CHALMERS
UNIVERSITY OF TECHNOLOGY

UNIVERSITY OF GOTHENBURG

Software development bot ecosystems

Master's thesis in Computer science and engineering

Dimitrios Platis

Department of Computer Science and Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
UNIVERSITY OF GOTHENBURG
Gothenburg, Sweden 2021

Master's thesis 2021

Software development bot ecosystems

Dimitrios Platis

UNIVERSITY OF
GOTHENBURG

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Computer Science and Engineering
Chalmers University of Technology
University of Gothenburg
Gothenburg, Sweden 2021

Software development bot ecosystems
Dimitrios Platis

Supervisor: Francisco Gomes de Oliveira Neto, Computer Science and Engineering
Examiner: Gregory Gay, Computer Science and Engineering

Master's Thesis 2021
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Typeset in LaTeX
Gothenburg, Sweden 2021

Software development bot ecosystems
Dimitrios Platis
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg

Abstract

Bots that facilitate software development, or DevBots, are increasing their presence and popularity in software projects. As the development scales, so does the number of DevBots as well as the complexity of their interactions with other DevBots, humans and assets of the project. This circumstance insinuates the need for a new perspective on DevBots, that of an ecosystem. Gaining inspiration from ecosystems in software and biology, this study proposes a faceted taxonomy that includes four DevBot roles, or species, when viewed through an ecosystem prism: (i) Keystones, (ii) Niches, (iii) Dominators and (iv) Observers. The taxonomy is validated with a utility demonstration enabled via a case study conducted in an automotive supplier of autonomous driving software.
Furthermore, the different DevBot ecosystems present in the company are described and are found to surround specific assets related to software development, such as the source code, the hardware resources and the simulations.
Next, testimonies from practitioners are utilized to determine what motivated them to contribute to the development of the DevBot ecosystems, the challenges they encountered, the reasons behind the emergence of Dominator DevBots and a collection of best practices when designing DevBot ecosystems. The discovered motivations and challenges not only verify findings from the existing literature but also complement them by introducing new perspectives. When it comes to knowledge around challenges, the study improves the current state-of-art by taking into consideration interactions of DevBots with other DevBots as well as assets of the system.
Finally, common software development best practices should be followed when developing DevBot ecosystems, with one of the additional requirements being the means to systematically manage dependencies so to avoid needless DevBot execution as well as to facilitate debugging by establishing known baselines.

Keywords: Software development, bots, software ecosystems, case study, taxonomy

Acknowledgements

First and foremost, I would like to thank my academic supervisor Francisco Gomes de Oliveira Neto for the tremendous support, help and faith in this thesis and me. This thesis would not have been such a pleasure to work with unless it was for him. Next, I would like to express my gratitude to my employer for agreeing to devote all necessary resources for conducting the case study, as well as all the colleagues who readily participated in the interviews, despite the very short notice. Their input was particularly insightful and instrumental. Last but not least, I would like to help my manager and industrial supervisor, Hans Djurfeldt, for being continuously available to help with the "logistics" and expedite the progress of this paper. He has been a catalyst for completing this study.

Dimitrios Platis, Gothenburg, June 2021

Contents

List of Figures

List of Tables

1

Introduction

Software is becoming the primary differentiating factor in an increasing number of industries. Naturally, this brings higher attention to the software development process and new ways to increase its efficiency [2] and effectiveness are being continuously devised.

Towards that goal, software bots that assist development, while often having human-like traits and using the development team's communication channels, have emerged. We adopt the term "DevBots" [1] to refer to them and their adoption is spreading from open source projects [3] to industrial state-of-the-art products involving large numbers of individuals and organizations.

In the meantime, with the emergence of autonomous mobility, vehicles becoming increasingly connected to the Internet, infrastructure or other vehicles, additional revenue streams that heavily rely on software have come within the reach of the traditionally hardware-centric automotive industry [4]. This paradigm shift [5], has lead the car manufacturers, as well as their suppliers, to enhance their roles as software-makers. Therefore they have begun adopting software engineering practices and tools customary to different industries, such as Continuous Integration (CI) and DevBots.

When multiple DevBots start being utilized in a project, interesting roles, interdependencies and patterns start to appear, reminiscent of software ecosystems and systems of software agents. Gaining a deeper understanding of this phenomenon by drawing parallels and identifying any unique behavior, may improve how DevBots are used in an ecosystem context.

While a consensus in the literature, on a concrete definition about the concept of DevBots has yet to be reached [6], there is a number of recent studies that aim to characterize DevBots. That being said, in contemporary software projects the need to combine multiple DevBots into the same software development process begins to emerge and so are the dependencies between them and their surrounding environment, such as project resources and of course humans. This growing number of complicated relationships related to DevBots, increases the need for DevBots to be viewed from an ecosystem perspective just as businesses or software with similar traits are.

The current state of the art in software bot literature does not provide a characterization of DevBots ecosystems. Despite DevBots being recognized as distinct entities from other software bots [1] no research has been conducted from a system of agents or an ecosystem perspective. This raises the concern that both researchers, as well as practitioners, may not know how to create DevBot ecosystems, how to utilize them effectively and how to design DevBots that collaborate with other DevBots. Moreover, coordination over shared assets is reminiscent of Software Ecosystems, however, it is currently unclear how much we can correlate the two concepts and take advantage of existing expertise.

The goal of this study is to describe DevBots from an ecosystem perspective, and to determine their shared assets, stakeholders as well as record different concerns when it comes to the development of DevBot ecosystems. To achieve this three research questions are proposed:

1. What is the role of the different DevBot species in the ecosystem?
2. How can we describe the DevBot ecosystems?
3. How can the development of DevBot ecosystems be described?

We respond to the first research question by proposing a new taxonomy for DevBot roles in an ecosystem. Then we conduct a qualitative case study at an automotive supplier that develops and delivers the software stack for autonomous driving functions. The company has been using a plethora of DevBots to facilitate different development activities, to a large extent within Continuous Integration. The case study aims to validate the taxonomy by identifying the theorized roles in an industrial setting. Next, we delve further into the different ecosystems present in the company and we describe them by focusing on the different shared assets between the DevBots. Additionally, we conduct an interview study with practitioners to document the developers' perspective on DevBot development and what does developing DevBot ecosystems entails.

The proposed taxonomy introduces 4 "species" of DevBots, when viewing DevBots from an ecosystem perspective: (i) Keystone, (ii) Niche, (iii) Dominator and (iv) Observer DevBots. They differ in their primary purpose, as well as those who they primarily serve, their potential for disruption in case they misbehave as well as in different attributes related to the way they communicate or the breadth of the area of responsibility.

Moreover, we shed light upon the different DevBot ecosystems that are active within the company and surround important assets of the company. These assets include but are not limited to the source code of the autonomous driving product, the hardware resources such as Hardware-In-the-Loop (HIL) rigs, logs and simulations. In regards to the developers' perspective, it was observed that the motivations behind contributing to DevBot ecosystems match known DevBot benefits found in existing literature, as do some of the encountered challenges, especially related to the interactions between DevBots and humans. This study increases the knowledge on the challenges through its ecosystem perspective. Therefore, challenges related

to DevBot development have been brought up in regards to interactions between DevBots as well as between DevBots and assets. Practitioners also pointed out the need to abide by common development best practices and having ways to manage dependencies, when designing and building DevBot ecosystems.

In Chapter 2, an overview of the literature is presented, particularly containing background and context information primarily on bots, DevBots and software ecosystems. Next, Chapter 3 includes the suggested taxonomy for DevBots from an ecosystem perspective, where the four roles mentioned earlier are analyzed. Chapter 4 specifies the methodology followed in this study and Chapter 5 which present the results of the case study. Moreover, in Chapter 6 findings from the previous chapter are combined with each other as well as cross-referenced with existing literature. Additionally, the chapter includes the threats to validity and ways to mitigate them. Finally, the conclusion i.e. Chapter 7, summarizes the finding and brings forward some ideas related to future studies on the subject.

2

Literature review

To the best of our knowledge, there has not been researched explicitly focused on software ecosystems comprised of software development bots. That being said, there is a lot of bibliographical background around the individual elements which when combined they compose the larger topic in this thesis proposal. Next, we outline related literature and draw parallels between them to justify how they converge on the topic of the thesis.

2.1 Software agents and systems of agents

Research on software agents makes up the foundation for the work on our topic and has been active for many decades. Nwana (1996) traces research related to software agents back in the 1970s when they were referred to with the term "actors" and were characterized as software that is self-contained, interacts and is being run on the background [7]. Furthermore, the author attempts to classify them from three different perspectives, i.e. their "mobility", their proactiveness as well as three primary criteria: their level of autonomy, their ability to learn and the extent they cooperate with other agents.

Green et. al. (1997) attempt to give a more direct interpretation of software agents. They define agents as "computational" entities that generally aid users with computer-related tasks [8]. More specifically, they autonomously act on behalf of others and exhibit a degree of proactivity or react to changes in the environment. Interestingly, they also include the learning as well as cooperation with other agents as part of what determines them.

Jennings and Woolridge (1996) recognize that while there was not a complete consensus in regards to the definition of the term, they have specified three types of software agents based on the way they perform their tasks. First, they have identified the "gopher" agents that are rather simple and perform some predetermined tasks by themselves. Second, the "service performing" ones, act after a user interacts with them and finally, the "predictive" agents exhibit a more proactive behavior without requiring the user necessarily having to interact with them so to be triggered.

As we previously noted, a key characteristic of software agents was the extent they

cooperate with other software agents. This led to the concept of systems of agents. Sycara and Zeng (1996) explain that software agents need to form cooperative systems because as systems expand and get more complicated, containing this vast functionality into a single entity would be problematic [9]. This is due to the high amount of computational resources required for a single node to accommodate all the functionality. Additionally, maintaining one agent would turn it into a single point of failure for the entire application. Furthermore, following a distributed architecture allows for compartmentalization and gives the agents the possibility for specializing on a task that offers greater flexibility. They point out two distinct characteristics. Agents do not only serve humans but also other agents and agents now need to bear the technical means to communicate with other agents. Moreover, the researchers have distinguished three types of agents depending on what other actors they interact with as well as the information they have access to. Particularly, the "interface agents" interact directly with the human user, "task agents" may also exchange data with other agents and finally "information agents" offer access to a large pool of diverse data.

Flores-Mendez (1999) mentions that the inability of an agent to collaborate, either with other agents or humans, will eventually eliminate its usefulness. Furthermore, he points out the complementary nature of the agents within multi-agent systems, which allows for agents in such systems to collectively solve larger and more significant problems, with the Internet playing an important role in their emergence [10]. The author points out the two characteristics that collaborative agents need to possess which are the ability to discover each other as well as interact. To facilitate this, a protocol for agent interaction is suggested along with a system architecture. While the work described does not focus on agents that support software development we believe that several characteristics prevalent in systems of agents are relevant to our investigation. For example, in a system of agents the goal is to have different agents specialized in small tasks and collaborate, but also humans, to aid towards a specific goal. This is similar to the DevBots collaborating with developers but also exchanging information with each other to facilitate software engineering processes.

2.2 Software ecosystems

Having software and humans interacting and complementing each other to achieve a greater goal, brings resemblance to software ecosystems. Knauss and Hammouda (2014) mention that Software Ecosystems are defined as a collection of coordinated businesses that interact with each other and their shared market while sharing a common technical platform [11]. Interactions between businesses may be related to information, resources or artifacts.

van den Berk et. al. (2010) outline the different roles that the software ecosystem actors assume [12]. Particularly, they distinguish between actors that mainly produce information and value while others primarily consume those. At the same time, they have identified several smaller "niche player" roles that specialize in value creation for narrow use cases. Here some parallels can be drawn to systems of agents

which are specialized for different tasks individually, but collectively strive for a common goal. The possible correlation between the two concepts becomes more apparent when taking into consideration Campbell and Ahmed's (2010) remark, about software ecosystems depending on a common architecture [13].

Furthermore, while software ecosystems are recognized to boost innovation and decrease costs, Joshua et. al. (2013) mention that evolution management and the tools for the infrastructure which foster collaboration between the actors, are among the most typical challenges encountered in the relevant bibliography [14]. We have encountered similar obstacles in the literature surrounding systems of agents, where the need for standardizing coordination and architecture has been brought forward [9], [10], [15].

There are several roles for the actors prominent in SECOs which. Jansen et. al. (2009) specify three major players in SECOs: (i) Keystones, (ii) niches, (iii) dominators [16]. Keystone players are concerned with building and providing the common technology platform which forms the basis of a software ecosystem. Niche players on the other hand utilize the shared platform to provide value to one or more specialized market segments they are dedicated to. Next, dominators are players who try to either remove or integrate other keystone, niche or dominator players. Their goal is to serve the purpose of the assimilated players and be a prominent source of value creation within the ecosystem. While they can be initially successful, they jeopardize the long-term sustainability of the ecosystem. Furthermore, an additional role was specified; the bridge players. Entities active in more than one ecosystem can be considered as bridge players, however, it should be noted that this characterization is complementary to the rest, as it should be primarily seen as an additional attribute, not an exclusive trait.

2.3 The distinction between Bots and DevBots

Narrowing down the wide spectrum of software agents, we get bots which is software that integrates itself into different human task-solving processes. According to Lebeuf et. al. (2017) contemporary bots maintain human traits, typically demonstrated by holding conversations in natural language. [17] The authors note that bots can be classified based on the way they interact, proactively versus reactively as well as the range of their "intelligence". They may utilize the same communication channels as humans and often appear like other users of the system, being registered with another username, having a profile etc. [18]

Another way to categorize bots is the domain they are deployed. While they can be malicious by playing a detrimental role to network security [19] or spreading fake news on social media [20], they can be beneficial for fields such as software development. We refer to bots that facilitate software development as "DevBots". Wessel and Steinmacher (2020) claim that bots in software development aid by being the interfaces between humans and the various tools being utilized. [18] Furthermore, Erlehnhov et. al. (2020) [6] propose a classification for DevBots which has simplic-

ity as a driver in order to attract practitioners. Specifically, they outline different personas of DevBots users that are interested in particular software development activities or objectives. For instance, the "Charlie" DevBot users incorporate natural language into their vocal or textual communication, the "Sam" bots that generate code as well as the "Alex" bots that publish written messages in communication channels used by the development teams. Moreover, they describe various benefits of utilizing DevBots, which include improvements in productivity, information gathering, improving quality and task automation. Similar to the bot-using personas suggested by the authors, the actors as well as the shared assets between DevBots and humans, may add unique value or dependencies to software development. This is reminiscent of software ecosystems in the sense that practitioners can exploit those unique benefits of interactions and shared assets to enhance software development activities, such as Continuous Integration pipelines.

The current literature examines DevBots individually while this work intends to position them within a broader context. For example, the taxonomy proposed in [1] describes DevBots based on their purpose, the way they are triggered, how they communicate primarily with humans and the extent of their intelligence. Similarly, in [6] the authors mention several DevBot benefits and challenges mainly from a human perspective. What happens when DevBots interact with their surrounding environment, such as the various resources or other DevBots is only briefly touched upon. Specifically, challenges were detected when DevBots were interfering with each other [6]. Furthermore, the “Integrated” facet of the proposed taxonomy in [1] implies interaction with a particular infrastructure that could potentially be seen as other DevBots. This study extends the current literature on the characterization of DevBots as well as the developer impressions on them, by focusing not only on the human perspective but also taking into consideration their interactions with other DevBots and system resources.

3

Ecosystems of DevBots

The novel nature of the study requires the introduction of several roles which form a new taxonomy that describes DevBots from an ecosystem perspective. The different terminology is inspired by existing terminology in Biology and literature on Software Ecosystems (SECOs). With that in mind, the different *species* (i.e. roles) of DevBots we have theorized to coexist and function as an ecosystem are (i) Keystone bots, (ii) Niche bots, (iii) Dominator bots, (iv) Observer bots. The proposed species are primarily inspired by the different roles already defined in the SECO bibliography. This taxonomy differentiates from the existing one presented in [1] by taking into account the communication between DevBots as well as their interactions with other resources and humans. In other words, the taxonomy we introduce places DevBots within a project context, offering a holistic view on DevBot adoption and utilization in software development at scale.

Specifically, the first three can also be found in a SECO context [16]. In a SECO, the keystone players provide the technology which constitutes the foundation of the ecosystem, while the niche players utilize the said technology to produce value for particular business segments and, finally, the dominators try to assimilate other players in their attempt to control the ecosystem. The last role in our taxonomy, i.e., the observer, could not be directly identified in the SECO context, hence the observer role stems from Biology representing a scientist that observes an ecosystem in order to assess its status and identify threats or stability of the ecosystem's specimens and resources.

Expanding the metaphor of the natural ecosystem, we need to determine the surrounding context that different DevBot ecosystems operate in. In Biology, this context would be referred to as the "environment". The environment in the case of Ecosystems of DevBots is comprised of a collection of *assets*, that are involved in software development. When talking about software development projects, these assets are often immaterial, for example, source code or executable files, logs, machine learning models. However, assets may include hardware used for development purposes, such as servers, equipment, or Hardware-In-the-Loop rigs which due to their physical nature pose limitations in regards to availability and capacity.

In order to illustrate the different aspects of DevBot Ecosystems, we provide a series of tables and figures. Table 3.1 provides an overview of the different species that

we propose as well as high level descriptions of their primary trait. The primary trait defines what sets DevBots apart from each other without necessarily meaning that it is a trait exclusive to the particular species. Furthermore, for each role, we provide a characteristic representative of the species if the particular bots were to be utilized in an ecosystem context. Particularly, we illustrate these roles by referring to off-the-shelf bots and common in open source software development.

Table 3.1: Overview of resources in DevBot Ecosystems

Species	**Prime trait**	**GitHub Sample**
Keystone	Generate data consumed by other bots	doxygen-action
Niche	Generate data consumed by humans	codecov
Dominator	Generate composite data to achieve a multitude of goals	docker-build-push
Observer	Transform data for human consumption	dependabot

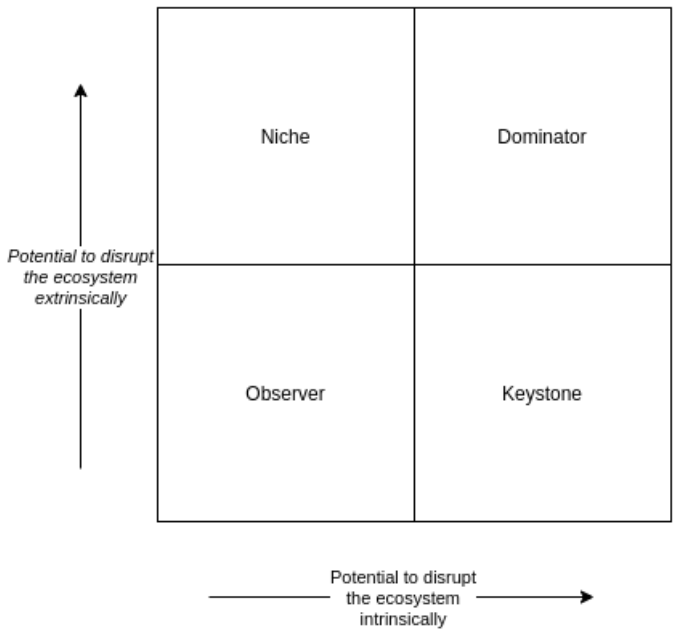
Table 3.2 presents the different resources the different species "feed" on. *Resources* in this case refers to artifacts or information from the environment and the classification as "processed" or "raw" respectively applies to whether the resource has been generated by other bots or not. Resources can also include physical assets, such as those described earlier. Overall, if a resource is *raw* it is also an *asset* of the environment. In turn, a *processed* resource cannot be considered an asset unless it *persists* and outlives the execution of the bot. Next, we detail each species and illustrate them with some specimens.

Table 3.2: Overview of species in DevBot Ecosystems

Species	**Resources**
Keystone	Predominately raw
Niche	Raw & processed
Dominator	Predominately raw
Observer	Raw & processed

Taking on the Ecosystem of DevBots from a different viewpoint, Figure 3.1 outlines the disruption potential of the different species. In other words, we refer to disruption in relation to the severity of adverse effects in the scenario where the DevBots misbehave or stop functioning. The vertical axis refers to the disruption in daily software development activities by humans and the horizontal to that of the ecosystem, i.e. other bots. In short, the figure conveys the potential for *intrinsic* and *extrinsic* disruption when the DevBots of a corresponding species malfunction. Intrinsic disruption refers to other DevBots of the ecosystem, whereas extrinsic disruption focuses on the ones that the ecosystem itself is designed to serve, i.e. humans.

Figure 3.1: Potential for disruption



3.1 Keystone bots

Similar to keystone players in SECOs, keystone bots comprise the foundation of DevBot Ecosystems. They primarily feed on raw resources and while the data they generate may be of use to software developers, the main beneficiaries of their activity are other bots of the ecosystem. Their primary purpose is to produce the *processed* resources which will be consumed by other bots. For example, they may typically be the prerequisites for the execution of other bots, ensuring that the environment is in an appropriate state that other bots can reasonably operate. Additionally, they can produce artifacts that are consumed by other bots, such as binaries or configurations.

A DevBot that could satisfy the criteria for being a keystone bot if used in an ecosystem context would be the doxygen-action bot found in the popular [21] GitHub Actions marketplace on GitHub. The particular bot parses source code and, if both the comments and configuration found in it are correct, then HTML pages are produced. It is typically the task of a different DevBot, to fetch the artifacts produced by doxygen-action and either publish this generated content directly or integrate it in a larger release. In other words, the doxygen-action DevBot will not handle the deployment, only create the artifacts to be deployed. The specific keystone may be useful for developers to know whether a change they have introduced contains syntactically correct documentation comments, however, the primary benefactor is a *different* DevBot which is, potentially, responsible for publishing a new release of the product's documentation, that may very well include more resources.

Keystone bots have the potential to majorly disrupt the ecosystem since they can directly affect the operation of other bots. In other words, those dependent DevBots

rely both on their own seamless operation as well as the fitness of the generated artifacts or information. Therefore, keystone bots should not only be characterized by high availability, but also by their reliability and correctness. If keystones do not conform to the expectations that the ecosystem has on them, there is a high risk that other DevBots will starve. That being said, the probability to *directly* influence software development activities is rather low as humans do not constitute the main stakeholder in the workflow of this type of DevBot.

3.2 Niche bots

Niche bots are reminiscent of niche players in SECOs in the sense that they focus on providing value for certain niches or to put it into context, very specific and atomic, use cases. These niches map directly to software development activities. Niche bots utilize both raw and processed resources, in other words, they *may* depend on other bots and their main stakeholders would be software developers. Their repertoire includes the generation of data and information that supports development activities such as testing, design or deployment. They are typically expected to compose the bulk of the DevBot population in an ecosystem since they provide the most immediate value to their human users who are the primary stakeholders.

Multiple off-the-shelf DevBots can be viewed as members of the Niche species since they are the ones considered as most appealing for the human stakeholders. For instance, Dependabot analyzes the dependencies of a project, determines potentially insecure ones and depending on whether a viable alternative exists, either warns the user about the risk or proposes a solution. Occasionally, the particular bot may *also* be given the authority to autonomously change the existing code base and eliminate the risk if it has proven itself trustworthy. Subsequently, Dependabot facilitates both the implementation and maintenance phases of software development.

Niche bots have the potential to disrupt software development activities since being unavailable or malfunctioning directly impacts the human-driven activity they are designed to facilitate. At the same time, their impact on other members of the ecosystem can be seen as minimal, as their output is typically not consumed by other DevBots or even if it is, other bots are supposed to be very loosely coupled to Niche bots (otherwise, they would become keystone bots).

3.3 Dominator bots

Dominator bots take inspiration from the dominators encountered in SECOs. In a SECO context, dominators seek to *assimilate* other players or *remove* them from the ecosystem while aiming to cover the entire spectrum of the business needs which was previously accommodated by the different players. Symmetrically, dominator bots attempt to satisfy multiple use cases by composing functionality that could under different occasions be distributed into separate bots. The data or information they produce is rather heterogeneous and due to their centralized nature, they exhibit

a monolithic nature in their design. Dominators tend to engage in multifaceted operations due to *convenience* and similarly to their SECO progenitors, they may cause the collapse of the ecosystem if they continue to assimilate functionalities and expand their responsibilities. For example, it may be technically easier to develop a single bot, i.e. a dominator, that performs multiple activities instead of having individual bots that must communicate with each other and therefore bear the technical overhead necessary to achieve this synergy.

It is relatively difficult to find dominator specimens in the "wild", as one could argue that a wide variety of features for a single bot is sometimes undesirable. That being said, bots such as the Docker build-push-action incorporate a set of functionalities that could be split into smaller and more flexible bots. The particular DevBot builds the Docker image maintained in a repository and if that succeeds it publishes it online. The problem with this approach is that one cannot reap, in isolation, the benefits of the two different activities, i.e. building and pushing, as they are coupled together. To give an example, one cannot *build* a Docker image as an attempt to merely verify whether the introduced changes are syntactically correct without publishing it. The particular DevBot seems to have been designed in this way because it was more straightforward. However, if it was utilized as a part of an ecosystem and the set of its responsibilities were to increase then the ecosystem would start becoming unbalanced by being increasingly reliant on them and individual bots to be incrementally devoured by the dominator.

Dominators have the power to disrupt an ecosystem both intrinsically and extrinsically. In regards to the former, as soon as dominators start to incorporate responsibilities that would canonically be attributed to a plethora of keystone or niche bots, an increasing number of bots has to depend on them or even be integrated into its codebase. At the same time, software development activities broaden their reliance on the dominators, since they accommodate a broad segment of the users' needs. As a result, if problems, such as a decrease in availability or correctness of a dominator are observed, then this has a major impact on both the ecosystem but also the software development that depends on them.

3.4 Observer bots

Contrary to the previously illustrated bots, the observers do not stem from a role present in SECOs but Biology. In nature, disciples of different scientific fields *observe* ecosystems in order to understand them, and characterize the impact of the different species in the environment (and vice-versa). In our context, we define the observer as the role that aims to extract information from the DevBot ecosystem in a non-invasive manner, without obstructing, influencing or somehow meddling with the internal affairs of ecosystem members. The observer strives to merely formulate an accurate *view* of the system and for their presence there not to affect the behavior of the species.

In other words, an observer DevBot utilizes raw and processed resources to gather

different *views* of the system but *not to generate any new resources* that can be considered as part of the ecosystem or its surrounding environment. For example, an observer may notify about changes, create reports on the current *status quo* and visualize metrics in a manner that is easily digestible by software developers. It can be utilized to document and increase the understanding of the different metrics that depict the *health* of the ecosystem or the environment.

A DevBot that could be considered as an observer when utilized in an ecosystem context is Codecov, which is a bot that *displays* testing coverage for multiple platforms. We classify Codecov as a prime sample of the Observer species because it uses both raw and processed resources to create human-readable reports. To be precise, it utilizes source code files (raw resource) and combines them with test coverage metadata that come as machine-readable files (processed resources) to generate illustrative reports which indicate the current and past test coverage of the project. In contrast, Codecov does not generate any *new* information that feedbacks into the ecosystem, i.e., all the necessary data are already present in the environment or have been produced by other bots. For example, DevBots that would require sufficiently high code coverage to operate, have no interest in Codecov's reports as the information they need is already available. In short, an Observer bot merely *transforms* already existing material, for human consumption.

Specimens of this species have a low capacity to disrupt the ecosystem because they do not generate any resources and, consequently, other DevBots do not rely on them. As previously stated, the information that they transform already exists and can be consumed by other species. Furthermore, the extrinsic disruption they can cause is also limited. While it may become less *convenient* for humans to visualize metrics on the health of the ecosystem and the environment, the relevant data is still present and can be parsed by different means if necessary. Strictly speaking, Observer bots can be considered as a commodity and thus the overall risk for them to destabilize the ecosystem or software development activities is relatively low.

3.5 Scaling DevBot ecosystems

Throughout a software development project, we may encounter multiple DevBot ecosystems surrounding different assets. For example, we may have a DevBot ecosystem around machine learning models, another one around the source code of the product, and a third one that facilitates activities around the utilization of hardware tools. When DevBots are active in multiple ecosystems they can be considered as Bridge bots or, to be more precise, "bridge" is a characteristic rather than a distinct novel species. The name is inspired by bridge players in SECOs which are responsible for the mobility of various resources and information among the SECOs that they are operating in [16]. If an interaction between DevBot ecosystems is deemed necessary, Bridge bots facilitate it, ultimately enabling the formulation of large ecosystems encompassing smaller ones, akin to what is taking place in nature.

At this stage, we do not focus on a refined definition of a Bridge bot, since our goal

is to validate the roles composing a single Ecosystem. Consequently, we encourage researchers to expand on the facet of Bridge bots in future iterations of our taxonomy.

4

Methodology

4.1 Case study

To shed light on DevBot Ecosystems and evaluate our taxonomy, we conduct a case study. Specifically, we study the existing utilization of DevBots in an automotive project. The company behind the project is an automotive supplier in Gothenburg, Sweden, that specializes in the development of the entire software stack that enables autonomous driving for next-generation vehicles. The stack includes everything related from low-level camera-related drivers to software running in the cloud. The various components process the video stream using computer vision as well as machine learning to determine the path the vehicle should follow. As a consequence of the large vertical, despite contributing towards a common goal, the different components within the project are often heterogeneous in regards to their scientific domain, programming language as well as safety requirements.

In the particular project, multiple DevBots are being utilized to conduct various activities in different stages of the development and verification process. While there is a diversity of DevBots interacting with software developers, most DevBots communicating with other DevBots reside in the Continuous Integration pipeline and typically output information into the communication channel for code reviews.

In our study, we take into consideration (i) the DevBot activities, (ii) the interactions and dependency relationships between them as well as (iii) the resources consumed by the different bots, such as source code, logs, binaries, etc. Furthermore, we interview individuals who have been involved in DevBot development to acquire qualitative data on DevBot design considerations and how humans perceive various aspects in regards to their operation and design. Table 4.1 outlines the case study based on the guidelines proposed by Runeson and Höst (2008) [22].

4.2 Research Questions and Data Collection

We devised *three* research questions. With RQ1 our goal is to evaluate the ecosystem DevBots taxonomy we previously proposed. To achieve this, we describe prime specimens of each DevBot species found in the DevBot ecosystems of the company we study. With RQ2, we delve into the different DevBot ecosystems that we discov-

Table 4.1: Case study summary

Objective	**Explore**
Context:	DevBot ecosystems in Continuous Integration
Case:	Automotive supplier for autonomous driving software
Theory:	Software Ecosystems, DevBots
Methods:	Analysis of archival data, metrics and an interview study
Analysis:	Utility demonstration of the taxonomy and thematic analysis of the interview study

ered during our case study, the DevBots that comprise them and the assets the bots consume. Next, RQ3 offers insight from the DevBot developers' perspective. Particularly, we look into the different design and architectural considerations related to DevBot development.

- **RQ1: What is the role of the different DevBot species in the ecosystem?**
  1. How can we characterise the species in DevBot ecosystems?
  2. What are the assets the different species produce and consume?

With this research question, our goal is to describe the role of the different species in DevBot ecosystems. Specifically, 30 of the DevBots used in the company were analyzed and 20 of them were selected for illustration purposes. We selected those 20 DevBots based on the DevBots characteristics proposed in [6]. We use the sampled DevBots to perform a utility demonstration to evaluate the faceted taxonomy [23] we proposed in an earlier chapter. From the selected bots we identify *specimens*, a representative for each species, and illustrate how they operate, their importance in the ecosystem, and their relation with ecosystem assets. Furthermore, aside from the proposed taxonomy which specifies several roles, we highlight the unique aspects for each of those roles to the chosen DevBots. To determine the above we look into the source code to determine the purpose of the DevBots and the artifacts they produce or consume. Specifically, these artifacts include messages posted in code reviews or chat rooms along with the discussions that ensued.

- **RQ2: How can we describe the DevBot ecosystems?**
  1. How can we describe the different DevBot ecosystems?
  2. What are the assets of the different ecosystems?
  3. What are the roles of the different DevBots in the ecosystems?

After RQ1, where we contemplated upon how the different DevBot species manifested in four of the case study's core ecosystems, RQ2 dives into three different ecosystems and takes a detailed look at the types of DevBots that comprise them. This research question gives a comprehensive view of the different ecosystems that flourish in the particular automotive supplier, their features and the DevBots that are active within them. Compared to the first research question which offered a *macroscopic* view on the status quo, here we inspect the different DevBot ecosystem

members. Metaphorically speaking, if we draw parallels to biology we can imagine the automotive supplier we study resembling a savanna. In RQ2, we lay out the different ecosystems, i.e. the water paddle, the grassland and the forest as well as articulate on the daily routines of the *dwellers*, found in the said ecosystems. To attain this perspective focused on the ecosystems, we perform a qualitative analysis and study artifacts similar to those in the first research question. These include the DevBot source code, dependencies between DevBot ecosystem members as well as the different assets they are involved with, either as producers or consumers.

- **RQ3: How can the development of DevBot ecosystems be described?**
  1. Why contribute to the development of DevBot ecosystems?
  2. What are the challenges related to interactions between DevBots and assets?
  3. What are the challenges related to interactions between DevBots?
  4. What are the challenges related to interactions between DevBots and humans?
  5. What are the system and software design considerations when developing DevBot ecosystems?
  6. What are the trade-offs related to Dominator DevBots?

In RQ3, we expand on the different challenges and influencing factors that should be reflected upon when developing DevBots meant to flourish in an ecosystem. Specifically, we interview practitioners involved in the development and maintenance of different DevBots to determine the various trade-offs related to the interactions of individual DevBots with their environment. We perform a thematic analysis [24] over the data collected via semi-structured interviews to determine the different architectural and design concerns when developing ecosystem DevBots. Furthermore, we shed more light on the dominators to determine the reasons behind their emergence as well as how they are perceived by practitioners, despite being theoretically considered antipatterns.

Table 4.2 outlines the collected artifacts along with the research questions that they help to address. In summary, we answer RQ1 via utility demonstration of our taxonomy and inspection of existing DevBots manually selected. RQ2 revolves around the characterization of various ecosystems based on the assets that they revolve around as well as different DevBots inside the said ecosystems. Lastly, we conduct an interview study and thematic analysis in RQ3 to understand how practitioners perceive DevBot development and its different aspects. Next, we detail the components of our interview study. Figure 4.1 presents the elements of the overall approach in regards to the planning, the data collection, and the data analysis.

4.3 Interview Study

We selected participants for our interview study via convenience sampling [25]. Specifically, 20 of the most prominent DevBot contributors were selected by analyzing the version control history of the most actively developed DevBots. Next, we contacted them via direct messages on the company's discussion platform and presented the topic of the research as well as inquired about their interest to partic-

Table 4.2: Collected artifacts

Collected artefact	Artefact description	Related RQ
DevBot source code	Source code of the DevBots	RQ1, RQ2
DevBot console logs	Console output during DevBot execution	RQ1, RQ2
Ecosystem assets	Execution results and binaries produced by other bots, product source code, hardware resources	RQ1, RQ2
DevBot text reports	Reports or messages sent by DevBots and published in team communication channels	RQ1, RQ2
Developer discussions threads	Comments made by developers initiated from DevBot reports or messages	RQ1, RQ2
DevBot developer interviews	Interviews with practitioners involved in DevBots development and maintenance	RQ3

ipate in an interview. We received 16 responses and 11 participants agreed to take part in the interviews. The ones who declined claimed they were not familiar with the topic. All 11 interviews took place within one week.

The interviews were semi-structured, each lasting circa 30 minutes and were conducted via video calls, using the company's sanctioned teleconference software. All individuals belonged to different teams and work as software developers. The interview guide, found below, was designed based on the goals of RQ3 and iterated by the academic supervisor.

1. What is your role in the company?
2. How did you contribute in DevBot development?
3. Why did you change the bot? What was your motivation behind making the said contributions?
4. What are the challenges when developing DevBots with respect to:
   - The assets used by the bots
   - The interactions between bots
   - The interactions between bots and humans
5. What are the architectural/design considerations when developing DevBots?
6. If you had a choice, would you prefer several functionalities offered by one bot, or those same functionalities spread across multiple bots?
7. Have you been involved in developing a DevBot with a broad functionality?
   - Where did the need to create a bot with broad functionality come from?
   - How would you assess building multiple bots with narrower responsibility that depend on each other instead?

Table 4.3 presents how the different interview questions link to specific parts of RQ3 and its underlying questions. While conducting the interview, timestamped notes were taken which were utilized along with the interview recordings to produce codes

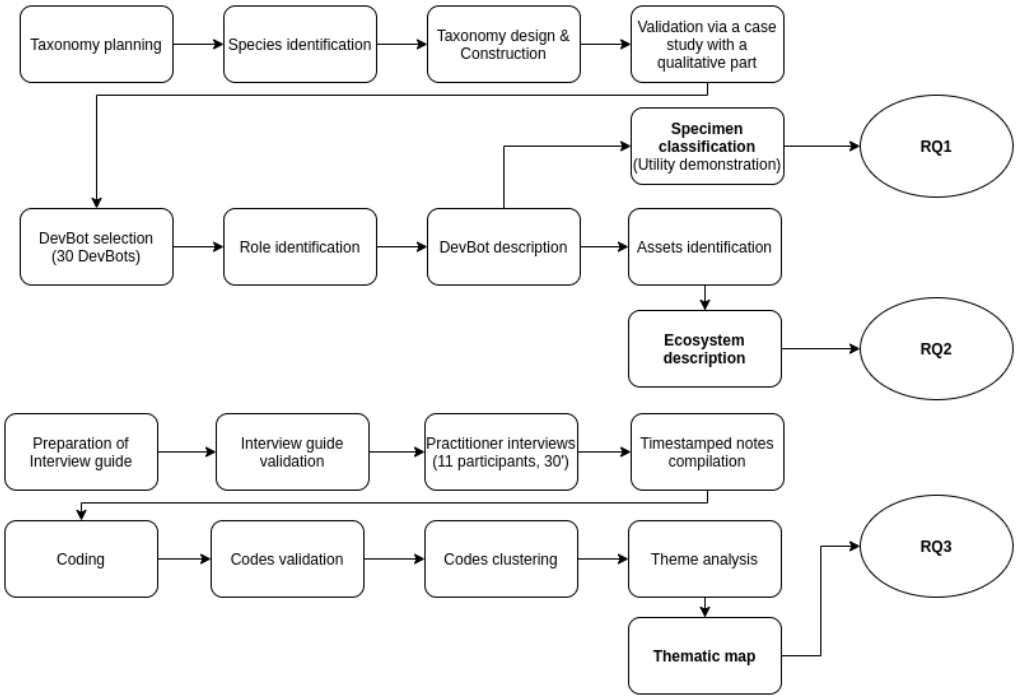


Figure 4.1: Methodology overview

that highlighted the responses of the interviewees to the different questions. Next, the codes were clustered and a theme analysis was performed to produce a thematic map. The map was used to address the final research question.

Table 4.3: Interview questions motivation

Interview question	Research question	Notes
1	N/A	Introduction
2	3.1	Motives behind development
3	3.1	Motives behind development
4	3.2, 3.3, 3.4	Development challenges
5	3.5	Design considerations
6	3.6	Dominator development
7	3.6	Dominator development

5

Results

5.1 RQ1 - Classifying DevBots from an ecosystem perspective

Due to the lack of previous research on the subject, in an earlier chapter, we introduced a taxonomy to describe DevBots in Ecosystems. To allow flexibility in our taxonomy and accommodate aspects or perspectives that we missed, we performed a faceted analysis [23]. A *facet-based* taxonomy allows for multiple characterizations for each entity that we classify. We validate our taxonomy via a *utility demonstration* by analyzing the different classifying different bots in the Continuous Integration pipeline of our case company.

Figure 5.1 illustrates some of the most active DevBots in the company. The assets and bots are described throughout this section. The dependencies imply that the dependant bot *requires* a piece of information such as a successful execution or artifacts such as generated source code, a binary or a configuration file from the parties it depends on. Table 5.1 includes brief descriptions for each of the selected DevBots.

The DevBots that were studied to validate the taxonomy consume are focused around certain *assets*. The main asset, attracting the most of the DevBots is the source code of the company's next-generation product. The source code consists of a single repository with all components necessary for the product residing inside. The DevBots are active in the Continuous Integration pipeline and are tasked with building, testing the software and ensuring its adherence to various quality and safety standards.

Despite the source code being potentially the most important asset, it is not the only one that DevBots engage with. After the software is built, it is being tested in Hardware In the Loop (HIL) rigs which allow the deployment of software in an environment that is similar to the production one. Next, various tests verify the product's functionality. That being said, the HIL rigs are not only utilized in the CI pipeline but also independently by developers that want to try the software out or investigate issues. Since there is a finite amount of rigs and a limited amount of activities that can take place on them at a given time, they need to be *booked* before

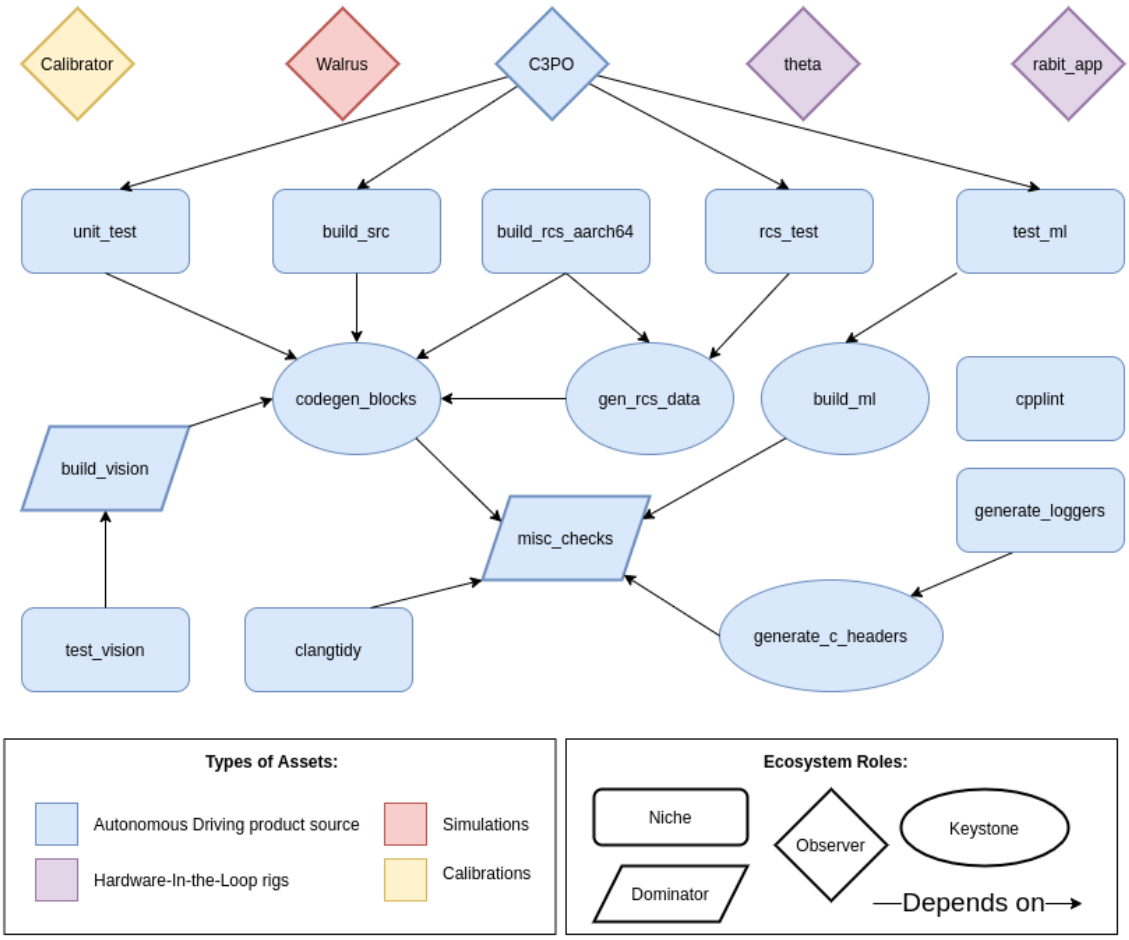


Figure 5.1: An overview of the most active bots. The different bots are color-coded depending on the asset they are related to, whereas their shape indicates the species. Furthermore, the dependencies between them are indicated via arrows.

being used.

Furthermore, to verify the quality of the product and quantify the performance according to several Key Performance Indicators, the simulations are continuously evaluating the product. Another asset that is being consumed by DevBots is *calibration* which includes results related to camera placement benchmarks on autonomous vehicles. Both of these assets do not consist of developed software or binaries, but rather results related to the runtime behavior of the product.

To determine whether a software agent is a DevBot, a simple model has been proposed [6]. For a DevBot to exist, it is essential for it to communicate via the software development team's channels. In our case, the DevBots are active in two communication channels: i) the version control management software where code reviews are taking place, ii) the chat and voice-based communication platform where developers and the teams hold their daily discussions and collaborate. In the former channel, DevBots communicate by leaving comments and declare whether the code is in a desirable state for being merged or not. In the latter, DevBots publish messages

Table 5.1: Selected DevBot descriptions

DevBot	Description
codegen_blocks	Generates code from model driven development tool
gen_rcs_data	Transforms high level scenarios to serialised configurations
build_ml	Builds machine learning related code
generate_c_headers	Generates code from configuration
unit_test	Builds and runs unit tests for the product's source code
build_src	Compiles the product's source code
build_rcs_aarch64	Builds scenario related components
rcs_test	Runs scenario related component tests
test_ml	Runs machine learning related component tests
cpplint	Runs lint tool
generate_loggers	Builds executables for logging
clangtidy	Runs static analysis tool
test_vision	Run vision related tests
Calibrator	Reports calibration CI job results
Walrus	Reports simulation CI job results
C3PO	Notifies about code review related events
theta	Notifies about rig reservations
rabit_app	Reports CI job results running on rigs
build_vision	Builds vision components and unit tests
misc_checks	Verifies configuration, code style and runs sanity checks

either in public channels used by teams or send private messages directly to the concerned developers.

5.1.1 Keystone bots

A number of keystone bots were identified. Their typical directives include assessing whether the code is in an appropriate state for other, more time-consuming, DevBots to operate. Alternatively, they may generate code from models or configuration files. As for keystones, the DevBots' primary stakeholder are other bots which rely on their output to operate. Keystones may depend on other individuals of their kind and have at least one other DevBot depending on them.

A prime specimen of the species is *codegen_blocks* which generates code based on model-driven software development tool models. Despite producing useful code, the bot is not considered critical to the project functionality-wise, yet, it is a cornerstone for building the product's software. This is because multiple components are dependent on the generated code. If no issues are encountered during its operation, codegen_blocks uploads the generated code for it to be utilized by the multiple bots that depend on it. Another representative of the species is *build_ml*. Similar to the previous bot, this one also generates artifacts utilized by other DevBots. Specifically, its output is necessary to deploy the company's machine learning evaluation environment. The number of bots that depend on it is very limited since the artifacts

it produces support a narrow use case.

5.1.2 Niche bots

Niche bots appear to be the most populous species and are the most visible to software developers. They are tasked with a wide variety of responsibilities which stretch from running static code analysis tools to building the entire software stack and running different kinds of tests. Niches output primarily aims to accommodate the needs of software developers directly. They often depend on a single keystone, less frequently on more than one (e.g., 7 out of our 9 niche bots depend only on another bot). Occasionally niche bots do not depend on any keystone at all, feeding exclusively on raw resources (cpplint).

A characteristic example of a niche bot is *build_src* which builds the entire software stack for the product. It utilizes the generated code produced from the keystone bot that it depends on (codegen_blocks) and provides the developer very significant feedback on whether the introduced change breaks the build of the entire system or not. Another keystone that is of equally high importance to developers is *unit_test*. The particular bot builds the unit tests for the entire software stack, executes them, calculates the coverage and finally posts a comment along with the code reviews that contains a verdict on whether the commit is suitable to be merged. If it is not, then a link is provided to a report that contains the details a software developer would be interested in to discover the reasons behind the DevBot's failure. Note that both the *unit_test* and the *build_src* have the same dependent relationships from and towards other bots, yet, they are designed as separate entities of the ecosystem, since they handle different resources (code and tests), and fulfill unique purposes.

5.1.3 Dominator bots

Recall that Dominator often fulfills a series of composite tasks, which often indicates that it adds a composite value to the ecosystem, by handling complex tasks. We argue that Dominator bots could be considered anti-patterns and therefore are expected to be scarce in a healthy DevBot ecosystem. To detect them, we delved into the responsibilities for each bot, seeking *monolithic* design and features. In Figure 5.1, we classified two dominator bots. One is still being utilized at the company, whereas the other has been deprecated. Interestingly, aside from the multitude of functionalities they are tasked with, there are also rather *irregular* dependencies on them. Specifically, a large number of DevBots are directly or indirectly dependent on them, which can be explained by the generic nature of the dominators to fulfill a more generalist role.

For instance, *misc_checks* is the first DevBot to be triggered after a new commit is introduced and includes a wide variety of tasks to check over the committed code. Particularly, the bot tries to determine whether multiple qualitative criteria are satisfied and if so, allow other more computationally expensive DevBots to execute. Therefore, we classify *misc_checks* as a dominator due to the heterogeneous nature of its duties. Instead of creating multiple bots with modular tasks, software

developers seemingly find it more convenient to increase the functionality of the existing bot. Consequently, the degree of *specialization* of this bot is progressively decreasing. The increasing amount of diverse features concentrated in the particular DevBot is the reason for it being a major source of risk in the system in case part of its large functionality malfunctions. Even though it is deprecated, *build_vision* is another indicative specimen of a dominator. The particular DevBot would build both the software stack *and* the test binaries. Then the test binaries would be uploaded and utilized by another DevBot that would merely run them. Building unrelated binaries in the same DevBot was done purely for convenience during the build and configuration phases. However, this design added an increased complexity in the software that instruments and runs the test. Eventually, this approach was abandoned in favor of the existing DevBot that builds and runs just the unit tests. In short, this dominator bot provided a broad functionality that, while initially acceptable, the drawbacks of its multiple responsibilities eventually outweighed the benefits.

5.1.4 Observer bots

Observer bots offer users impressions of the related assets or the rest of the ecosystem. Observers are easy to isolate and classify in our taxonomy. They are particularly visible to the bot users (developers, managers, etc.) since they communicate in relatively more illustrative manners, compared to the other species of DevBots in the ecosystem. Observer bots depend either on processed resources produced by other DevBots, or raw assets. In the DevBot ecosystems that we study we identified occurrences of both variants.

In Figure 5.1, *C3PO* is a DevBot that monitors activities taking place in code reviews. This includes comments posted by other DevBots. When there is a new development, it relays it as a private message to users that are participating in the code review. The particular bot can also be configured via commands in natural language. While *C3PO* depends on other DevBots, no DevBot depends on it, meaning that the intrinsic disruption that this particular bot can introduce to the ecosystem is minimal. Furthermore, it is seen as a commodity to the developers because they can merely visit the code review and check for new material by themselves. Thus it will not hinder software developers from their activities in case the bot becomes defective.

Calibrator is an observer that handles a different asset, specifically, the results from calibration operations. Particularly, teams that initiate calibration activities, which involve various physical components and hardware found on vehicles, utilize the Calibrator to sends messages to their team channels regarding the progress and posts the results of their effort. The particular teams do not rely on the DevBot, as the information reported can also be found in primary sources, however, it provides them with a highly appreciated convenience, considering the discussions that get triggered from the DevBot's messages.

5.1.5 Bridge bots

Bridge players in SECOs are not considered to be distinct species but a *characteristic* of players active in multiple ecosystems. In the DevBot ecosystems analyzed in this section, we did not discover any Bridge bots. Interestingly, DevBots that could be identified by their bridge properties proved rather elusive since we could not immediately spot DevBots active in multiple ecosystems. Instead, the bridge property of DevBots was revealed by looking closely into the source code of the bots. Particularly, we observed that components that were part of a Dominator bot in one ecosystem were, in fact, also utilized as distinct and "isolated" bots in another ecosystem.

When we investigated another ecosystem related to machine learning models, we identified a residing bot named "*copyright* bot", which is responsible for verifying whether the copyright notices in the different source code files are up to date and valid. Its source code reveals that the bot is made up of one of the components that constitute the *misc_checks* which, in our investigated ecosystem is a Dominator bot. Here, we distinguish these two ecosystems, as they are actually two distinct repositories. Moreover, another component of our Dominator bot is used as an individual bot in the machine learning ecosystem, namely, the *commit-message* bot. Specifically, this DevBot verifies whether a commit message abides by a set of best practices.

Upon an initial inspection, both the *copyright* and the *commit-message* bots are classified as niche bots in their native ecosystem (the machine learning ecosystem), but both are also integral elements of a Dominator bot in our analyzed ecosystem. Moreover, the bridge property of the dominator certifies its role as a *compositor* of diverse functionality which could have otherwise constituted multiple distinct DevBots. Those observations indicate that the role of the Bridge bot requires a more detailed inspection of the assets used or shared by the bots. The reason is that DevBots may be pervasive to different ecosystems through, e.g., reuse, yet they do not necessarily act as a communication bridge between those two ecosystems.

A deeper look reveals more DevBots that realize synergies between ecosystems. The *cd_config_generator* DevBot monitors the source code repository for the main autonomous driving product and when changes are introduced in particular directories, automatically generates a configuration pipeline for the Continuous Deployment tool, creates a code review containing the proposed changes and adds the ones who approved of the original change as reviewers. While the particular DevBot does not perform any visible activity in the source code repository, it is a significant contributor to the ecosystem that is responsible for the pipeline configuration for the Continuous Deployment environment.

5.1.6 DevBot ecosystem species characterization

In [1] the authors propose a facet-based taxonomy for DevBots which can naturally be applied to DevBots in ecosystems. Table 5.2 shows the characterization of the

different species introduced by the current study, viewed under the light of the existing DevBot taxonomy.

Table 5.2: DevBot Facet classification according to [1]. Since many of the classifications overlap between our taxonomy roles, we highlight in bold the unique classifications per species.

Species	**Facets**
Keystone	Specialist, system-based, integrated, keyword-based, output, static, no-input
Niche	Specialist, system-based, integrated, keyword-based, output, static, no-input
Dominator	**Generalist**, system-based, integrated, keyword-based, output, static, no-input
Observer	Specialist, system-based, integrated, **natural language**, output, static, no-input

After classifying the DevBots included in this study, we find several similarities between them, irrespective of the species. Particularly, our DevBots appear to be *system-based* which implies their execution is not initiated by software developers directly, but by the software system itself via the Continuous Integration pipeline. Furthermore, they are *integrated* meaning that they could not reasonably exist outside this particular environment, e.g., an open source project, without substantial modifications. Next, their direction is *output* as they merely react to user activity and in regards to their ability to adapt, they should be considered as *static*. Finally, they typically require no guidance from the user once they are triggered and thus characterized as *no-input*.

The majority of the bots have a narrow, typically atomic, set of goals, therefore should be viewed as *specialist* bots. Dominators are an exception to this rule since they strive to accommodate a plethora of use cases and are therefore labeled as *generalists*. Another point of divergence is the language the DevBots opt to communicate. All species except observers favor a keyword-based language when sending presenting information to the user. For example, the messages can be as simple as "PASSED" or "FAILED". On the other hand, observers have adopted a *natural language* approach when sending messages, which are more verbose and reader-friendly.

RQ1: What is the role of the different DevBot species in the ecosystem?

Based on the utility demonstrations of our taxonomy, we saw many of the roles below fulfilled by one or more DevBot in our case company.

- **Keystone DevBots** primary concern is to generate data for other DevBots
- **Niche DevBots** most significant stakeholder is humans
- **Dominator DevBots** synthesize functionality with broad responsibility
- **Bridge DevBots** show activity or connect multiple ecosystems

5.2 RQ2 - Description of DevBot Ecosystems

In Figure 5.1 some of the most active DevBots in the company are presented to give the reader a larger picture of the dependencies between the bots and a rough estimate of the population ratio between the species. While we previously evaluated our DevBot ecosystem species taxonomy and put some of the most characteristic specimens in the spotlight, in this section we give the reader a macroscopic view of the ecosystems, beginning with a description of various ecosystems themselves and then we delve into the individuals thriving in them.

Building upon the ecosystem *metaphor* and the analogies to biology, we can view that the software development organization we study as a *savanna*, comprised of different ecosystems. In nature, the ecosystems would be focused around different physical landmarks, such as a body of water, an area with low vegetation, or a forest with tall trees. For software development, we can consider the different *assets* as pillars of the different ecosystems. In the following sections, we describe some of the different ecosystems that exhibit DevBot activity. Note that the DevBots belonging to the category within each ecosystem can belong to different species, such that the classifications mentioned below in each ecosystem are orthogonal to the DevBots' species (e.g., keystone, dominator, etc.).

Next, three ecosystems will be presented: (i) source code, (ii) hardware and (iii) simulations. The source code ecosystem surrounds the product's source code and related assets such as code reviews and binaries. The hardware ecosystem includes DevBots which are focused around Hardware-In-the-Loop rigs and data collection vehicles, while configuration files and collected logs comprise the center of the simulations ecosystem. Figure 5.1 highlights a subset of the bots which populate these three ecosystems and Table 5.1 briefly describes them.

5.2.1 Source code

The most significant ecosystem, in terms of DevBot activity, is the one revolving around the source code of the company's autonomous driving product. The source code is concentrated in a single repository (i.e. *monorepo*) and includes the majority

of the autonomous driving software vertical that spans from low-level camera drivers to high-level planning of how the vehicle should move.

The ecosystem of DevBots around the main product's source code is being *actuated* by the Continuous Integration pipeline. DevBots in this ecosystem are related to code reviewing activities and in a nutshell, try to determine whether a code change is worthy to become part of the codebase. To word this differently, the DevBots in this ecosystem are related to the *software implementation* phase and the primary human action that initiates their activity is a *push* to the version control system.

Inhabitants of the particular ecosystem include DevBots which are active primarily before, but also after, a code change has been merged. They can be classified under the following categories based on the technical nature of the activity they are engaged in: (i) code formatters, (ii) static code analyzers, (iii) code generators, (iv) runtime analyzers, (v) code builders.

The formatters are tasked with ensuring that the source code looks consistent and follows the same standards in regards to indentation, aesthetics, naming conventions etc. There are multiple DevBots tasked with formatting the source code not only because different programming languages are used, but also due to the broad range of formatting rules defined by the company, that cannot be easily contained in a single DevBot.

Static code analyzers examine different branches in non-compiled source code aiming to determine potential issues connected to performance, undefined behavior, safety, adherence to modern best practices, language-specific idioms and readability. There are multiple DevBots tasked with static analysis because different tools are utilized even within the same programming language. The underlying tools can either be general-purpose open source software that examines source code for compliance with commonly adopted best practices or specialized proprietary products that discover domain-specific defects and violations of functional-safety standards.

Code generators utilize configuration files and models produced by model-driven development tools to produce source code that can be included in the product's compilation. Generated code is necessary for multiple other bots, especially those that are related to compiling the product's source code.

Runtime analyzers include DevBots which perform various checks on runtime artifacts that are generated by compiling parts of the source code. Specifically, DevBots of this category often execute unit and integration tests. Moreover, they may profile code and use sanitizers or other dynamic analysis tools to discover potential memory leaks and illegal memory access.

Code builders are tasked to compile the product's source code for all the supported variants meaning that different compilation toolchains are used. Different permutations of the product are being built and deliverables for customers are compiled as if they were to be released and deployed to a production environment. DevBots that operate as code builders assure, to an extent, that the software product is

continuously in a releasable state and that quality is regularly safeguarded.

5.2.2 Hardware

Another important asset that attracts bot activity around it is hardware that enables the testing and verification of software under development. Developers may try out their software in an in-house reference platform that substitutes the need to have access to a fully equipped and working production system, i.e., a vehicle. Specifically, the following resources are at their disposal: (i) Hardware In the Loop (HIL) rigs where a bare-bones setup reminiscent of a production system allows them to verify basic functionality, (ii) Reference vehicles that are similar to the customers' system which enable developers to test larger verticals of the company's product.

HIL rigs and the reference vehicle platforms are utilized to assess the *integration* of the different software and hardware components. As expected, the availability of the hardware assets is limited, therefore DevBots are assisting in booking the resources and scheduling their usage. Furthermore, we discovered DevBots also tasked with utilizing the HIL rigs to automatically verify software that has been already merged to the code base and report the results. In a nutshell, DevBot activity in this ecosystem is related to the *verification* of software and they are triggered either by a request to reserve a resource or code being *merged*. DevBot activity in this ecosystem is focused on two activities: (i) reservation of hardware assets, (ii) periodic verification of software on hardware.

Hardware, due to its material nature, comes in finite numbers and therefore exhibits limited availability. There are different stakeholders with a need to utilize the hardware assets, such as developers manually testing their software as well as DevBots periodically verifying the latest build on actual hardware. This creates the need for synchronizing access to the hardware resources, as for most use cases they can be reliably utilized by exclusively one party at the time. DevBots handle the booking management of the said resources and additionally notify human users via private chat messages about the status of their booking.

Furthermore, there are DevBots that periodically verify the latest software build on actual hardware to test larger verticals of the system. They do not run before introducing each software change but at regular intervals due to limited hardware availability which also needs to be shared with developers who utilize it for manual testing. These DevBots report their findings as chat messages in, often, dedicated team channels and trigger discussions among the developers about the potential corrective actions that need to be taken.

5.2.3 Simulations

The Autonomous Driving domain requires the *verification* of software in multiple levels. This partly includes assessing the correctness and performance of a significant portion of the software stack against a set of KPIs in open and closed-loop simulations. The simulations rely on configuration files that point to different types

Table 5.3: Ecosystems overview

Ecosystem	Assets	Activity
Source code	Source code files, code reviews, product binaries	Implementation, Testing
Hardware	HIL rigs, Data collection vehicles	Testing, Verification
Simulations	Configuration files, Logs	Testing, Verification

of logs that have been previously collected. These logs are consumed by the simulations which provide continuous feedback on the quality and performance of the company's software stack. These simulations, utilize the previously accumulated real-world data to test the software stack, and considering they are very costly to collect, they are deemed as one of the most high-valued assets of the company.

The DevBot activity in this ecosystem is concentrated around the execution of the open and closed-loop simulations and reporting the latest status via chat messages in team communication channels. The relatively few inhabitants of the simulation resources ecosystem are *periodically* triggered and their activities fall into the testing and verification domain. Specifically, they report the simulation results, along with the source code baseline involved in the tests and the configuration involved in each execution run.

5.2.4 Summary about Ecosystems

Multiple DevBot ecosystems can be found in the "savanna" that we delved into. By taking a closer look at some of the ecosystems we realize they are centered around relevant resources, such as the source code and code reviews, Hardware-In-the-Loop rigs and data collection vehicles, simulation configuration files and logs. We consider these assets as part of the same, larger, landscape since not only they are part of the same R&D effort, but also are either closely related or derivatives of each other. For example, code reviews and binaries are a "consequence" of source code, logs are a "result" of running the binaries on data collection vehicles etc. This realization conveys the broader correlation between different ecosystems and justifies the perspective of individual ecosystems when looking at the DevBots. Table 5.3 presents a high-level illustration of the savanna, the local landmarks as well as the related software development activities.

RQ2: How can we describe the DevBot ecosystems?
Our manual analysis indicates that a software project can host multiple ecosystems. Each ecosystem is centered around assets, such as the product's source code, the hardware resources and the simulations. Ecosystems have a distinct role in the project's **lifecycle** and are oriented towards specific software development activities.

5.3 RQ3 - Designing and Developing DevBot Ecosystems

To determine the different considerations and trade-offs involved in the development of DevBot Ecosystems, we interviewed 11 practitioners who have been involved in DevBot development. The practitioners are all software developers in different teams within the same company. They work towards the development of the same product while belonging in different functional areas. We assign different identification numbers to protect the participant's identity and, from the quotes presented below, have redacted or removed any references to specific technologies.

Figure 5.2 shows our resulting thematic map containing the main themes and subthemes that characterize the responses to the interview questions. We detail each theme in the following sections.

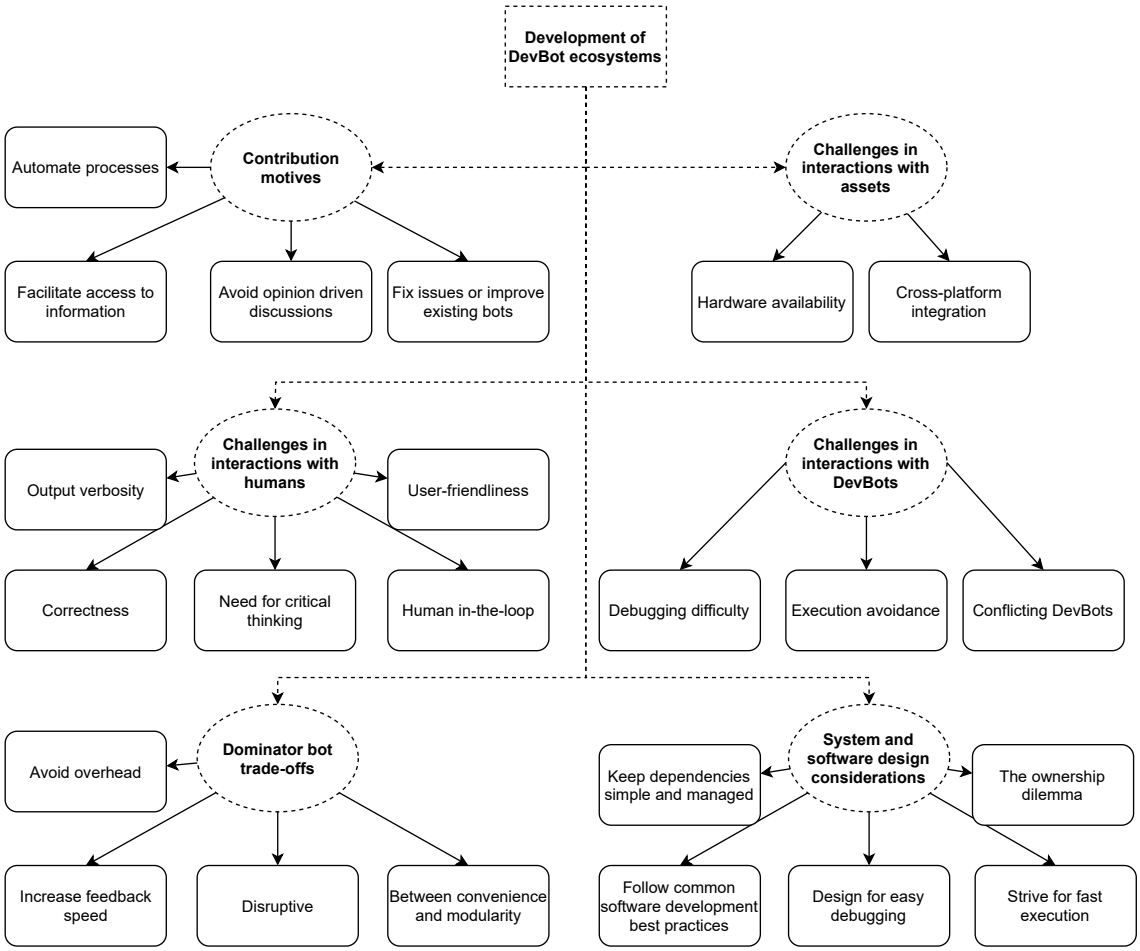


Figure 5.2: Thematic map from interviews with practitioners. Each ellipse represents a question that is mapped to a research subquestion. Moreover, each rounded rectangle represents a theme that emerged from the practitioner testimonies.

5.3.1 Contribution motives

In the process of describing the development of DevBot Ecosystems, we need to start from the beginning i.e. the reasons that triggered software development in the first place. Below, we outline the most common motivations behind contributing to the development of DevBots in the company's DevBot Ecosystem.

Automate processes: The most common motivation, among the interviewees, in regards to the reasons behind their involvement in DevBot development was the wish to automate processes. Particularly, since DevBots are utilized to automate numerous checks and tests both before and after code has been merged, developers found DevBots instrumental in expediting code reviews since laborious checks no longer have to be conducted by humans.

> "*We spend way too much time in code review for things that aren't automated*" - P8

Furthermore, the benefits of automation reaped via the adoption of DevBots were held to high esteem and were deemed as instrumental parts of the general move towards automation in the company.

> "*I (Contributed to DevBot development to) automate as much as possible in order to scale software development*" - P9

Facilitate access to information: Next, practitioners worked on DevBots in the ecosystem so to ease access to information. This information was already generated by tools or the execution of bots, however, it was (i) published in a medium along with other information of lesser importance and relevance, (ii) found in a channel not habitually accessed by the developers, or (iii) was made available in a somewhat inconvenient manner. To put it differently, a catalyst for involvement in DevBot development was the drive to increase the *discoverability* of certain data for the developers.

> "*The script was too slow to run manually, takes several hours, so we integrated it into the CI pipeline (as a DevBot) to follow the progress*" - P1

Moreover, DevBots were used to bring certain events to the attention of the developers, by relaying information between different communication channels, e.g. from the code review software to the one used for chat and all-around discussions.

> "*(Created the DevBot to) provide notification events*" - P7

Avoid opinion-driven discussions: Furthermore, DevBots were perceived as the means to decrease subjective remarks being raised during code reviews, often resulting in long, opinion-driven discussions which could have been avoided by letting a DevBot be the one that concludes such arguments before they start. Specifically,

by having the DevBot enforce coding guidelines and standards, reviewers are on one hand no longer required to pay attention to such details and on the other not *expected* to express a remark over them that could derail the code review process and extend its duration.

> "*(Created the DevBot to) eliminate the human from the (code review) loop*" - P8

Fix issues or improve existing bots: Becoming engaged in DevBot development is often attributed to the need for maintenance work. Particularly, the reasons could be a DevBot misbehaving or room for improvement being identified by the individual developer or a stakeholder external to the team.

> "*It's often feature requests from people to do something, something is not working or working badly or sometimes creating a new job that will do something new*" - P5

5.3.2 Challenges in interactions with assets

DevBots typically need to interact with the surrounding environment and consume or produce various assets. These assets stem from source code, logs generated by data collection vehicles and binaries produced by other bots, to Hardware-In-the-Loop rigs, to messages sent by humans in chat rooms. The interviewees identified some challenges in regards to how DevBots work with assets which are illustrated below.

Hardware availability: Hardware availability was brought up as a recurrent challenge related to interactions of DevBots with assets. As there is a limited number of hardware resources they need to be shared by humans and DevBots. For example, rigs utilized for software verification in an environment that is very close to the production one are utilized by both developers who want to manually test their work but also by DevBots that are running as part of the Continuous Integration pipeline.

> "*It's always tricky with HIL rigs and booking them*" - P5

Furthermore, hardware resources were perceived as being prone to failure due to trivial issues with high impact, such as cabling. These problems are often hard to identify because they often manifest themselves in similar ways to software-related shortcomings. Moreover, to troubleshoot hardware issues it is often required to be physically present near the resource which may be inconvenient or occasionally not even feasible.

> "*As soon as you have physical interfaces it gets difficult, it can be cables in a car or memory shortage on a server*" - P9

Furthermore, while not directly related to the *availability* of hardware resources,

such resources exhibit a degree of variability which influences the way they should be interacted with by DevBots. To put it simply, differences on the hardware level require different handling by software and subsequently, DevBots need to take differences among the hardware platforms into account in order to operate seamlessly across all variants.

> "*There are no identical HIL rigs, they are all unique so we need abstraction layers since we don't want a DevBot uniquely tied to an asset*" - P3

Cross-platform integration: Consuming and producing data for different systems is a commonly witnessed problem with DevBot Ecosystem development. This often occurs when a DevBot needs to consume assets stored on different systems, such as source code from a version-controlled repository and downloadable binaries located on a remote server. Moreover, the DevBot may need to output information on separate systems such as the code review portal and chat rooms. In such instances, DevBots are expected to accommodate for the different communication protocols and APIs of the interconnected systems, which can be viewed as troublesome.

> "*You need to know about the APIs and how to use them when using multiple assets (...) they are not tailored for iterative work*" - P11

Aside of difficulties working with 3rd party APIs, acquiring artifacts necessary for the compilation of the source code, was challenging if the said artifacts were not part of the version controlled repository. Particularly, assets that were not version controlled in the same way as the source code were relatively *transparent* to the build system that could not efficiently determine whether they have been altered and whether their integrity has been maintained across different build stages.

> "*Dependencies to external files not modeled in build configuration system*" - P8

Overall, establishing a concise baseline when it came to dependencies appears to be problematic, when those dependencies are not all contained within the same asset. This was not only the case about software builds but also the software configuration of hardware assets such as the Hardware-In-the-Loop rigs.

> "*Hardware assets are difficult to debug when a HIL rig has different dependencies than what I have on my computer*" - P6

5.3.3 Challenges in interactions with DevBots

Many of the DevBots operating within an ecosystem context are expected to interact with other bots. This implies that there are DevBots that consume relatively *intricate* artifacts produced by other DevBots, such as logs or binaries. That being said, the interaction between bots can ensue on a simpler basis, such as that one DevBot requires one or more others to have successfully executed for it to run.

The most significant issue, based on the frequency it was touched up during the interviews, was dependency management. Ensuring that all preconditions are fulfilled before DevBot execution was often seen as an impediment. This was due to *missing* the systematic means to formulate dependency relationships and therefore ensuring all preconditions for a DevBot's seamless execution are satisfied.

> "*We need to make sure when one bot produces something and someone else wants it, then the other bot should know what it should fetch and make sure that actually exists, which means you need to verify whether a bot actually ran*" - P7

> "*We need a formal way to describe and fulfill dependencies*" - P9

Overall, dependencies between DevBots were considered as a source of complexity and something that should be kept as simple as possible. The reasons behind this mindset were the inability to formally establish and ensure the fulfillment of dependency chains between DevBots. The number of dependencies to other DevBots was explicitly correlated with complexity and instabilities by several interviewees.

> "*(Dependencies between bots) are sometimes necessary, I think we are looking at reducing these dependencies, because they cause a lot of complexities in the system*" - P10

> "*If you have a DevBot that is consuming artifacts from many different sources, it becomes very fragile, the whole ecosystem becomes very fragile*" - P5

Debugging difficulty: Another issue related to interactions between DevBots was the difficulty to debug. Particularly, DevBots that interacted with other DevBots were considered inherently more difficult to debug. Specifically, fulfilling the preconditions for a DevBot to run could be laborious. When executing DevBots manually to determine the cause of an error, it would often take significant time to execute the entire DevBot dependency chain so as to end up in a state where debugging could take place.

> "*You need to run DevBots in a specific order to get the correct result*" - P8

> "*When there is a chain of multiple dependencies it becomes very difficult to reproduce problems locally*" - P4

Furthermore, part of the difficulty to debug could be attributed to the absence of a systematic way to determine which exact snapshot of dependencies were included in a run. In particular, there was no straightforward way to ensure the production of the exact same artifacts, generated by various DevBots during a problematic execution, which would be also present when attempting to reproduce an error. To put it differently, the reproducibility of DevBot runs with long dependency chains

is low, as it is not guaranteed that DevBots will produce or consume the same artifacts across different runs. This leads to the emergence of sporadic failures that are cumbersome to troubleshoot and solve.

> "*Make sure DevBots are synchronized, bots should run from the same baseline*" - P8

> "*(It is challenging to) trigger bots using the same baseline in different time-dependent phases*" - P11

Execution avoidance: Unnecessarily running DevBots prolongs the time required for the DevBots to finish their execution and is partly related to the aforementioned problems with dependency management between DevBots. If a DevBot needs to consume several resources generated by other DevBots, then the said resources should *not* be produced again if there have been no relevant changes. Instead, those dependencies should be cached and quickly fetched without a rebuild and the needless re-execution of the DevBots in the dependency chain to be necessary.

> "*We should push for build and test avoidance, it's easy to set up a bot that always runs*" - P3

It is difficult to determine whether an asset generated by DevBots and stored outside the version-controlled repository, that is eloquently handled by the build configuration system, needs to be regenerated. Specifically, the build configuration system's ability to manage dependencies and cache results is held in high esteem, however, this cannot be said for resources outside its "realm", such as artifacts produced by DevBots and stored outside the version control system.

> "*We are forced to regenerate the model-driven development tool code for every commit. If the code is generated by our build configuration system, then the code wouldn't need to be regenerated unless the model was changed*" - P1

Conflicting DevBots: Conflicts between DevBots were recognized as a typical source of annoyance. They occur when DevBots follow contradicting sets of rules or interpret the same rules in opposing manners. This is often the case when applying coding standards using different tools. For example, fixes proposed by a DevBot utilizing a general-purpose open source static analysis tool may not be considered valid by another DevBot tailored for safety-critical systems in the automotive domain and abides by a stringent set of constraints. Furthermore, a DevBot tasked with linting may contradict a DevBot that formats the code. Such issues can be resolved by manually applying the fixes of each bot in the correct sequence.

> "*One of the biggest challenges we've had is when they are producing conflicting results*" - P8

5.3.4 Challenges in interactions with humans

Multiple bots have humans as their primary stakeholders, therefore need to pay additional attention to how they interact with developers.

Output verbosity: The verbosity of DevBot output was frequently seen as hindering the users from discovering the relevant information. While the DevBot often concisely offer their execution result, this cannot always be said for the reason behind failure. Particularly, developers often need to browse through verbose logs generated by the underlying tools utilized by the DevBots before discovering the part that is relevant to the failure or the problem they are facing.

> "*It's OK for machines to parse thousands lines of code, but for humans there should be just a snippet*" - P11

> "*It should be easy to find information so users shouldn't need to "Ctrl F"*" - P8

User-friendliness: The practitioners that were interviewed expressed a number of concerns regarding the user-friendliness of the DevBots. Specifically, they stressed the importance of DevBots relaying information in a way that is easy to understand. When interacting with humans comes into play, DevBots are expected to communicate in a comprehensive manner that appeals to most developers regardless of technical level and prior experience.

> "*(It is a challenge to) make the bots speak a language the human can understand (...) it has to speak to the very advanced people but also the junior or the new employees*" - P5

Moreover, the user interface to the DevBots is believed to have an impact on their perceived friendliness. A well-designed user interface decreases the cognitive load required for a user to effectively interact with a DevBot.

> "*You need to think of how to create your UI, even if the input to the bot is text, If you (as a DevBot user) need to think of everything all the time, it creates a cognitive load*" - P10

Correctness: Humans need to be able to rely on DevBot results. To be exact the DevBot results need to be correct. While this may sound self-evident, in reality, DevBots which are rumored to occasionally generate false output, often end up being ignored and potentially accurately pointed out issues tend to be accommodated slower. DevBots seen as "cry wolves" tend to needlessly notify individuals over alleged issues that eventually prove to be non-problems. This hurts the trust in the DevBot and can lead to actual complications being neglected.

> "*(We need to) get rid of false positives which sends message to everyone involve and*

> *in the end you turn off that communication channel*" - P9

Need for critical thinking: While DevBot developers strive to render the DevBots as correct as possible, ultimately one should not *blindly* trust them but exercise critical thinking to a certain extent. No matter how low the probability of a false positive is, DevBots are created by humans and therefore are inherently flawed. Even if they are built upon industry-proven software, there can be always valid cases where a developer should not abide by their suggestion and instead consciously choose to diverge from it. While this should not be the case under normal circumstances, the practitioners we interviewed recognized it is a credible scenario and should be taken into consideration when working with DevBots.

> "*Don't rely on the bot too much, it may not always be accurate*" - P2

Interestingly, it was proposed that DevBots influence the way developers work. Once developers begin becoming increasingly reliant on a DevBot, it has implicitly determined their way of working. Therefore, this can be interpreted as developers needing to periodically reassure themselves the DevBot output remains valid.

> "*(DevBots) affect the way of working, they change the way people approach problems*" - P2

Human in-the-loop: Another challenge in regards to the interaction between DevBots and humans was pinpointed in the cases where a human was necessary in evaluating the DevBot output. If the DevBot is involved in a semi-automated process that depends on human input, i.e. produces output that cannot be automatically assessed as to its fitness, then this is seen as a challenge and the DevBot should be potentially treated specially. Such DevBot output may include graphs or other visual artifacts and generally relates to results that cannot be assigned to a distinct binary state, e.g. a success or a failure.

> "*If a human needs to be involved in evaluating the DevBot output, then this DevBot shouldn't run as part of the main flow*" - P3

5.3.5 System and software design considerations

Practitioners involved in DevBot development were asked to describe any software or system design considerations when it came to developing DevBots and rolling out DevBot ecosystems.

Keep dependencies simple and managed: The subject of dependency management resurfaced when discussing DevBot software and system design. The majority of developers stressed the need to keep DevBot dependencies to other DevBots or assets as simple as possible and in any case, there should be a systematic way of describing and fulfilling them. On one hand, DevBots which have a large number

of dependants on them should exhibit a high degree of robustness and availability, because if they misbehave a large part of the ecosystem is affected.

> "*(When developing a DevBot that many others depend on it) we should make sure it doesn't fail because everything depends on it*" - P2

Furthermore, it was often suggested that dependencies should be minimized unless there is a way for them to be formally managed, e.g. through the build configuration system the company uses for building the source code. Having an effective way to manage dependencies would also eliminate the needless execution of DevBots that consume the Continuous Integration pipeline's resources and result in longer feedback loops.

> "*We have to look into run avoidance with formally described dependencies in the build configuration system*" - P9

> "*(When designing DevBots we should) reduce dependencies between bots and have an architecture underneath that helps them work*" - 10

Moreover, not all dependencies can be relied upon. Practitioners suggested that there is DevBot output that cannot be trusted as much as others, therefore this should be taken into consideration when determining whether a DevBot dependency has been satisfied or not. For example, if a DevBot depends on the successful execution of another DevBot which is characterized by inconsistent behavior and false positives when it comes to failures, it should potentially attempt an execution anyway.

> "*(Decrease failure rate due to unsatisfied dependencies by) Stamping DevBot artifacts with different levels of confidence*" - P11

The ownership "trilemma": The "ownership" of DevBots was frequently illustrated as an issue by developers, however, opinions differed on the approach that needs to be adopted. While most agreed that DevBot development often gets neglected and the responsibility of their development or maintenance falls on the shoulders of those with the good-will to welcome it, participants could not agree whether there should be a dedicated team responsible for DevBots or this task to be distributed among different teams who may be considered the "natural" DevBot stakeholders.

> "*(We need) either some people who have the ownership of DevBots, or people more willing to contribute*" - P1

DevBots were considered to "fall through the cracks" since many of them are not considered the sole responsibility of a particular team, but as common code. Fur-

thermore, since DevBots are not explicitly part of the product, but rather utilities that facilitate its development, it is not always easy for them to attract the necessary attention and be given the appropriate priority when it comes to their development and maintenance.

> "*There are very few people in the "gray" area, to do stuff in between and DevBots are not on the Product Owner's table*" - P1

On the other hand, a consensus on how this situation could be remedied was not reached through the responses we received from the practitioners. Three different types of responses were identified: (i) there should be dedicated teams or task-forces that do the heavy-lifting when it comes to DevBot development, (ii) there should be owners mainly responsible with a high-level architecture overview who provide directions, (iii) it is not feasible to assign ownership to particular teams DevBots should be a shared responsibility among teams.

> "*Would be good to have dedicated teams working on these things, right now it's a grass root thing where everyone adds whatever they want*" - P11

> "*We don't have a clear maintenance responsibility, it's based on goodwill. I don't think people should own code, we can have guardians who are concerned with architecture instead*" - P9

> "*It's challenging for a single team to own the DevBots, (there is) no competence within a single team to maintain them*" - P7

To summarize, three different *degrees* of ownership were suggested, from DevBot development being a primary concern of specific teams to increasing awareness about them across the organization and distributing the workload.

Follow common software development best practices: DevBots are software so ultimately common software development best practices apply, according to the practitioners who were involved in the interviews. A plethora of suggestions was brought forward with coding style consistency, code reuse, and abstracting communication layer details, such as external APIs, standing out from the rest. The general feeling was that while DevBot internal software quality is important, adhering to the same high standards as those adopted in the product's source code is not a strict necessity.

> "*Try to reuse code and of course unit test your stuff as much as you can. Try to apply the same coding principles as in regular production code and tools but we don't need to be as strict*" - P8

Avoiding coupling between high-level code and lower-level details was brought up during the discussions. Since DevBots need to communicate with other DevBots and

services that offer access to assets, it is important that those domain-specific details, such as the particular API calls required to fetch a resource, remain abstracted out. This allows the application logic inside the DevBots to remain reusable.

> "*Standardize the communication framework between parts of the ecosystem instead of relying on external REST APIs*" - P11

Design for easy debugging: Difficulties in debugging were illustrated as one of the most significant challenges when developing DevBots. Therefore, a DevBot that can be easily maintained by clarifying what has gone wrong was highly valued by the interviewees. Especially when troubleshooting DevBots with extensive dependency chains, facilitating the discovery of which preconditions are not satisfied, expedited the process to fix the defect.

> "*We need to be able to reproduce things locally to get a faster feedback loop*" - P8

Facilitating debugging is not only applicable to DevBot developers but also DevBot users. Specifically, DevBot users should be given the possibility to determine the cause of a DevBot run failure, similar to their colleagues who developed the DevBots.

> "*(We should have a) surveillance system to know when something goes down, instead of cryptic messages*" - P5

Strive for fast execution: The time it takes for a DevBot to provide its verdict was deemed important by some of the interviewees. Since the ecosystem is comprised of multiple interdependent DevBots and many of them are involved in the Continuous Integration pipeline, the feedback loop needs to be kept as short as possible for the software development activities to seamlessly continue.

> "*Runtime and resource consumption should be considered. We should look for bottlenecks, DevBots need to be fast*" - P4

> "*A bot can't take more than 30 minutes to run, if it takes more than 30 minutes I get annoyed*" - P6

5.3.6 Dominator bot trade-offs

Based on the literature on Software Ecosystems as well as the proposed taxonomy on the DevBot Ecosystem species, Dominator bots are considered an antipattern. Therefore to formulate a comprehensive picture of how does the development of DevBots in ecosystems looks like, we discussed with our interview participants about Dominator bots in general but also specifically about the Dominators they have developed in their ecosystem.

Avoid overhead: A common motivation behind the emergence of dominators was the need to avoid the overhead involved in launching a DevBot. Specifically, DevBots operate in a containerized environment and due to the infrastructure's design, they have to download several resources before their execution. This means that for every DevBot being executed there is a related minimum flat cost of time. By letting a DevBot perform a series of diverse operations, which are *individually* associated with short execution time, the overhead cost of launching multiple DevBots is mitigated.

> "*If we have a DevBot small enough to fit inside the job overhead, then it doesn't make sense to be its own bot*" - P3

> "*There's a trade-off between how fast a bot is and the overhead necessary to set it up*" - P11

> "*(Smaller DevBots should) run in parallel ideally, but in practice there's overhead*" - P5

Increase feedback speed: Overall, the existence of Dominators was tolerated if their benefits outweighed their drawbacks. Particularly, the argument of increased speed was invoked to support the development of Dominators. Similar to the point on avoiding overhead that was previously mentioned, keeping the feedback loop to its minimum duration was considered important enough to create a Dominator bot despite their *theoretical* drawbacks.

> "*I don't mind adding more stuff to the dominator as long as it keeps under 30 seconds*" - P6

> "*Generally, (I prefer) one thing per DevBot, but it boils down to "how much feedback can you get in 5 seconds".*" - P6

Disruptive: Dominators were perceived as potentially disruptive and difficult to work with. Additionally, they were implicated in having a large codebase. Furthermore, the interviewees could recall difficulties when working with or maintaining DevBots charged with a broad spectrum of responsibilities.

> "*The benefit of having many DevBots (instead of one) is that you can replace one without a company crisis*" - P1

> "*The huge unit test DevBot was broken out to find out what was actually wrong*" - P7

Between convenience and modularity: Generally, whether to go for a Dominator or multiple smaller DevBots instead was a matter of balance between the different benefits and drawbacks. In a nutshell, the developers had to decide on the tradeoff between convenience and modularity. Convenience refers to the arguments

regarding avoiding overhead and gaining speed, while modularity is focused on software design concerns, which, in turn, have an impact on the disruptive potential of Dominators.

> "*Ideally we should split Dominators into smaller parts, make those parts do one thing and do it well, be as generic as possible so they can be reused*" - P11

DevBot developers recognized the dilemma they are facing. On one hand, they want to receive the maximum amount of value from their effort while creating and maintaining DevBots. On the other, they realize that software development, design and architecture are not only important for production code but DevBots as well. Therefore they need to abide by best practices and keep DevBot-related technical debt under control.

> "*If the coupling between the different functionalities is very low then just run them in the same DevBot*" - P1

> "*There's probably some kind of balance, we cannot have 1000 bots that do different things*" - P7

RQ3: How can the development of DevBot ecosystems be described?

We summarise the key findings of our study to describe the development of DevBots ecosystems according to the following points:

- **Motivation** stems from the need for automation, accessing information and fixing problems
- **Challenges** arise from lack of hardware availability, output verbosity and lack of execution avoidance
- **Dominators** avoid the overhead of multiple bots but can be disruptive if they misbehave
- **Design considerations** appear similar to production software while dependencies need to be formally managed

6

Discussion

In the following section, we reflect upon the results and combine them to discover further findings. Additionally, various threats to validity are outlined and the relevant mitigation strategies are presented.

6.1 Suggestions on Designing DevBot Ecosystems

When examining the thematic map which emerged from the interviews with the practitioners, a number of "superclusters" with related themes can be highlighted. The first one relates to dependencies, which are an inherent side-effect when synergies between ecosystem actors materialize. Dependency management was raised as a concern when it came to challenges related to (i) interactions between DevBots, (ii) design considerations when developing DevBot ecosystems as well as (iii) when contemplating upon the adoption of dominators. Specifically, it was relayed that unnecessary execution of DevBots can be avoided once an effective way of determining dependencies has been established. Having a formal way of describing dependency chains will facilitate caching of DevBot execution results or artifacts as well as the baseline of other assets a DevBot must consume so to be able to operate. As a consequence, the ability to utilize already existing information from past runs will decrease the need for creating monoliths, i.e. Dominator bots, as well as increase the speed of the feedback loop.

Furthermore, accelerating the time it takes for a DevBot to provide feedback is held in high esteem by the developers of the DevBot ecosystem. Fast DevBot execution is important to expedite source code verification during the development and testing phases which in turn makes debugging issues with the DevBot itself or the related assets easier. Facilitating troubleshooting was pointed out as a significant attribute that should be promoted by a DevBot's software design and can be ultimately correlated to a DevBot's user-friendliness from a DevBot developer standpoint. This is because a DevBot user is often also a DevBot developer. Developers running into problems with DevBots was a common motivation to start contributing to DevBot development. Moreover, user-friendliness does not only concern the user interface characteristics and the ability to easily troubleshoot issues, but also the DevBot output's verbosity. Particularly, our interviewees continuously brought up the issue of *noise* in the logs as a determining factor of the relationship between DevBots and

humans. Overall, we may group execution speed, a design that facilitates debugging and limiting output verbosity under the umbrella of user-friendliness in DevBot ecosystems.

Next, a broader bundle of themes is assembled around DevBot correctness. DevBots need to provide correct information, however, they may be flawed, contain bugs, or abide by a different set of rules than their users would expect. This has two results: (i) conflicting DevBots and (ii) avoiding blind trust in DevBots. The former is often the case when two DevBots either try to perform similar operations, e.g. static analysis and they are either defective or follow different guidelines or the output of one DevBot is incompatible with another. For example, a static analysis tool may apply some fixes which would not be valid according to the DevBot that is tasked with formatting. The latter, i.e. exhibiting critical thinking when reviewing DevBot output is related to DevBots being flawed and therefore not entirely trustworthy. Ultimately, there may always be exceptions to rules that require special handling and cannot be feasibly accommodated by a generic tool. On the other hand, the need for "skepticism" implies that opinions maybe after all necessary. This contradicts, to an extent, one of the major motivations behind developing a DevBots ecosystem in the first place: Avoiding opinion-driven discussions. Practitioners are expected to find a balance between the inherent DevBot imperfections and treating DevBots as authorities to avoid unnecessary discussions or conflicts.

The discussion above includes various aspects that should influence the design of DevBots ecosystems, ranging from deciding on the DevBot's interactions, dependencies, or functionalities. We summarise the discussion above, in the key points below:

- Utilize a dependency management system to avoid unnecessarily running bots and facilitate establishing baselines
- Design for quick execution and user-friendly output to decrease the cognitive load and facilitate debugging
- Minimize false positive or conflicting output to avoid DevBots being ignored or requiring human intervention

6.2 Challenges and motivations compared to literature

While past literature on DevBots is scarce, it provides a reasonable benchmark for this study to compare findings against. Specifically, [6] elaborate on the *benefits* of utilizing DevBots as well as the related *challenges*. These are matched against the *motivations* of contributing to DevBot development as well as the challenges when developing DevBot ecosystems of this study.

Starting with comparing the motivations behind contributing to DevBot ecosystem development and the benefits of DevBots in general, there is a very high degree of alignment between the findings of [6] and those of the current study. Overall,

the benefits of DevBots can be summarized as (i) improving productivity and (ii) offering improvements unfeasible for humans to perform. Specifically, a main benefit of DevBots is their ability to automate tedious tasks. Automating processes was one of the primary motivations behind contributing to DevBot development in this study too.

Next, another benefit of DevBots, according to [6], appears is the expedition of information collection. DevBots can be utilized to either discover new information or integrate information from existing sources. In particular, receiving notifications for various events was outlined as a very common use case of DevBots in the literature. Similarly, the practitioners that were interviewed claimed that one of the driving forces behind developing a DevBot ecosystem was to facilitate access to information. This information already exists, however not in a form that is easily readable, accessible, or discoverable by humans. In fact, in DevBot ecosystems Observer DevBots are responsible for satisfying this purpose, namely transforming and presenting data for human consumption.

Furthermore, DevBots enable developers and organizations to handle tasks at scale. This was also part of a theme related to the motives behind DevBot development that emerged during the interviews. DevBot ecosystem developers contribute to DevBots because they want to automate processes. At the same time, automation was highlighted as the means to scale the software development efforts. In other words, enabling software engineering at a large scale is both a DevBot advantage but also a strong motive behind developing DevBot ecosystems.

Moreover, DevBots are connected with improving quality, as they are quick to verify sets of rules and do it consistently thus eliminating human error. From a DevBot ecosystems perspective, the interviewees suggested that developing DevBots to avoid opinion-driven discussions as well as apply uniform formatting and quality standards, is a significant catalyst of getting code reviews merged. Disagreements over style or going meticulously over each line of code to discover obscure quality violations are a thing of the past since DevBots became responsible for such activities.

Being able to continuously accommodate tasks regardless of the day and time is considered to be a DevBot benefit. This was not explicitly mentioned as a motivation by the practitioners that were interviewed. However, it can be regarded as a vital part of automation, scaling, and collecting information. Subsequently, we consider this DevBot benefit as implicitly present among the reasons which trigger the development of DevBot ecosystems.

The existing literature covers challenges related to DevBot *usage*. As such, there is a lot of alignment with challenges from a DevBot ecosystem development standpoint, especially related to challenges around the interaction of DevBots and humans. The first and one of the most prominent issues in DevBot usage was "interruption and noise". Specifically, there is a need to contain the amount of information exposed to the user as well as the need to involve users in the DevBot process, i.e. by requesting feedback. Similarly, challenges related to DevBot ecosystem development include

output verbosity as one of the major issues around the collaboration between humans and DevBots. DevBot feedback containing uninteresting output, a developer has to skim through, is considered a factor that harms usability. Moreover, a DevBot requiring human input to operate in a semi-automatic manner is perceived as a potential risk, as the human involvement inherently decelerates the entire process and obliges the user to increase the number of times they divert their attention to the DevBot.

Trust was previously outlined as a factor thwarting the usage of DevBots. DevBots providing feedback that contains false positives results in developers being more skeptical towards the said results and therefore spending more time in manual inspections. This concern was apparent in this study, through the *DevBot correctness* and *need for critical thinking* themes. Specifically, developers of DevBot ecosystems pointed out that false-positive output results in a DevBot being perceived as unreliable and potentially increasing the probability of being ignored by the developers. Moreover, several practitioners involved in DevBot ecosystem development advise against completely trusting some DevBots. According to them, critical thinking is always necessary to an extent when dealing with DevBot results and feedback as they cannot be fully trusted.

Usability is a general challenge related to DevBot development. In [6], this was primarily a concern of DevBots that communicate via a natural language interface. From a DevBot ecosystem perspective, developers believe that DevBots should speak a language humans understand to decrease the cognitive load on their users. Similarly, in [6] it was pointed out that DevBots should appeal to stakeholders regardless of technical ability. A relevant suggestion was discussed in the interviews; DevBots should be able to provide output that is fit for developers irrespective of experience level and familiarity with the project.

While most challenges in [6] focused around interactions of DevBots with humans, one point was related to the interaction between DevBots. In particular, it was reported that DevBots may cause problems when they interfere with each other. When viewing things from an ecosystem perspective, one of the challenges practitioners brought up was conflicting DevBots. As previously discussed, DevBots may conflict with each other if they perform similar operations, e.g. static analysis or formatting, but abide by different rule sets or if the output of one DevBot is not compatible with the directives of another. Interestingly, inconsistent DevBot contributions compared to the project's standards in regards to formatting were also a problem in [26].

In summary, when DevBots are investigated through an ecosystem perspective, challenges exhibit a high degree of diversity. There are three dimensions of challenges to take into consideration that were relevant for our study but were not, necessarily, emphasised/connected in existing DevBot literature. Those dimensions are (i) Assets, (ii) other DevBots, (iii) humans. In this study, every challenge mentioned in [6] and anticipated to be prevalent in DevBot development was encountered. As expected, these challenges mostly revolved around interactions between DevBots

and humans.

6.3 Reflections on Dominator DevBots

Dominators, in the context of Software Ecosystems (SECOs) seek to assimilate other players or remove them from the ecosystem and serve their role [16]. Unless constrained, they eventually tend to suffocate the ecosystem and their long-term sustainability is limited. In DevBot ecosystems, dominators tend to accumulate functionality, resulting in monoliths with a high potential for disruption. The broad functionality they encompass instills them the capacity to become a bottleneck, as a considerable amount of DevBots may depend on the dominator.

Despite the dominator, DevBot theorized drawbacks, developers recognize they have practical benefits. Eventually, the decision between creating or expanding a dominator and deprecating a dominator boils down to a trade-off between the convenience of a monolith and the known benefits of modular software design. To begin with, favoring a dominator over individual bots that collectively offer the same functionality can be justified in the name of speed. Launching DevBots implies a fixed cost in time, for the container in which the DevBot typically operates to be instantiated and the necessary dependencies to be fetched. If the execution of a DevBot lasts shorter than the time it takes for it to be initiated, then it is sensible to incorporate it in a larger entity, i.e. a dominator. The dominator ends up including a diverse set of operations which independently are characterized by short execution time and few or *common* dependencies.

Moreover, when dependencies have to be produced by a different DevBot, the mechanism to transfer those resources between DevBots entails a computational as well as temporal toll. For example, for a binary produced by one DevBot to be executed by another, this binary would have to be uploaded to some server and then fetched by the DevBot that needs to execute it. In the absence of an effective caching mechanism that would allow the bot which produces the dependency to not always be run, or when we cannot avoid running the DevBot, it may be convenient to merge the two DevBots into a single one. This would enable us to avoid the overhead of transmitting and receiving dependencies between DevBots.

At the same time, developers recognize the problematic nature of dominators. Dominators can be disruptive; a failing or misbehaving dominator usually has a disproportionately large impact and cause a "crisis" in the company. Furthermore, their extensive functionality may end up hard to debug. In fact, practitioners made note of a dominator which was deprecated and its functionality refactored into different bots, due to difficulties in maintaining it.

While the case of dominators in SECOs appears to be tangled with negative connotations, this is not necessarily the case for the dominators of DevBot ecosystems. Dominator DevBots exhibit undeniable positive traits. Despite their monolithic nature hinting towards technical debt in regards to the design of the DevBot ecosys-

tem, they may seasonably be appropriate. As long as developers maintain a middle ground between the opportunistic approach that offers convenience and modular software design, the potential adverse effects of dominators in DevBot ecosystems can be mitigated or remain within manageable levels.

6.4 Threats to validity

Recognizing and outlining the various threats to the validity of a qualitative study such as the present one, as well as the methods employed to mitigate them are important to present. They enable the reader to understand that the authors conducting the research are required, occasionally, to be subjective and make use of their personal perspective to interpret as well as respond to the environment [27] while the research is proceeding. To classify the different validity threats for software engineering studies, the four dimensions of validity, as presented in [22] are adopted: (i) construct validity, (ii) internal validity, (iii) external validity and (iv) reliability. In the following subsections, the threats to the different validity aspects will be described and the mitigation techniques that were adopted will be presented.

6.4.1 Construct validity

Construct validity is the extent to which the selected means of interpreting a phenomenon are appropriate for the particular purpose. In the current study, missing to identify a species of the DevBot ecosystem can skew the ability to describe DevBot ecosystems as well as classify individual DevBots to certain species. That being said, the proposed taxonomy is faceted which means that it is easily extendable. Future revisions of the taxonomy may include additional species if such are identified.

Furthermore, in the literature, there is not a clear consensus on what a bot and particularly a DevBot is. Therefore one should not assume the participants in the interviews to be in alignment with the study's definition of the terms, which would result in the interview responses diverging from the intended subject. To tackle this threat to the construct validity, the way to define what is a DevBot presented in [6] was explained to them along with concrete examples from their particular project.

6.4.2 Internal validity

Internal validity is the probability of a phenomenon being caused by the factors under investigation and not by different ones. To put it differently, internal validity measures the trust in the causal relationships researched in the study. Since the study was focused on evaluating the proposed taxonomy, describing the status quo through observations as well as practitioner testimonies collected via interviews, there are no particular cause-and-effect links that were studies that could introduce risks to the internal validity of this work.

6.4.3 External validity

External validity is the extent to which results and conclusions of the current study can be applied to a broader context, i.e. generalized. In other words, external validity measures the fitness of the reusability of this study's outcomes by different researchers and practitioners. To begin with, the fact that the case study has been conducted involving software and developers from a *single* company can be considered as a risk to the ability to generalize the study's different results. This is indeed a potential threat, however, the proposed taxonomy does not contain any company or domain-specific details and should be applicable in different contexts. The same can be claimed for the different motives, challenges and design considerations when developing DevBot ecosystems. Specifically, the different motives and challenges related to developing DevBots were also encountered in previous literature, while the design and architectural remarks were technology-agnostic.

Furthermore, researcher bias may be undermining the external validity of the study, due to the author being an employee of the company where the case study was conducted. This would make the different participants in the interviews the author's colleagues as well as the different software that was studied, parts of the author's daily professional engagement. Having an existing relationship with the interviewees may have an effect on their responses during a semi-structured interview. This risk was partly mitigated by verifying the interview guide with an academic supervisor as well as not selecting any participants that work in the same team as the author. Furthermore, being already familiar and potentially have preconceptions about the software artifacts that were studied may influence the result extraction process. To avoid this threat, DevBots that the author had personal involvement in their development were excluded from the study.

6.4.4 Reliability

Reliability refers to the degree of coupling between a study's results and the particular researchers involved in it. In other words, future researchers following the same methodology should reproduce the results and conclusions. For instance, the theme analysis which followed the interviews is regarded as a potentially risky activity in regards to the reliability of the current work. Other researchers may code the input from the interviews differently and end up with a divergent thematic map. To mitigate this, an academic supervisor was involved to triangulate [28] the results of the theme analysis and ensure that relationships between different themes were reasonably interpreted. Moreover, a late draft of this study was sent along with an executive summary to the interviewees. Additionally, three of them participated in a follow-up meeting where the proposed taxonomy, the classification of the company's DevBots as well as practitioner insights on DevBot ecosystem development were presented. The practitioners expressed interest in the findings and did not raise any objections.

A threat to the reliability of this study is the small number of practitioners (11) that were interviewed. While the number of participants was low, they represent some

of the most active developers on DevBots, therefore their opinion is regarded as highly significant. Furthermore, all interviewees belonged to different teams. This threatens the reliability of the results since if there is no common ground between the practitioners we interview, then it may not be feasible to combine their testimonies and extract results. In other words, the input received from the interviews may be randomly heterogeneous. This risk was mitigated by selecting participants from the same project, therefore the different practitioners in the study share common references despite belonging to different teams.

7

Conclusion

This study delves into the novel topic of DevBot ecosystems. Bots that facilitate software development are increasing their presence in modern software development workflows. DevBots, cooperate as well as with humans to perform a variety of tasks, related to quality assurance, resource monitoring and testing among others. They may not be the focus of the software development efforts for the majority of a software organization, however, they have become essential utilities [29], [30]. They relieve developers from a plethora of responsibilities so that they can instead focus on the core business offering of their company.

Specifically, the study introduces the term itself; DevBot ecosystems. The term, inspired by Software Ecosystems, is accompanied by a taxonomy that characterizes the different roles of DevBots found in the said ecosystem. The taxonomy is evaluated via a case study conducted in an automotive supplier that delivers the full software stack for autonomous driving vehicles. The case study also offers a unique insight into the different DevBot ecosystems active in the organization as well as a chance to record practitioner testimonies on developing DevBot ecosystems.

In regards to the proposed taxonomy, four distinct roles, or "species", are attributed to DevBots from an ecosystem perspective: (i) Keystone, (ii) Niche, (iii) Dominators, (iv) Observers. The former three are inspired by the different players found in Software Ecosystems and the latter from biology, where scientists observe ecosystems. *Keystone DevBots* primary concern is offering value to other DevBots in the ecosystem. Their output may be occasionally useful for humans too, but the primary consumer is other DevBots. On the other hand, *Niche DevBots* cater primarily to humans, while they may or may not consume data generated by other DevBots. Next, the key characteristic of Dominators is that they are non-atomic in nature. They can be seen as a composition of functionalities and accommodate a plethora of use cases that could be otherwise served by multiple bots. Observer bots' primary directive is to refine existing information and make it fit for human consumption. These bots characteristically do not generate any new information, merely transform it.

DevBots from all species were identified inhabiting the different ecosystems established in the "savanna", i.e. the software development project that was investigated in this study. Specifically, DevBot ecosystems get formed around different assets.

For example, there is an ecosystem around the product's source code, another around Hardware-In-the-Loop rigs and data collection vehicles etc. While all assets are to some degree related, since we are talking about the development of the same product, DevBots in different ecosystems typically do not interact with each other. That being said, some DevBots are active in more than one ecosystem which are called Bridge DevBots.

11 developers who contribute to DevBot ecosystems were interviewed and shared insights in regards to the motives behind their involvement, the typical challenges related to DevBot ecosystem development as well as system and software design considerations. The motives behind the practitioners' involvement, matched the known DevBot benefits as outlined in previous literature. Moreover, this also happened with challenges in regards to interaction between DevBots and humans. That being said, a lot of discoveries were made in regards to DevBot development challenges from the unique aspects that an ecosystem perspective offers. Specifically, a noteworthy amount of impediments was brought up concerning the interactions between DevBots as well as DevBots and assets of the ecosystem. Issues such as hardware availability when handling assets or the desire to avoid execution of DevBots that produce data consumed by other DevBots was frequently stressed during the discussions.

Furthermore, when designing DevBot ecosystems, practitioners emphasized the importance of keeping the dependencies between DevBots simple and maintained, adopting a design that facilitates easy debugging as well as following similar software development best practices to those prevalent in software meant for production. Additionally, the topic of Dominator DevBots was contemplated upon and the general feeling was that dominators can be beneficial to the ecosystem as long as they are managed and risks involved with disruptive potential are acknowledged and controlled.

The larger the software development scales, the higher will be the number of DevBots that get adopted. After a critical mass is reached, DevBots should be viewed from an ecosystem perspective as synergies, dependencies and relationships begin to crystallize. This means that to move forward, a two-tailed challenge is encountered. On one hand, practitioners should start viewing DevBot development from an ecosystem perspective and start developing DevBots taking into consideration not only the interactions between DevBot and humans but also those between DevBots as well as DevBots and assets. On the other hand, we propose the academia look further into the proposed taxonomy and further refine or extend the DevBot ecosystem species according to the new findings.

Bibliography

[1] L. Erlenhov, F. G. de Oliveira Neto, R. Scandariato, and P. Leitner, "Current and future bots in software development," in *2019 IEEE/ACM 1st International Workshop on Bots in Software Engineering (BotSE)*, pp. 7–11, IEEE, 2019.

[2] M.-A. Storey and A. Zagalsky, "Disrupting developer productivity one bot at a time," in *Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering*, pp. 928–931, 2016.

[3] M. Wessel, B. M. De Souza, I. Steinmacher, I. S. Wiese, I. Polato, A. P. Chaves, and M. A. Gerosa, "The power of bots: Characterizing and understanding bots in oss projects," *Proceedings of the ACM on Human-Computer Interaction*, vol. 2, no. CSCW, pp. 1–19, 2018.

[4] H. H. Olsson and J. Bosch, "Going digital: Disruption and transformation in software-intensive embedded systems ecosystems," *Journal of Software: Evolution and Process*, vol. 32, no. 6, p. e2249, 2020.

[5] C. Salzmann and T. Stauner, "Automotive software engineering," in *Languages for system specification*, pp. 333–347, Springer, 2004.

[6] L. Erlenhov, F. G. d. O. Neto, and P. Leitner, "An empirical study of bots in software development–characteristics and challenges from a practitioner's perspective," *arXiv preprint arXiv:2005.13969*, 2020.

[7] H. S. Nwana *et al.*, "Software agents: An overview," *Knowledge engineering review*, vol. 11, no. 3, pp. 205–244, 1996.

[8] S. Green, L. Hurst, B. Nangle, P. Cunningham, F. Somers, and R. Evans, "Software agents: A review," *Department of Computer Science, Trinity College Dublin, Tech. Rep. TCS-CS-1997-06*, 1997.

[9] K. Sycara and D. Zeng, "Coordination of multiple intelligent software agents," *International Journal of Cooperative Information Systems*, vol. 5, no. 02n03, pp. 181–211, 1996.

[10] R. A. Flores-Mendez, "Towards a standardization of multi-agent system framework," *XRDS: Crossroads, The ACM Magazine for Students*, vol. 5, no. 4,

pp. 18–24, 1999.

[11] E. Knauss and I. Hammouda, "Eam: Ecosystemability assessment method," in *2014 IEEE 22nd International Requirements Engineering Conference (RE)*, pp. 319–320, IEEE, 2014.

[12] I. Van Den Berk, S. Jansen, and L. Luinenburg, "Software ecosystems: a software ecosystem strategy assessment model," in *Proceedings of the Fourth European Conference on Software Architecture: Companion Volume*, pp. 127–134, 2010.

[13] P. R. J. Campbell and F. Ahmed, "A three-dimensional view of software ecosystems," in *Proceedings of the Fourth European Conference on Software Architecture: Companion Volume*, ECSA '10, (New York, NY, USA), p. 81–84, Association for Computing Machinery, 2010.

[14] J. Joshua, D. Alao, S. Okolie, and O. Awodele, "Software ecosystem: features, benefits and challenges," *International Journal of Advanced Computer Science and Applications*, vol. 2, 2013.

[15] D. L. Martin, A. J. Cheyer, and D. B. Moran, "The open agent architecture: A framework for building distributed software systems," *Applied Artificial Intelligence*, vol. 13, no. 1-2, pp. 91–128, 1999.

[16] S. Jansen, S. Brinkkemper, and A. Finkelstein, "Business network management as a survival strategy: A tale of two software ecosystems.," *Iwseco@ Icsr*, vol. 2009, 2009.

[17] C. Lebeuf, M.-A. Storey, and A. Zagalsky, "Software bots," *IEEE Software*, vol. 35, no. 1, pp. 18–23, 2017.

[18] M. Wessel and I. Steinmacher, "The inconvenient side of software bots on pull requests," in *Proceedings of the IEEE/ACM 42nd International Conference on Software Engineering Workshops*, pp. 51–55, 2020.

[19] D. Geer, "Malicious bots threaten network security," *Computer*, vol. 38, no. 1, pp. 18–20, 2005.

[20] C. Shao, G. L. Ciampaglia, O. Varol, A. Flammini, and F. Menczer, "The spread of fake news by social bots," *arXiv preprint arXiv:1707.07592*, vol. 96, p. 104, 2017.

[21] T. Kinsman, M. Wessel, M. A. Gerosa, and C. Treude, "How do software developers use github actions to automate their workflows?," *arXiv preprint arXiv:2103.12224*, 2021.

[22] P. Runeson and M. Höst, "Guidelines for conducting and reporting case study research in software engineering," *Empirical Software Engineering*, vol. 14,

p. 131, Dec 2008.

[23] M. Usman, R. Britto, J. Börstler, and E. Mendes, "Taxonomies in software engineering: A systematic mapping study and a revised taxonomy development method," *Information and Software Technology*, vol. 85, pp. 43–59, 2017.

[24] V. Braun, V. Clarke, and G. Terry, "Thematic analysis," *APA handbook of research methods in psychology*, vol. 2, pp. 57–71, 2012.

[25] P. Sedgwick, "Convenience sampling," *Bmj*, vol. 347, 2013.

[26] C. Brown and C. Parnin, "Sorry to bother you: designing bots for effective recommendations," in *2019 IEEE/ACM 1st International Workshop on Bots in Software Engineering (BotSE)*, pp. 54–58, IEEE, 2019.

[27] J. Maxwell, "Understanding and validity in qualitative research," *Harvard educational review*, vol. 62, no. 3, pp. 279–301, 1992.

[28] K. Jonsen and K. A. Jehn, "Using triangulation to validate themes in qualitative studies," *Qualitative Research in Organizations and Management: An International Journal*, 2009.

[29] R. S. Geiger, "Are computers merely" supporting" cooperative work: towards an ethnography of bot development," in *Proceedings of the 2013 conference on Computer supported cooperative work companion*, pp. 51–56, 2013.

[30] P. Hukal, N. Berente, M. Germonprez, and A. Schecter, "Bots coordinating work in open source software projects," *Computer*, vol. 52, no. 9, pp. 52–60, 2019.

A

Appendix 1