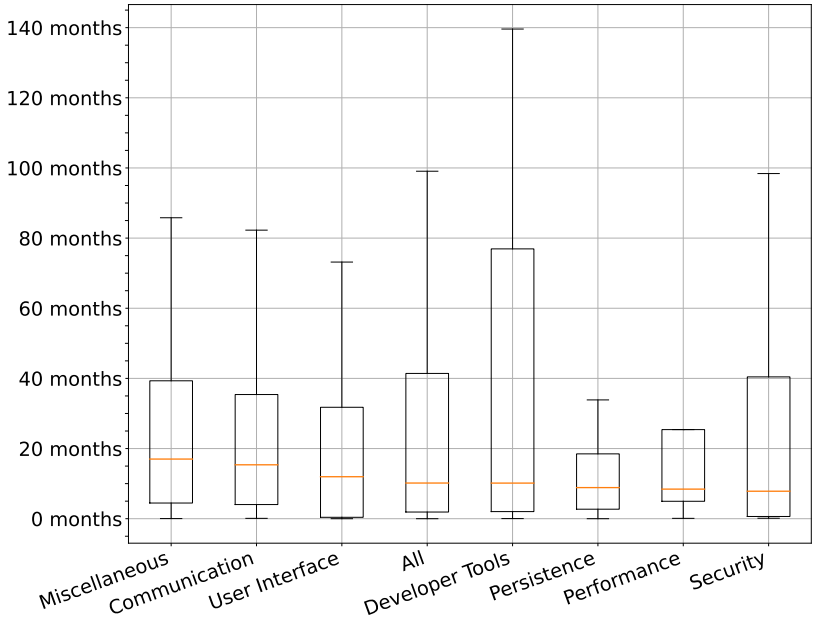


Figure 4.4: Lack of activity distribution of the reusable applications (in months).

Figure 4.5 presents a complementary view of the lack of activity distribution. It classifies each reusable application as very active (lack of activity is up to the first quartile), medium (lack of activity is between the first and the third quartile), and inactive (lack of activity is greater than the third quartile). We can observe that *developer tools* have the highest concentration of inactive applications (31.7%), while *security* has the highest concentration of active applications (38.1%).

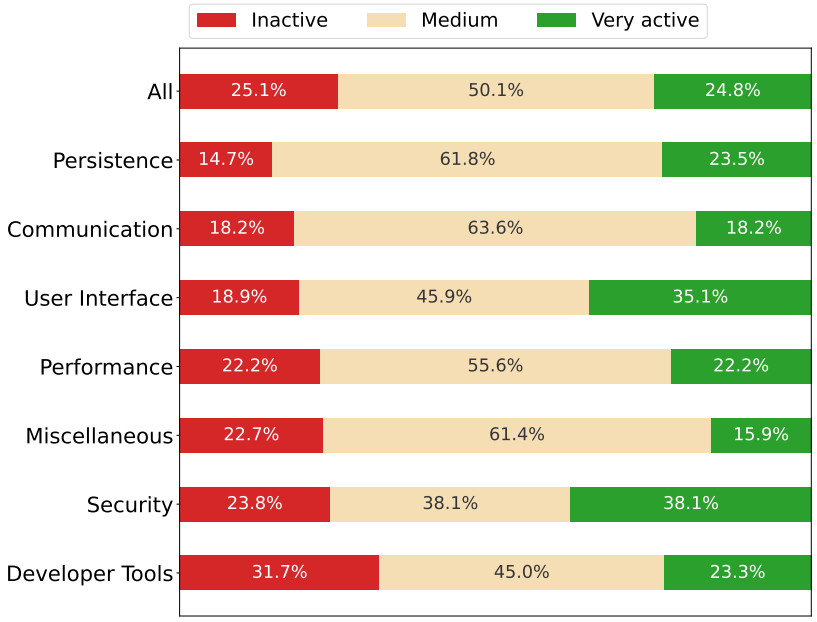


Figure 4.5: Summary of the lack of activity of the reusable applications.

RQ2 Summary (part 1): The reusable applications are 7.7 years old and have 10.17 months without commits (on the median). The category *developers tools* has the highest concentration of young and inactive applications. In contrast, *persistence* has the oldest ones, while *security* has the most active ones.

4.2.2 Deprecation Analysis

We have manually inspected 24 reusable applications with detailed reasons for the deprecation and grouped them into five categories: *usurped by competitor* (8 cases), *lack of time of the main contributor* (6 cases), *based on outdated technologies* (4 cases), *included in main Django framework* (4 cases), and *lack of interest of the main contributor* (2 cases). The results are summarized in Figure 4.6. Four out of five reasons are similar to a previous study that explored the reasons behind project deprecation [13]: *usurped by competitor*, *lack of time*, *lack of interest*, and *based on outdated technologies*. The category *included in main Django framework* is novel and specific to our analysis. Next, we detail each category and provide examples.

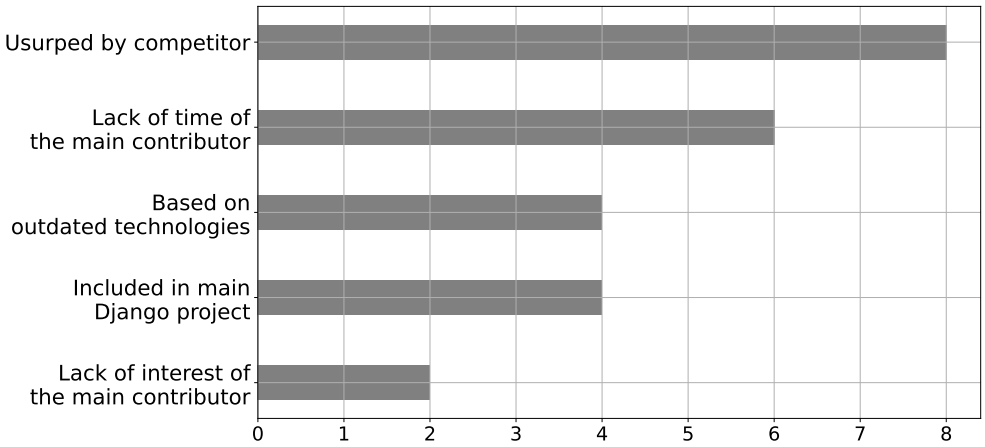


Figure 4.6: Reasons to deprecate reusable applications.

Usurped by competitor. This category represents the reusable applications that were deprecated in favor of another one. Examples of apps in this category include *omab/django-social-auth*, *jbalogh/jingo*, and *mpasternak/django-monitio*, which include a deprecation warning linking to the competitor app. For example, the app *omab/django-social-auth* has the following deprecation note: "*This library is deprecated in favor of python-social-auth*". Another case that was also considered for this category is when one fork ends up becoming the official repository of the project, like in the reusable appli-

cation *aschn/drf-tracking*, which includes the following note: "*Work has moved to a fork to continue on https://github.com/lingster/drf-api-tracking*". Interestingly, the category *usurped by competitor* is most frequent in our analysis and also in a prior analysis that investigated deprecation in open-source projects [13].

Lack of time of the main contributor: This category happens when the author do not have available time to maintain the reusable app. Notice that lack of of time is a well known problem in open-source projects [43, 12], and reusable apps are not different. Examples of apps in this deprecation category include *no-dice/django-bootstrap-themes*, *byashimov/django-controlcenter*, and *jazzband/django-admin-sortable*. For example, the author of app *byashimov/django-controlcenter* provides the following deprecation note: "*Unfortunately, I have no time to add new features*".

Based on outdated technologies: The reusable applications in this category are related to specific technologies that became deprecated, outdated, or obsolete over time. Consequently, the reusable application development was also suspended. Examples of apps in this category include *jmcclell/django-bootstrap-pagination*, *jrief/django-angular*, and *dyve/django-bootstrap-toolkit*. For example, the app *jrief/django-angular* has the following deprecation note: "*Since AngularJS is deprecated now, this library shall not be used for new projects anymore. Instead please proceed with my follow-up project django-formset*". Similarly, the app *dyve/django-bootstrap-toolkit* provides the following message: "*The time for development on Bootstrap v2 has passed. This app is provided 'as is', and will not be updated. Everyone using Django and Bootstrap is encouraged to upgrade to django-bootstrap3*".

Included in main Django Project: These reusable applications were deprecated because they were merged into the Django main code. Therefore, the client developers of these applications do not need to depend on them anymore because are directly available in Django. Next, we present the four reusable applications and their respective deprecation notes:

- *carljm/django-secure*: Reusable app to improve your Django site's security: Deprecation note: "*This project was merged into Django 1.8, and is now unsupported and unmaintained as a third-party app*".
- *elky/django-flat-theme*: Reusable app to improve the Django Admin interface. Deprecation note: "*django-flat-theme is included as part of Django from version 1.9*".
- *elky/django-flat-responsive*: Extension for Django admin. Deprecation note: "*django-flat-responsive is included as part of Django from version 2.0*".
- *jezdez/django-discover-runner*: A test runner based on unittest test discovery. Deprecation note: "*This runner has been added to Django 1.6 as the default test runner. If you use Django 1.6 or above you don't need this app*".

Lack of interest of the main contributor: It happens for projects that are abandoned because the developers do not have interest in their maintenance, for example, due to a job change. The dataset contains the reusable applications *antonagestam/collectfast* and *zhangfisher/DjangoUeditor*.

RQ2 Summary (part 2): Reusable applications are deprecated due to multiple issues faced by open-source projects, such as *usurped by competitor*, *lack of time of the main contributor*, *based on outdated technologies*, and *lack of interest of the main contributor*. Interestingly, we also find that reusable applications are deprecated because they are *included in main Django framework*, thus, they are not needed anymore as a separate app and are directly available in Django.

4.3 RQ3: How tested are the reusable applications?

We explore to what extent the reusable applications have tests. Figure 4.7 presents the distribution of the proportion of test methods by the non-test methods. On the median, the reusable applications have 0.45 test methods per non-test method. Overall, the categories with the highest proportion of test methods are *persistence* (0.51), *communication* (0.49), and *developer tools* (0.48), while the category with the lowest proportion is *user interface* (0.35).

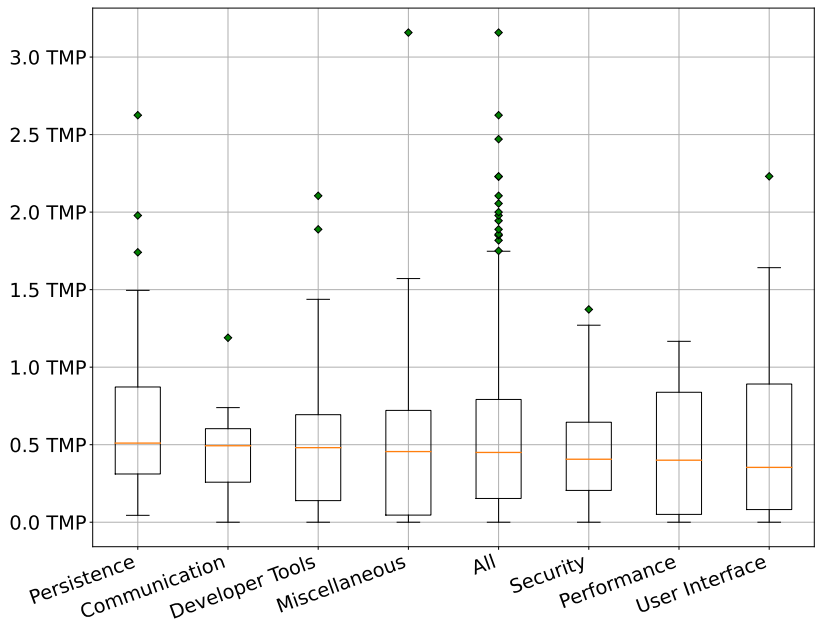


Figure 4.7: Distribution of the proportion of test methods of the reusable applications.

Figure 4.8 presents a complementary view of the distribution of the proportion of test methods. It classifies each reusable application in terms of low (proportion is up to the first quartile), medium (proportion is between the first and the third quartile), or high (proportion is greater than the third quartile) amount of tests. Here, we notice that the *miscellaneous* category has the highest concentration of applications with a low number of tests (36.4%), while *performance* has the highest concentration of applications with a high number of tests (33.3%).

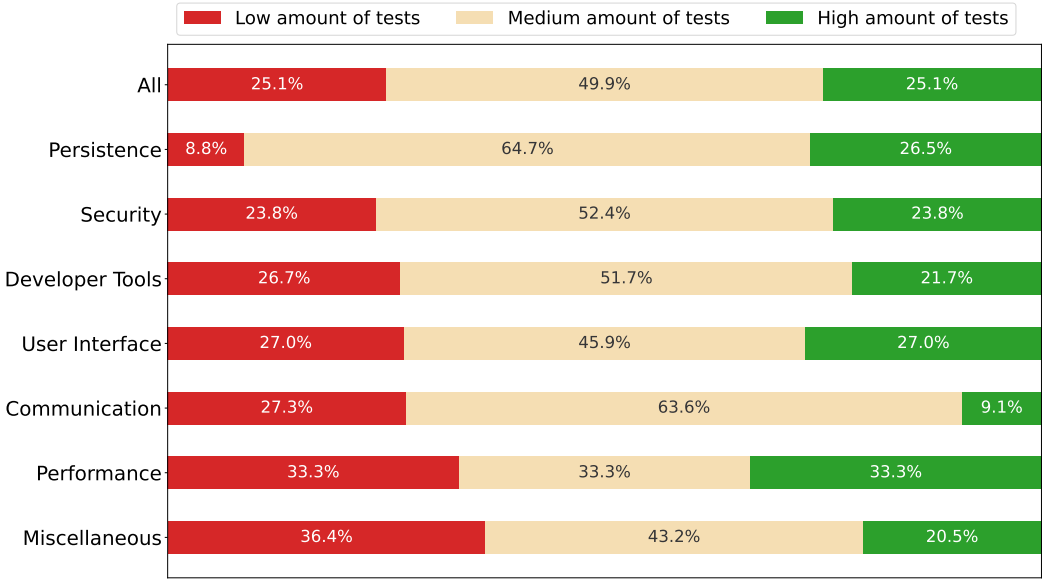


Figure 4.8: Summary of the proportion of test methods of the reusable applications.

RQ3 Summary: On the median, reusable applications have 0.45 test methods per non-test method. This means that half of the reusable applications have less than one test for each non-test method. However, we find some variation depending on the application category, for example, *persistence* (0.51) apps have proportionally more tests than *user interface* ones (0.35).

4.4 RQ4: How compatible are the reusable applications?

This final research question is divided into two parts. First, we analyze the Python and Django versions supported by the reusable applications. Second, we explore the compatibility of the reusable applications.

4.4.1 Supported Versions

Figure 4.9 presents the percentage of reusable applications that inform the supported Python and Django versions. We notice that 80% of the reusable applications inform their supported Python versions, while Django versions are found in 64%. Moreover, close to 83% of reusable applications present Python or Django versions, while 61% present both Python and Django versions.

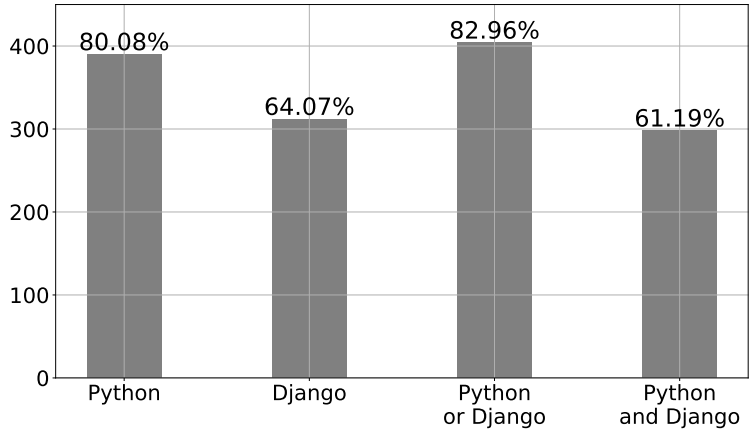


Figure 4.9: Summary of reusable applications that inform the supported versions.

Figure 4.10 details the supported Python versions of the reusable applications. We present the support for seven Python versions, from Python 3.5 to 3.11. The most supported versions are 3.7 to 3.11, all with over 60% of support, while the old versions 3.5 and 3.6 are the most unsupported ones. Figure 4.11 summarizes the support for five Django versions: 2.2, 3.1, 3.2, 4.0, and 4.1. In this case, the most supported Django versions are 3.2 and 4.0, while versions 2.2, 3.1, and 4.1 are the most unsupported ones.

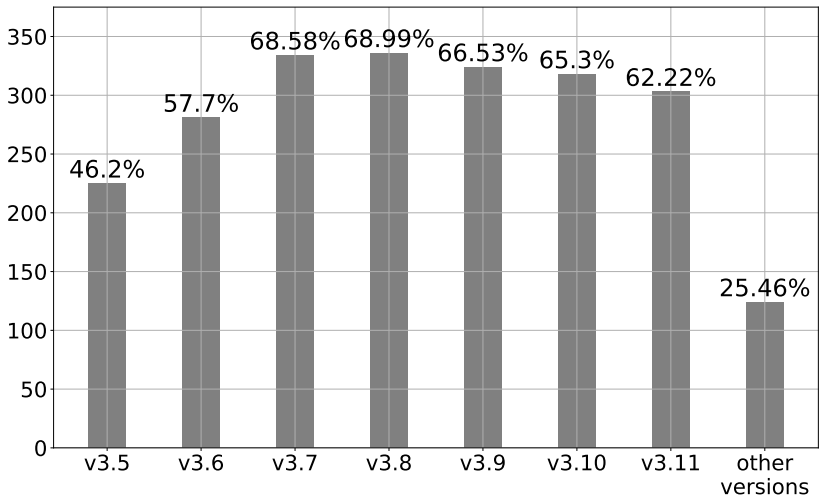


Figure 4.10: Supported Python versions.

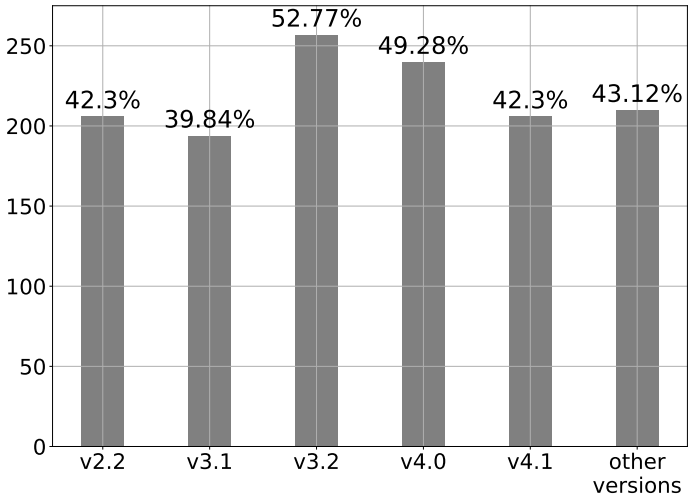


Figure 4.11: Supported Django versions.

RQ4 Summary (part 1): We find that 80% of the reusable applications include the supported Python versions, 64% include the Django versions, and only 61% include both Python and Django versions. The most supported Python versions are 3.7 to 3.11, while the most unsupported ones are 3.5 and 3.6. The most supported Django versions are 3.2 and 4.0, while the most unsupported ones are 2.2, 3.2, and 4.1.

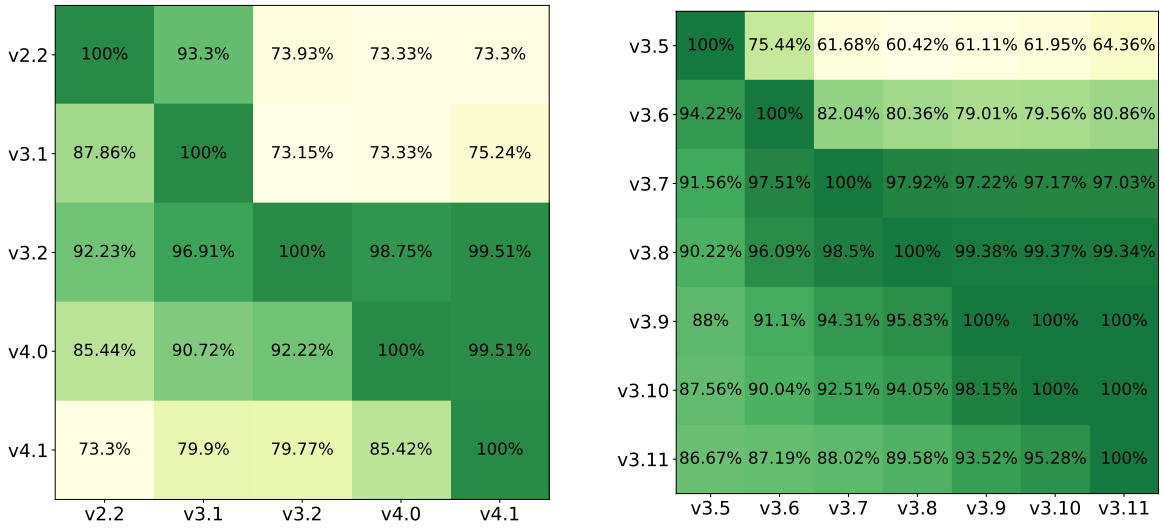
4.4.2 Compatibility

Besides identifying the Python and Django versions supported by the reusable applications, another important aspect is related to the evolution of web applications that depend on reusable applications. Since reusable applications are used as a complementary project, they can be a blocker for the evolution of their dependent web applications. For example, suppose that a developer decides to build a web application on top of Django 3.2 and relies on a specific reusable application that supports Django 3.2 but not Django 4.0. Due to the version compatibility of the dependency, the developer will not be able to upgrade the web application to rely on Django 4.0.

Figure 4.12a and Figure 4.12b summarize this problem for the Python and Django versions supported by the reusable applications. In this matrix, each axis presents the supported versions. Considering i as the version represented on each row, and j as the version represented in each column, each cell presents the percentage of reusable applications that support version i that also supports version j. For example, in the Django

matrix, consider the cell 93.3% presented in the first row (v2.2) and second column (v3.1). This indicates that among the reusable applications that support Django 2.2, 93.3% also support Django 3.1. Overall, by checking the support for subsequent versions, we notice that the compatibility tends to decay. For instance, among the reusable applications that support Django 2.2, 73.93% support Django 3.2, 73.33% support Django 4.0, and 73.3% support Django 4.1. Notice, however, that this compatibility issue becomes less problematic from version 3.2 to 4.1. For example, considering the applications that support Django 3.2 or 4.0, we find that 99.51% support Django 4.1.

The same analysis can be performed in the Python matrix. For instance, consider the cell 75.44% presented in the first row (v3.5) and second column (v3.6). In this case, among the reusable applications that support Python 3.5, 75.44% also support Python 3.6. Like in Django, the Python support for subsequent versions also tends to decay. For instance, considering again the reusable applications that support Python 3.5 (first row), only 64.36% support Python 3.11 (last column). As in Django, this compatibility issue becomes less problematic at some point. In this case, from the Python version 3.7 to 3.11. For example, considering the reusable applications that support Python 3.7, we detect that 97.03% support Python 3.11. Moreover, among the apps that support Python 3.9 or 3.10, we find that 100% support Python 3.11.



(a) Django versions **(b)** Python versions

Figure 4.12: Compatibility matrix of Django and Python supported versions.

The previous analysis presented that upgrading to recent versions of Django and Python may not be supported in some reusable applications. However, it is not clear if this migration impediment comes from Django and Python themselves. To investigate this issue more deeply, we further explore the possible compatibility between Django and Python.

Figure 4.1[4] presents the Python versions supported by each Django version, as provided by the Django documentation [21]. For example, Django 3.1 is available on Python 3.6, 3.7, 3.8, and 3.9, while Django 3.2 is available on Python 3.6, 3.7, 3.8, 3.9, and 3.10. Therefore, if a reusable application claims to support Django 3.1 and Django 3.2, ideally, it should provide support to their respective Python versions, which is in this case: Python 3.6, 3.7, 3.8, 3.9, and 3.10 (this would be the *ideal* support). However, in practice, it is unclear to what extent the reusable applications support the available Python versions according to their Django versions. For instance, considering the previous example, a reusable application could provide support to Python 3.9 and 3.10 only (this would be the *real* support).

Django Versions	**Python Versions**
2.2	3.5, 3.6, 3.7, 3.8 (added in 2.2.8), 3.9 (added in 2.2.17)
3.1	3.6, 3.7, 3.8, 3.9 (added in 3.1.3)
3.2	3.6, 3.7, 3.8, 3.9, 3.10 (added in 3.2.9)
4.0	3.8, 3.9, 3.10
4.1	3.8, 3.9, 3.10, 3.11 (added in 4.1.3)

Table 4.1: Relation of which Python versions are available for each Django framework version, extracted from the Django documentation [21].

To better explore this problem, for each reusable application and its supported Django versions, we computed the ratio of supported Python versions (real support) to the available Python versions (ideal support). In this metric, 1 means that the reusable application supports all available Python versions, while 0 means that it supports none. Figure 4.13 presents the distribution of the ratio of supported Python versions to the available Python versions. On the median, the reusable applications support 86% of the available Python versions. The first quartile is 67%, while the third quartile is 100%.

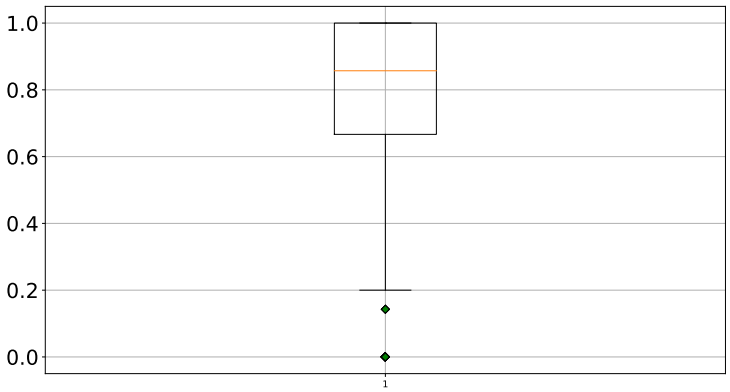


Figure 4.13: Distribution of the ratio of supported Python versions (real support) to the available Python versions (ideal support) in the analyzed reusable application.

[4]This table is already presented in our Study Design. We repeat it here for convenience.

RQ4 Summary (part 2): Upgrading to recent versions of Django and Python may not be supported in some reusable applications. For example, among the reusable applications that support Python 3.5, we find that only 75.44% support Python 3.6 and 64.36% support Python 3.11. However, compatibility issues become less problematic in Django versions 3.2 to 4.1 and in Python versions 3.7 to 3.11. Lastly, we find that, on the median, the reusable applications support 86% of the available Python versions.

4.5 Threats to Validity

Selecting reusable applications. The Django Package platform categorizes the Django packages in *apps*, *frameworks*, *projects*, and *other*. In this study, we selected the category *apps* to detect reusable applications as it is defined as "*small components used to build projects*". We also included the apps from the category *framework*, which is defined as "*large efforts that combine many python modules or apps*". Therefore, as this category could also potentially contain reusable applications, we included it in our filtering process to avoid losing some relevant applications. However, it is important to note that we conducted a manual analysis of the dataset to make sure that all packages are reusable applications.

Availability of the reusable applications in the Django Packages platform. Regarding the packages used to build our dataset, a first concern is that we can not be sure that every reusable application from the Django ecosystem is listed in the Django Packages platform. Despite acknowledging that some applications might not be registered, the Django Packages platform is an initiative with more than a decade of history and is well known in the Django community, which gives confidence regarding the provided applications.

Manual classification of the reusable applications. In RQ1 we manually classified the reusable applications according to their domain, such as *developer tools*, *user interface*, and *persistence*. We acknowledge that the definition of these categories involves a certain degree of subjectivity. To minimize the subjectivity of the manual classification, we adopted thematic analysis [15].

Generalization of the results. In this study, we assess real-world reusable applications of the Django framework's ecosystem. Django and Python are among the most popular web frameworks and programming languages nowadays. Several big tech companies are currently using the Django framework to build their applications, and consequently,

possibly relying on the reusable applications available in this ecosystem. Despite these observations, our findings – as usual in empirical software engineering studies – may not be directly generalized to other ecosystems and programming languages. Further studies should be performed in the future to address other software ecosystems.

4.6 Final Remarks

In this chapter, we presented the results to assess the usage of reusable apps in the Django ecosystem. The dataset of 487 reusable applications was quantitatively analyzed in several aspects, a randomly defined sample of 216 was manually categorized, and a set of 59 deprecated packages was individually assessed. We highlight the following findings:

- *Categorization*: Packages that aim to help developers in a development task are the most often subject in the dataset, followed by packages related to user interface and persistence.
- *Activity*: Packages identified as deprecated in the Django ecosystem tend to be abandoned without detailed explanations from the authors, while the rising of alternative options is the most common reason provided. Almost half (49,3%) of the Django reusable apps had their last commit more than 10 months before the mining of the data.
- *Tests*: More than half (55%) of the reusable applications have less than 0.5 test methods per productive method. When grouping by categories, applications related to *Persistence* are slightly better in this aspect.
- *Compatibility*: Packages supporting older versions such as Django framework 2.2 and 3.1 have less compatibility support for the subsequent versions of the framework. The same applies to packages supporting versions like Python 3.5 and 3.6, with less compatibility support for the subsequent version of the programming language.

Chapter 5

Discussion

In this chapter, we assess the results that are presented in the previous chapter and the implications for both practitioners and researchers. Section 5.1 explores some empirical data about the Django ecosystem. Section 5.2 and Section 5.3 discuss, respectively, the level of activity and the level of tests of the reusable applications. Section 5.4 analyzes the compatibility issues of the reusable applications. Section 5.5 explores the high-level domains and the interest of the community of the Django ecosystem. Section 5.6 highlights the occurrences of reusable applications being merged into the source code of the Django framework, and then, we present a summary of the findings as final remarks in Section 5.7.

5.1 Novel empirical data about the Django ecosystem

Despite the relevance of the Django ecosystem, to the best of our knowledge, it is under-studied by the literature. While there are studies covering other software ecosystems, like Android [30, 16, 38], Apache [3, 17, 4], Spring [40, 41], and Eclipse [45, 11], unfortunately, we find no research in the context of Django.

In this research, we address four important aspects of the Django ecosystem: target domain, activity level, frequency of tests, and compatibility. In RQ1, we found that reusable applications are mostly concentrated in three categories: *developer tools* (27.8%), *user interface* (17.1%) and *persistence* (15.9%). Overall, we find that the categories of reusable applications have distinct characteristics. For example, *developer tools* have the highest concentration of young and inactive apps, while *persistence* applications are the oldest ones. In RQ2, we detected that reusable applications are 7.7 years old but have 10.17 months without commits (on the median). Moreover, they are deprecated due to multiple issues, such as *usurped by competitor*, *lack of time of the main contributor*, *based on outdated technologies*, *included in main Django framework*, and *lack of interest of the main contributor*. In RQ3, we found that 55% of the reusable applications have less than 0.5 test methods per non-test method. Lastly, RQ4 showed that 80% of the reusable

applications include the supported Python versions, 64% include the Django versions, and 61% include both Python and Django versions. Moreover, upgrading to recent versions of Django and Python may not be supported in some reusable applications. Therefore, our findings contribute to the software ecosystem literature [30, 16, 38, 3, 17, 4, 40, 41, 45, 11] by shedding some light on multiple important aspects of the Django ecosystem and their possible impact on client applications.

5.2 Low activity-level of reusable applications

The lack of evolution of a software system can affect client applications in several ways such as bugs and security breaches remaining as not solved, new requirements raised from context changes not being addressed, blocking of other component's versions upgrades, etc. This stagnation can be caused by an explicit deprecation and by the project abandonment [13].

In our analysis, we found that 33 projects were deprecated without providing details and 2 projects were moved to new repositories. We also detected that 24 reusable applications were explicitly deprecated, including reasons such as *usurped by competitor*, *lack of time*, *lack of interest*, and *based on outdated technologies*, which is in line with prior literature [13].

When there is no explicit notice of deprecation, the definition of an open-source project as abandoned is not trivial. Some previous studies followed the strategy of setting a threshold that ranges from 2 months to 1 year without activity to decide if the project is not being maintained [14, 37, 23, 1]. In RQ2, we found that almost half (49,3%) of the Django reusable applications had their last commit more than 10 months before the data collection. Therefore, it is fair to say that a considerable number of reusable applications are suspected of being abandoned, which clearly should be a concern for practitioners and the overall Django community.

5.3 Possible low frequency of tests in reusable applications

RQ3 shows that, on the median, reusable applications have 0.45 test methods per non-test method. This means that half of the reusable applications have less than one test for each non-test method. However, we find some variation according to the categories, for example, *persistence* (0.51) apps have proportionally more tests than *user interface* ones (0.35). Of course, not all methods of the application should necessarily be tested, for example, there may exist trivial, debug-only, and non-runnable code that do not need tests [26, 27]. Considering that software testing is a vital practice to ensure software quality [24, 5], our results about the overall low frequency of tests in reusable applications may be a concern for the community. Further studies are needed to deeply explore this issue, for example, metrics like line and branch coverage can be used to better gauge the overall test quality.

5.4 Compatibility issues of reusable applications

Always relying on updated versions of third-party dependencies is a best practice important for accessing new features, bug fixes, and security fixes [16, 30, 31]. However, RQ4 showed that reusable applications that provide support to certain versions of Django and Python are more problematic to update to subsequent versions. Particularly, the reusable applications that support Django versions 2.2 and 3.1 and Python versions 3.5 and 3.6 are more critical and deserve attention from developers. For example, from the applications supporting Django 3.1, only 73.15% support Django 3.2. This means that over 25% of these reusable applications would block the evolution of the web application in case developers want to upgrade to start using Django version 3.2. Notice, however, that this compatibility issue becomes less problematic in recent Django (3.2 to 4.1) and Python versions (3.7 to 3.11). For example, considering the applications that support Django 3.2 or 4.0, we find that 99.51% support Django 4.1. Similarly, among the apps that support Python 3.9 or 3.10, we find that 100% support Python 3.11. This analysis can help practitioners when selecting reusable applications. It is important to verify the features provided by the reusable applications, but also their available support to recent versions of Django and Python. This can avoid future problems such as the need to fully rewrite parts of the web application to use a new reusable application to unblock a

framework or programming language upgrade.

5.5 Interest of the Django community: tooling, UI, and persistence

In RQ1, we found that reusable applications are mostly concentrated in three categories that reflect the interest of the Django community. The most frequent category is *developer tools* (27.8%), which includes reusable applications to support developers in their development tasks, like debugging, deployment, and caching, to name a few. It is important to recall that *developer tools* have the highest concentration of young (and inactive) applications, which may reinforce its relevance for the community but short life-cycle. The second most frequent category is *user interface* (17.1%), which contains reusable applications related to the creation of user interfaces, like forms, editors, menus, navigation, and pagination. The third most frequent category is *persistence* (15.9%), which includes reusable applications to deal with persistence. This shows that the overall interest of the community is in tooling, UI, and persistence. These results shed some light on the possible directions the Django framework can evolve to better embrace the needs of the community.

5.6 Reusable applications may be merged in the Django framework

In RQ2, we also find that some reusable applications are *included in main Django framework*, thus, they are not needed anymore as a separate app and are directly available in Django. This is an interesting finding, suggesting that a reusable application may become vital for the ecosystem, being integrated into the main project. This shows that the Django framework itself is evolving to embrace the community's needs, showing that the Django maintainers are open-minded to accommodate relevant reusable applications.

5.7 Final Remarks

In this chapter, we discussed the results presented in the previous chapter and some implications for both practitioners and researchers regarding the outcomes of this work. We discussed aspects such as compatibility, activity, tests, and target domain, as its intersections, and also explored topics such as the interest of the community according to our findings and the evolution of the framework by merging reusable applications into the main framework's code.

Chapter 6

Conclusion

This chapter concludes this dissertation. We provide an overview of the study in Section 6.1 and we propose future work in Section 6.2.

6.1 Overview

In this dissertation, we proposed an empirical study to assess the Django ecosystem. We explored the reusable applications and analyzed their target domain, level of activity, frequency of tests, and compatibility. We qualitatively and quantitatively analyzed 487 reusable applications provided by the Django Packages platform and answered four research questions. Our main findings can be summarizes as follows:

- *RQ1: What reusable applications are available for developers in the Django ecosystem?* Reusable applications are mostly concentrated in three categories: *developer tools* (27.8%), *user interface* (17.1%), and *persistence* (15.9%). We also find reusable applications related to *security* (9.7%), *communication* (5%), and *performance* (3.7%).

- *RQ2: How active are the reusable applications?* The reusable applications are 7.7 years old and have 10.17 months without commits (on the median). Reusable applications are deprecated due to multiple issues faced by open-source projects, such as *usurped by competitor*, *lack of time of the main contributor*, *based on outdated technologies*, and *lack of interest of the main contributor*. We also detected that reusable applications are deprecated because they are *included in main Django framework*, thus, they are not needed anymore as a separate app and are directly available in Django.

- *RQ3: How tested are the reusable applications?* On the median, reusable applications have 0.45 test methods per non-test method. This means that half of the reusable applications have less than one test for each non-test method. However,

we found some variation according to the categories, for example, *persistence* (0.51) apps have proportionally more tests than *user interface* ones (0.35).

- *RQ4: How compatible are the reusable applications?* We found that 80% of the reusable applications include the supported Python versions, 64% include the Django versions, and 61% include both Python and Django versions. Upgrading to recent versions of Django and Python may not be supported in some reusable applications. However, compatibility issues become less problematic in recent Django (3.2 to 4.1) and Python versions (3.7 to 3.11).

Finally, we discussed six implications for researchers and practitioners. We elaborated on the (1) novel empirical data about the Django ecosystem, which contributes to the software ecosystem literature We detailed three possible problems faced by the Django ecosystem: (2) the low activity level of reusable applications, (3) the possible low frequency of tests in reusable applications, and (4) the compatibility issues of reusable applications. We also discussed (5) the interest of the Django community in three types of reusable applications: tooling, UI, and persistence. We concluded by discussing (6) the fact that relevant reusable applications may be merged in the Django framework.

6.2 Future Work

As future work, we plan to perform more qualitative analysis to better understand *why* some reusable applications are more required by the community, for example, *developer tools*, *user interface*, and *persistence*. We also plan to better understand from the developer's perspective the reasons to adopt and avoid reusable applications. Regarding tests, we plan to better explore how well-tested are the reusable applications by computing metrics like line and branch coverage. Finally, we plan to perform a client-side analysis to assess how the reusable applications are used in practice.

References

[1] Guilherme Avelino, Eleni Constantinou, Marco Tulio Valente, and Alexander Serebrenik. On the abandonment and survival of open source projects: An empirical investigation. In *13th International Symposium on Empirical Software Engineering and Measurement (ESEM)*, pages 1–11, 2019.

[2] Lívia Barbosa and Andre Hora. How and why developers migrate python tests. In *International Conference on Software Analysis, Evolution and Reengineering (SANER)*, pages 538–548. IEEE, 2022.

[3] Gabriele Bavota, Gerardo Canfora, Massimiliano Di Penta, Rocco Oliveto, and Sebastiano Panichella. How the apache community upgrades dependencies: an evolutionary study. *Empirical Software Engineering*, 20:1275–1317, 2015.

[4] Gabriele Bavota, Gerardo Canfora, Massimiliano Di Penta, Rocco Oliveto, and Sebastiano Panichella. The evolution of project inter-dependencies in a software ecosystem: The case of apache. In *2013 IEEE International Conference on Software Maintenance*, pages 280–289, 2013.

[5] Kent Beck. *Test-driven development: by example*. Addison-Wesley Professional, 2003.

[6] Hudson Borges, Andre Hora, and Marco Tulio Valente. Understanding the factors that impact the popularity of GitHub repositories. In *32nd IEEE International Conference on Software Maintenance and Evolution (ICSME)*, pages 334–344, 2016.

[7] Jan Bosch and Petra Bosch-Sijtsema. From integration to composition: On the impact of software product lines, global development and ecosystems. *Journal of Systems and Software*, 83(1):67–76, 2010.

[8] Aline Brito, Marco Tulio Valente, Laerte Xavier, and Andre Hora. You broke my code: understanding the motivations for breaking changes in apis. *Empirical Software Engineering*, 25:1458–1492, 2020.

[9] Aline Brito, Laerte Xavier, Andre Hora, and Marco Tulio Valente. APIDiff: Detecting API breaking changes. In *25th International Conference on Software Analysis, Evolution and Reengineering (SANER), Tool Track*, pages 507–511, 2018.

[10] Raymond P. L. Buse and Westley Weimer. Synthesizing api usage examples. In *2012 34th International Conference on Software Engineering (ICSE)*, pages 782–792, 2012.

[11] John Businge, Alexander Serebrenik, and Mark van den Brand. Survival of eclipse third-party plug-ins. In *2012 28th IEEE International Conference on Software Maintenance (ICSM)*, pages 368–377, 2012.

[12] Fabio Calefato, Marco Aurelio Gerosa, Giuseppe Iaffaldano, Filippo Lanubile, and Igor Steinmacher. Will you come back to contribute? investigating the inactivity of oss core developers in github. *Empirical Software Engineering*, 27(3):76, 2022.

[13] Jailton Coelho and Marco Tulio Valente. Why modern open source projects fail. In *25th International Symposium on the Foundations of Software Engineering (FSE)*, pages 186–196, 2017.

[14] Eleni Constantinou and Tom Mens. Socio-technical evolution of the ruby ecosystem in github. In *2017 IEEE 24th International Conference on Software Analysis, Evolution and Reengineering (SANER)*, pages 34–44, 2017.

[15] Daniela S Cruzes and Tore Dyba. Recommended steps for thematic synthesis in software engineering. In *International Symposium on Empirical Software Engineering and Measurement*, pages 275–284, 2011.

[16] Erik Derr, Sven Bugiel, Sascha Fahl, Yasemin Acar, and Michael Backes. Keep me updated: An empirical study of third-party library updatability on android. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, pages 2187–2200. ACM, 2017.

[17] Georgios Digkas, Mircea Lungu, Alexander Chatzigeorgiou, and Paris Avgeriou. The evolution of technical debt in the apache ecosystem. In *Software Architecture: 11th European Conference, ECSA 2017, Canterbury, UK, September 11-15, 2017, Proceedings 11*, pages 51–66. Springer, 2017.

[18] Django. Django Admin. https://docs.djangoproject.com/en/4.2/ref/contrib/admin, October, 2023.

[19] Django. Django Applications. https://docs.djangoproject.com/en/4.2/intro/reusable-apps, October, 2023.

[20] Django. Django framework. https://djangoproject.com, October, 2023.

[21] Django. FAQ: What Python version can I use with Django? https://docs.djangoproject.com/en/4.1/faq/install/#what-python-version-can-i-use-with-django, October, 2023.

[22] Django Packages Org. Django Packages. https://djangopackages.org, October, 2023.

[23] Matthieu Foucault, Marc Palyart, Xavier Blanc, Gail C. Murphy, and Jean-Rémy Falleri. Impact of developer turnover on quality in open-source software. In *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering*, pages 829–841. ACM, 2015.

[24] Martin Fowler. *Refactoring.* Addison-Wesley Professional, 2018.

[25] Daniel M. German, Bram Adams, and Ahmed E. Hassan. The evolution of the r software ecosystem. In *2013 17th European Conference on Software Maintenance and Reengineering*, pages 243–252, 2013.

[26] Andre Hora. What code is deliberately excluded from test coverage and why? In *International Conference on Mining Software Repositories (MSR)*, pages 392–402. IEEE, 2021.

[27] Andre Hora. Excluding code from test coverage: Practices, motivations, and impact. *Empirical Software Engineering*, 1:1–36, 2023.

[28] Andre Hora, Anne Etien, Nicolas Anquetil, Stephane Ducasse, and Marco Tulio Valente. APIEvolutionMiner: Keeping API evolution under control. In *IEEE Conference on Software Maintenance, Reengineering and Reverse Engineering (CSMR-WCRE), Tool Demonstration Track*, pages 420–424, 2014.

[29] Andre Hora, Romain Robbes, Marco Tulio Valente, Nicolas Anquetil, Anne Etien, and Stephane Ducasse. How do developers react to API evolution? a large-scale empirical study. *Software Quality Journal*, 26(1):161–191, 2018.

[30] Jie Huang, Nataniel Borges, Sven Bugiel, and Michael Backes. Up-to-crash: Evaluating third-party library updatability on android. In *2019 IEEE European Symposium on Security and Privacy (EuroS&P)*, pages 15–30, 2019.

[31] Kaifeng Huang, Bihuan Chen, Congying Xu, Ying Wang, Bowen Shi, Xin Peng, Yijian Wu, and Yang Liu. Characterizing usages, updates and risks of third-party libraries in java projects. *Empirical Softw. Engg.*, 27(4), 2022.

[32] Slinger Jansen, Anthony Finkelstein, and Sjaak Brinkkemper. A sense of community: A research agenda for software ecosystems. In *2009 31st International Conference on Software Engineering-Companion Volume*, pages 187–190. IEEE, 2009.

[33] Suhas Kabinna, Cor-Paul Bezemer, Weiyi Shang, and Ahmed E. Hassan. Logging library migrations: A case study for the apache software foundation projects. In *Proceedings of the 13th International Conference on Mining Software Repositories*, pages 154–164. ACM, 2016.

[34] Dino Konstantopoulos, John Marien, Mike Pinkerton, and Eric Braude. Best principles in the design of shared software. In *International Computer Software and Applications Conference*, pages 287–292, 2009.

[35] Raula Gaikovina Kula, Daniel M German, Ali Ouni, Takashi Ishio, and Katsuro Inoue. Do developers update their library dependencies? an empirical study on the impact of security advisories on library migration. *Empirical Software Engineering*, 23:384–417, 2018.

[36] Libraries IO. Libraries IO. https://libraries.io, October, 2023.

[37] Bin Lin, Gregorio Robles, and Alexander Serebrenik. Developer turnover in global, industrial open source projects: Insights from applying survival analysis. In *2017 IEEE 12th International Conference on Global Software Engineering (ICGSE)*, pages 66–75, 2017.

[38] Xing Liu, Jiqiang Liu, Sencun Zhu, Wei Wang, and Xiangliang Zhang. Privacy risk analysis and mitigation of analytics libraries in the android ecosystem. *IEEE Transactions on Mobile Computing*, 19(5):1184–1199, 2020.

[39] Mircea Lungu. Towards reverse engineering software ecosystems. In *2008 IEEE International Conference on Software Maintenance*, pages 428–431, 2008.

[40] Gabriel Menezes, Bruno Cafeo, and Andre Hora. Framework code samples: How are they maintained and used by developers? In *2019 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)*, pages 1–11. IEEE, 2019.

[41] Gabriel Menezes, Bruno Cafeo, and Andre Hora. How are framework code samples maintained and used by developers? the case of android and spring boot. *Journal of Systems and Software*, 1:1–30, 2021.

[42] David G Messerschmitt, Clemens Szyperski, et al. *Software ecosystem: understanding an indispensable technology and industry*, volume 1. MIT press Cambridge, 2003.

[43] Courtney Miller, David Gray Widder, Christian Kästner, and Bogdan Vasilescu. Why do people give up flossing? a study of contributor disengagement in open source. In *Open Source Systems: 15th IFIP WG 2.13 International Conference, OSS 2019, Montreal, QC, Canada, May 26–27, 2019, Proceedings 15*, pages 116–129. Springer, 2019.

[44] Samim Mirhosseini and Chris Parnin. Can automated pull requests encourage software developers to upgrade out-of-date dependencies? In *2017 32nd IEEE/ACM*

International Conference on Automated Software Engineering (ASE), pages 84–94, 2017.

[45] Kazunori Mizushima and Yasuo Ikawa. A structure of co-creation in an open source software ecosystem: A case study of the eclipse community. In *2011 Proceedings of PICMET '11: Technology Management in the Energy Smart World (PICMET)*, pages 1–8, 2011.

[46] Simon Moser and Oscar Nierstrasz. The effect of object-oriented frameworks on developer productivity. *Computer*, 29(9), 1996.

[47] Frank Nagle. Learning by contributing: Gaining competitive advantage through contribution to crowdsourced public goods. *Organization Science*, 29(4):569–587, 2018.

[48] Python Package Index. PyPi. https://pypi.org, October, 2023.

[49] Dong Qiu, Bixin Li, and Hareton Leung. Understanding the api usage in java. *Information and software technology*, 73:81–100, 2016.

[50] Steven Raemaekers, Arie van Deursen, and Joost Visser. Measuring software library stability through historical version analysis. In *2012 28th IEEE International Conference on Software Maintenance (ICSM)*, pages 378–387, 2012.

[51] Mohamed Aymen Saied, Ali Ouni, Houari Sahraoui, Raula Gaikovina Kula, Katsuro Inoue, and David Lo. Improving reusability of software libraries through usage pattern mining. *Journal of Systems and Software*, 145:164–179, 2018.

[52] Widura Schwittek and Stefan Eicker. A study on third party component reuse in java enterprise open source software. In *Proceedings of the 16th International ACM Sigsoft Symposium on Component-Based Software Engineering*. ACM, 2013.

[53] Hudson Silva and Marco Tulio Valente. What's in a github star? understanding repository starring practices in a social coding platform. *Journal of Systems and Software*, 146:112–129, 2018.

[54] Cédric Teyton, Jean-Rémy Falleri, Marc Palyart, and Xavier Blanc. A study of library migrations in java. *Journal of Software: Evolution and Process*, 26(11):1030–1052, 2014.

[55] Jeffrey Voas and George Hurlburt. Third-party software's trust quagmire. *Computer*, 48(12):80–87, 2015.

[56] Wei Wang and Michael W. Godfrey. Detecting api usage obstacles: A study of ios and android developer questions. In *2013 10th Working Conference on Mining Software Repositories (MSR)*, pages 61–64, 2013.

[57] Laerte Xavier, Aline Brito, Andre Hora, and Marco Tulio Valente. Historical and impact analysis of API breaking changes: A large scale study. In *24th International Conference on Software Analysis, Evolution and Reengineering (SANER)*, pages 138–147, 2017.