Current and Future Bots in Software Development

Linda Erlenhov, Francisco Gomes de Oliveira Neto, Riccardo Scandariato and Philipp Leitner
Software Engineering Division,
Chalmers | University of Gothenburg,
Gothenburg, Sweden,
{linda.erlenhov|gomesf|philipp.leitner|riccardo.scandariato}@chalmers.se

***Abstract*—Bots that support software development ("DevBots") are seen as a promising approach to deal with the ever-increasing complexity of modern software engineering and development. Existing DevBots are already able to relieve developers from routine tasks such as building project images or keeping dependencies up-to-date. However, advances in machine learning and artificial intelligence hold the promise of future, significantly more advanced, DevBots. In this paper, we introduce the terminology of contemporary and ideal DevBots. Contemporary DevBots represent the current state of practice, which we characterise using a facet-based taxonomy. We exemplify this taxonomy using 11 existing, industrial-strength bots. We further provide a vision and definition of future (ideal) DevBots, which are not only autonomous, but also adaptive, as well as technically and socially competent. These properties may allow ideal DevBots to act more akin to artificial team mates than simple development tools.**

I. Introduction

Even though research on developer productivity is gaining traction [1], it is unlikely that we will be able to keep up with the increasing demand for software without making more, and better, use of artificial intelligence (AI) and automation. Hence, it is unsurprising that we have recently seen bots for software development gaining traction in industry [2] and research [3]. Bots are used, among many other tasks, to automate routine tasks in CI systems (e.g., building project images), to greet and brief newcomers to open source projects, or to automatically generate CSS stylesheets for HTML pages. In this paper, we will use the term "DevBots" as a shorthand for "bots for software development".

We observe that current DevBots are quite different from bots in other domains. Where voice-activated personal assistants (e.g., Apple's Siri or Microsoft's Cortana) use fairly sophisticated AI techniques to generate a social experience for users, most DevBots hardly interact with human developers at all. Where voice assistants are quite general (and, at least to a limited extent) able to learn, many DevBots are little more than glorified scripts, with hardcoded functionality and little intelligence.

In this paper, we contribute two-fold to the state of research. Firstly, we provide a facet-based taxonomy of existing ("contemporary") DevBots, orthogonal to the one provided by Lebeuf [4]. Secondly, we provide a definition and vision of future ("ideal") DevBots, which go beyond the current state of the art, using AI to provide a social experience similar to voice assistants.

II. Related Work

Using automated tools to assist humans in software development was partly explored in the 1990's under the moniker of software agents [5], but lack of sufficiently powerful computing resources where obstacles to the development of AI related services and, consequently, ambitious agent-based systems never became mainstream [6]. Interface agents [6] have acted as successful digital personal assistants that optimise their user's time, learning from user actions to assist with emails and scheduling/rescheduling meetings [7]. However, actually assisting in core software engineering tasks, such as coding or testing, has to the best of our knowledge not been explored in the context of agent-based systems. Nonetheless, industrial usage of, as well as academic research on, bots supporting specifically software development tasks is getting more prevalent. Wessel et al. [2] studied the usage of bots on GitHub, and found that 26% of projects used bots for a variety of different tasks. Repairnator [8] is a program repair bot that monitors test failures in continuous integration environments. Very recently, SapFix[1], an AI-based debugging bot, has been announced by Facebook's research department. Scandariato et al. [9] present future directions of a bot that would take security goals as input and then start from known secure designs in order to evolve plausible ways of solving the security goals.

As a consequence of the advances in the field, proposed taxonomies can assist researchers and practitioners to better understand the opportunities and risks of applying bots, or AI tools in general, to software engineering tasks [10], [3], [11]. For instance, the AI-SEAL taxonomy [10] relies on three facets to classify AI applications in SE. The facets are high level and not exclusive to bots. The goal is to expose the risks of AI applications depending on when practitioners choose to adopt a technology (e.g., before deployment) and what is the level of autonomy of the AI technology. Most relevant to our present work, Lebeuf proposes a taxonomy to classify software bots [4]. Unlike DevBots, "software bot" is a more inclusive term to bot applications, hence not being focused on software engineering activities, artefacts and roles. Lebeuf's taxonomy is extensive, covering 22 facets organized into three dimensions describing: i) the surroundings in which the bot lives and operates (Environmental), ii) the internal properties

[1] https://code.fb.com/developer-tools/finding-and-fixing-software-bugs-automatically-with-sapfix-and-sapienz/

978-1-7281-2262-5/19/$31.00 ©2019 IEEE
DOI 10.1109/BotSE.2019.00009

of the bot itself (Intrinsic), and iii) the bots interactions with its environment (Interaction). Conversely, our proposed taxonomy is simpler. It comprises 4 facets only, which are orthogonal to Lebeuf's taxonomy. For instance, Lebeuf proposes a facet "Knowledge", that distinguishes the bots capabilities in store and access knowledge (sub-facet Memory) originated from various sources (sub-facet Source), both combined with the bot's ability to modify its own behaviour (sub-facet Adaptability). Our taxonomy covers that classification in the Adaptation sub-facet, but under coarser-grained criteria focused on software engineering aspects. Consequently, we argue that our DevBot taxonomy becomes a more suitable instrument for practitioners to understand and adopt different bots.

III. DevBot Taxonomy

We have two goals with our taxonomy: i) support classification of bots under few and simple facets, and ii) get an overview of the variety of DevBots being used by practitioners and researchers. We argue that a simpler taxonomy is more attractive to practitioners interested in adopting DevBots in their development processes, whereas the latter goal provides the main aspects being addressed by current DevBots. We created our DevBot taxonomy in several steps that begins with identification and extraction of data from literature, followed by design and construction using a faceted analysis [12], and ultimately, validation via utility demonstration [13]. The early-stage nature of our field of study complicates applying standard systematic data collection procedures. Hence, we opted for pragmatic data collection procedure. We do not claim that the data set is representative of all existing DevBots. However, we argue that the data set (presented in Table I) is sufficient as a starting point for constructing an initial, extensible taxonomy. The DevBots collected thereby are a mix of previously known bots, bots found by searching the internet and found via advertisement in social media. The criteria for them being included was that they were involved in some type of software development task.

A facet-based taxonomy is a multi-dimensional classification, where each branch in the taxonomy tree refers to a single dimension of classification [13]. That is, every concrete DevBot in our dataset will take a value (i.e., one of the leaf nodes in the tree) for each branch in the tree. This puts a facet-based taxonomy in clear contrast to a more traditional hierarchical taxonomy, where each instance can only take the value of a single leaf node. Facet-based taxonomies have been found particularly useful for emerging fields [13], such as ours. We validate our taxonomy by illustrating its classification capabilities (Table II), e.g., exposing the limitations of facets.

A. Facet-Based Taxonomy

We discuss the different levels of our taxonomy (Figure 1) within each facet in terms of different elements. We consider that the DevBot, on one end, accesses and interacts with a system (e.g., computer environments, software repositories), whereas on the other end, there will be a DevBot user, i.e., a human with a role such as developer or tester. We also distinguish between the DevBot user and its creator/maintainer.

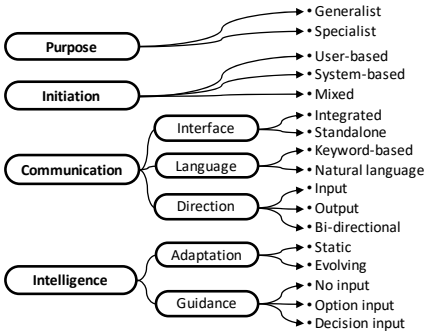

Fig. 1. Our proposed DevBot taxonomy with facets and respective levels.

1) Purpose: This facet contains two levels: *specialist* and *generalist* bots. Each level describes, respectively, whether the DevBot is built for one specific or several goals. We refer to goals as a composition of distinct tasks, such that the same task can belong to many goals. For instance, the goal of producing a patch for an error can be composed of i) recreating the error, ii) finding a suitable patch and iii) create a request to merge the code. Note that task 'iii' can also be a part of another goal, such as pushing a new test to the repository.

As an example of a generalist DevBots, Mary-Poppins can be extended with several plugins to perform different goals, such as responding to merge request with a checklist or prioritize issues based on labels. In contrast, the First-timers is a specialist bot whose only goal is to create a new issue when a branch with the name "first-timers*" is created. As a consequence to the development process, creators of generalists DevBots can reuse several tasks to compose distinct goals and expand their DevBots, whereas, specialist DevBots ultimately could handle larger and less modular tasks.

2) Initiation: The three levels within this facet represent the different actors triggering the DevBots activities. *User-based* DevBots are triggered by their user, whereas *system-based* DevBots are triggered by the environment (e.g., a build server). Finally, some DevBots can be triggered by both, its user and the environment (*mixed*).

These levels highlight the direct or indirect control the user will have over "when" the DevBot should act, which can affect the design of the DevBot. For instance, CSSRoster is initiated when the user pushes a button (user-based), while Spotbot's initiation is scheduled (e.g., using webhooks).

3) Communication: As DevBots are expected to actively contribute to development, communication with its user becomes a complex and essential element in our taxonomy. We defined three sub-facets comprising different aspects of that communication, namely: Direction, Language and Interface.

The **Direction** sub-facet describes the direction of the DevBot communication with its user. The *input* level includes DevBots that listen to user input but do not respond, whereas the opposite is classified in the *output* level. Alternatively,

TABLE I
DESCRIPTION OF ANALYZED DEVBOTS.

DevBot	Url	Description	Open Source
Dependabot	https://dependabot.com/	Automatic updates of your dependencies	No (Free for OS projects)
Greenkeeper	https://github.com/marketplace/greenkeeper	Automatic updates NPM dependencies	No (Free for OS projects)
Spotbot	https://spotbot.qa/	Takes screenshot of pages and looks for issues	No
Imgbot	https://imgbot.net/	Automates optimization of images in a project	No (Free)
Deploybot	https://deploybot.com/	Deploys your build	No
Repairnator	https://github.com/Spirals-Team/repairnator	Automatically repairs build failures on Travis CI	Yes (MIT)
First-timers	https://github.com/apps/first-timers	Creates starter issues for beginners in open source projects	Yes (Apache 2.0)
CssRooster	https://huu.la/ai/cssrooster	Writes CSS classes for HTML with Deep learning	No
Mary-Poppins	https://github.com/mary-poppins	Keeps your merge requests and issues tidy by different plugins	Yes (MIT)
Typot	https://github.com/chakki-works/typot	Fixes typos in merge requests	Yes (Apache 2.0)
Marbot	https://marbot.io/	Manage CloudWatch alerts	Yes(Apache 2.0)

DevBots reacting to input and responding to users are classified as *bi-directional*. These different levels affect the DevBot design, such that output and bidirectional DevBots should make use of user experience principles so that users can better receive and understand the output of the DevBot. Typically, the effort in designing input DevBots is mostly associated with the mechanism for accurately understanding the user's instruction.

The **Language** sub-facet describes how human-like the language used by the DevBot is. Even though this sub-facet can accommodate several levels (arguably even a scale), we decided to simplify the classification to only two levels, namely whether using the DevBot requires knowledge of a specific syntax (*keyword-based*) or whether users can communicate with the DevBot through *natural language*.

The difference between those levels affect both the DevBot user and its creator. Developing a keyword-based language is simpler for a creator, but it means that the user needs to be familiar with a specific syntax in order to interact with the DevBot. The opposite holds for natural language, where usage of the DevBot is simpler, but building such a bot is more intricate. However, most of the bots we examined use natural language for output messages, where the language is simply template based which is not difficult for the creator to make. An example of this is Dependabot that creates a merge request containing natural language.

The **Interface** sub-facet represents the dependency between the DevBot and its surrounding infrastructure, such as specific branded tools. We classify DevBots as *integrated* if they require specific tools to execute, or *standalone* if they can be integrated with different tools via, e.g., API or plugins. An integrated DevBot is often simpler to maintain since it is built for one main application, but users are dependent on specific tools or platforms.All of the DevBots we analyzed were integrated in some way (most of them through Github or Slack) such that one cannot choose another type of version control system or chat application. However, some DevBots at least are operable with a variety of different systems. Deploybot, for instance, supports multiple cloud providers and version control systems, but only uses Slack for notifications.

4) Intelligence: This facet describes how the DevBot handles and uses information to perform its tasks. There are two sub-facets, namely Adaptation and Guidance. The **Adaptation** sub-facet has two levels representing how adaptable the knowledge of the DevBot is. The *evolving* level comprise DevBots who are able to learn, i.e., whose behaviour changes when acquiring new knowledge. This is in contrast to the *static* level, where the DevBot acts only according to predefined knowledge. Note that even a static DevBot may use advanced machine learning, e.g., to learn rules from data, but it does not update these rules during operation.

The different levels affect the DevBot design as, for instance, evolving DevBots require mechanism to acquire new knowledge during runtime. Interestingly, we found no DevBots in the evolving level, but a variety of DevBots with different static knowledge, both scripted and trained. As an example, CSSRooster has been trained on 1000 websites on how class names were given to various elements but, once trained, these rules do not change.

The **Guidance** sub-facet describes what type of input, if any, the DevBot needs from humans in order to achieve its goal. Note that there may be different paths to reach a goal. When the user provides different options and the DevBot autonomously decides which option is the "best" one to take in order to continue, we classify it as a *option* input DevBot. Alternatively, if the DevBot comes to a halt and provides different options for the user to select from in order to continue we classify this as a *decision* input DevBot. Finally, there are DevBots which require *no* input from users, i.e., they run entirely autonomously. An example of a decision input DevBot would be Typot, which may detect a typo in a merge request and presents several modification candidates that the user then selects from. In contrast, Spotbot does not need any input, and simply pings its user on Slack whenever it found an error.

IV. A VISION OF FUTURE DEVBOTS

So far, we have analyzed and classified existing DevBots. These *contemporary* DevBots deliver value, but they lag behind bots in other domains, such as voice-activated personal assistants. Where voice assistants are quite general, contemporary DevBots are largely specialist; where voice assistants are, or at least pretend to be, social, contemporary bots rarely interact with human developers at all. Hence, we now

TABLE II
CLASSIFICATION OF DEVBOTS.

DevBot	Purpose	Initiation	Communication			Intelligence	
			Language	Interface	Direction	Adaptation	Guidance
Dependabot	Specialist	System based	NL	Integrated	Output	Static	Decision
Greenkeeper	Specialist	System based	NL	Integrated	Output	Static	Decision
Spotbot	Specialist	System based	NL	Integrated	Output	Static	No
Imgbot	Specialist	System based	NL	Integrated	Output	Static	No
Deploybot	Specialist	Mix	NL	Integrated	Bi-directional	Static	No
Repairnator	Specialist	System based	NL	Integrated	Output	Static	Decision
First-timers	Specialist	System based	NL	Integrated	Output	Static	No
CssRooster	Specialist	User based	-	Integrated	Output	Static	No
MaryPoppins-bot	Generalist	System based	NL	Integrated	Output	Static	No
Typot	Specialist	System-based	NL	Integrated	Output	Static	Decision
Marbot	Specialist	Mix	NL	Integrated	Bi-Directional	Static	Decision

introduce a working definition of *ideal* DevBots, which go beyond the current state of the art:

> An ideal DevBot is an artificial software developer which is autonomous, adaptive, and has technical as well as social competence.

The underlying philosophy of this definition is that an ideal DevBot should be seen as an artificial developer more than a tool. Within the practical limits set forth by the state of the art in artificial intelligence, we should have similar expectations on a future DevBot than on any human team member.

Autonomous: an ideal DevBot is largely self-sufficient. Once enabled, it does not have to be triggered explicitly, but decides on its own when a task that it can handle arises. Repairnator [8], for example, is not explicitly invoked for each broken build. Instead, the bot autonomously monitors the build system, decides on its own if there are broken builds that it could potentially fix, and then proposes a fix as a pull request. Part of this is also that an ideal DevBot needs to be able to decide when it *cannot* handle a task. In Repairnator, the bot decides through testing that a fix actually works before actually proposing a pull request.

Adaptive: an ideal DevBot should learn from previous attempts at solving comparable tasks. It should implement a closed-loop self-optimizing system, and get better at its task in the context of a project. Learning should happen on two levels: (1) Adaptation through implicit feedback, coming either from human developers (e.g., a DevBot which proposes pull requests that keep getting rejected should refrain from sending similar pull requests in the future) or from self-identified mistakes (e.g., a DevBot who decides, through their own facilities, that a specific solution does not work should avoid producing similar solutions in the future). (2) Adaptation due to explicit feedback. An ideal DevBot should also be able to process and incorporate explicit feedback, provided for instance through a natural language interface.

Technical competence: any DevBot needs to be able to produce solutions to a certain (even if potentially narrow) class of tasks. However, an ideal DevBot needs to also be sufficiently task-aware to understand the broader implications of the solutions it produces. For instance, a refactoring bot should be aware that it needs to verify that a refactoring also needs to make the code compile and the tests pass, as well as respect other quality criteria essential to the project. An ideal DevBot should know more than one way to attack a problem, so as to allow it to experiment with different solution approaches if the first attempt fails.

Social competence: in addition to technical competency, an ideal DevBot also needs to have (or, at least, fake) social competence. The need for social competence is also illustrated by [14], who showed experimentally that existing bot contributions to Stack Exchange get downvoted even if they are of high technical quality. Social competence for DevBots comes in three main flavors. An ideal DevBot needs to: (1) communicate over a human-like natural language interface, (2) exhibit some semblance of emotional intelligence (e.g., a bot whose task is to critique the work of humans may want to adapt how it communicates suggestions based on whose work it is challenging), and (3) observe when it can, and cannot, interrupt a human. Existing research by [15] on productivity and flow in software development indicates that a machine can in fact determine whether a developer is "in the flow", but contemporary DevBots are generally oblivious to this. Productivity awareness is particularly important to avoid the "Clippy effect"[2], i.e., a bot that is perceived to hamper productivity through unwanted interruptions.

V. CONCLUSIONS

In this paper, we have discussed and classified the current state of practice in software development bots (DevBots). We presented a simple taxonomy to classify 11 existing DevBots. We argued that contemporary DevBots lag behind their promise, and propose a working definition of ideal DevBots. With this "stretch goal" definition, we hope to trigger the academic and industrial community to think beyond the current state of DevBots, and embrace the promises of artificial intelligence to build future DevBots that act as artificial developers more than a simple development tool.

[2]https://en.wikipedia.org/wiki/Office_Assistant

REFERENCES

[1] A. N. Meyer, T. Fritz, G. C. Murphy, and T. Zimmermann, "Software developers' perceptions of productivity," in *Proceedings of the 22Nd ACM SIGSOFT International Symposium on Foundations of Software Engineering*, ser. FSE 2014. New York, NY, USA: ACM, 2014, pp. 19–29. [Online]. Available: http://doi.acm.org/10.1145/2635868.2635892

[2] 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," *Proc. ACM Hum.-Comput. Interact.*, vol. 2, no. CSCW, pp. 182:1–182:19, Nov. 2018. [Online]. Available: http://doi.acm.org/10.1145/3274451

[3] C. Lebeuf, M. Storey, and A. Zagalsky, "Software bots," *IEEE Software*, vol. 35, no. 1, pp. 18–23, January 2018.

[4] C. R. Lebeuf, "A taxonomy of software bots: Towards a deeper understanding of software bot characteristics," Master's thesis, University of Victoria, 8 2018.

[5] P. Maes, "Intelligent software," in *Proceedings of the 2Nd International Conference on Intelligent User Interfaces*, ser. IUI '97. New York, NY, USA: ACM, 1997, pp. 41–43. [Online]. Available: http://doi.acm.org/10.1145/238218.238283

[6] H. S. Nwana, "Software agents: an overview," *The Knowledge Engineering Review*, vol. 11, no. 3, p. 205244, 1996.

[7] P. Maes, "Agents that reduce work and information overload," in *Readings in HumanComputer Interaction*, ser. Interactive Technologies, R. M. BAECKER, J. GRUDIN, W. A. BUXTON, and S. GREENBERG, Eds. Morgan Kaufmann, 1995, pp. 811 – 821. [Online]. Available: http://www.sciencedirect.com/science/article/pii/B9780080515748500844

[8] S. Urli, Z. Yu, L. Seinturier, and M. Monperrus, "How to design a program repair bot? insights from the repairnator project," in *2018 IEEE/ACM 40th International Conference on Software Engineering: Software Engineering in Practice Track (ICSE-SEIP)*, May 2017, pp. 95–104.

[9] R. Scandariato, J. Horkhoff, and R. Feldt, "Generative secure design, defined," in *2018 IEEE/ACM 40th International Conference on Software Engineering: New Ideas and Emerging Technologies Results (ICSE-NIER)*, May 2018, pp. 1–4.

[10] R. Feldt, F. G. de Oliveira Neto, and R. Torkar, "Ways of applying artificial intelligence in software engineering," in *Proceedings of the 6th International Workshop on Realizing Artificial Intelligence Synergies in Software Engineering*, ser. RAISE '18. New York, NY, USA: ACM, 2018, pp. 35–41. [Online]. Available: http://doi.acm.org/10.1145/3194104.3194109

[11] 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*, ser. FSE 2016. New York, NY, USA: ACM, 2016, pp. 928–931. [Online]. Available: http://doi.acm.org/10.1145/2950290.2983989

[12] B. H. Kwasnik, "The role of classification in knowledge representation and discovery," *Library Trends*, vol. 48, 1999.

[13] M. Usman, R. Britto, J. Brstler, 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. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0950584917300472

[14] A. Murgia, D. Janssens, S. Demeyer, and B. Vasilescu, "Among the machines: Human-bot interaction on social q&a websites," in *Proceedings of the 2016 CHI Conference Extended Abstracts on Human Factors in Computing Systems*, ser. CHI EA '16. New York, NY, USA: ACM, 2016, pp. 1272–1279. [Online]. Available: http://doi.acm.org/10.1145/2851581.2892311

[15] M. Züger, C. Corley, A. N. Meyer, B. Li, T. Fritz, D. Shepherd, V. Augustine, P. Francis, N. Kraft, and W. Snipes, "Reducing interruptions at work: A large-scale field study of flowlight," in *Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems*, ser. CHI '17. New York, NY, USA: ACM, 2017, pp. 61–72. [Online]. Available: http://doi.acm.org/10.1145/3025453.3025662