Appendix A

Interviews with Software Practitioners

In order to assess the usefulness of the technical lag concept, we carried out interviews with software practitioners during the gathering of open source software developers in *FOSDEM 2019*[1]. Each interview was grouped into 4 parts:

- **Profile:** To gather demographic information about the interviewee such as his or her background and experience.
- **Software Characteristics:** To collect information about the software projects that the interviewee is involved in.
- **Updating Process:** To obtain information about the current updating process and policy that is followed in the software project in which the interviewee is involved.
- **Technical lag:** After explaining the concept of technical lag to the interviewee, we ask him/her about its expected usefulness.

This appendix includes all answers to all questions that have been asked during the interviews to all participants.

Profile:

Question: What is your role in the software project(s) you are involved in?
P1: I am a development coach. I am not involved in coding but I am working as a mentor to developers.
P2: I am the only developer.
P3: I am the leader of the team.

[1] https://fosdem.org/2019/

P4: I am a developer.
P5: My role is a developer advocate and I come from an operations background. For my company, I am helping people to use and operate our technology.

Question: Which professional experience do you have?
P1: As a developer I have 5 years experience, and as a coach I have 9 months. I also did some mentoring and computer science and machine learning training.
P2: I have 5 years of experience in programming, however, as a professional experience I have only 9 months, that's because I have just graduated my masters.
P3: I started working in computer engineering in 2003, that's 15 years now. I also worked in a research group for some years.
P4: I have 3 to 4 years of development, plus 4 years of research.
P5: More than 20 years.

Software Characteristics:

Question: Which kind of project is your software project? e.g, library, application, open source, industrial, etc.
P1: I am working on a pretty big project. The functionalities that the tool provides are mainly services. The tool is not open source.
P2: I am working on a tool that is a developer activity tracker. It is an internal tool that is not open sourced yet.
P3: The project is open source and it is mainly a library.
P4: The tool is more like an open source platform, it can be used as an application or as a library.
P5: It is an open source application.

Question: To which domain is your deployment dedicated?
P1: We mainly do infrastructure work.
P2: The tool is for my team and it is purely for management. Our team members contribute to open software projects and my tool tracks their contributions. So it is dedicated for managers, team leaders and also developers.
P3: The tool is for IT, data extraction, data analysis, etc.
P4: The tool is for IT practitioners, for data processing, data management.
P5: It is used by software IT companies. It is for deploying containers.

Question: What is the size of your software project?
P1: We are five to twelve people working full time on the project.
P2: I do not think it is a big project. In terms of lines of codes, we have 2000 lines of code.
P3: The project is pretty big, we have separate modules that can be used individually, but they can be used as one project as well. In general, I think we have about 6000 lines

of code.
P4: It is medium size, we are 4 people working on it (only development), but one or two people can hold all the knowledge.
P5: The project is pretty big and it is a popular one in the world.

Question: Which technology do you use for the development/deployment of the software project?
P1: We use Python with many other technologies, e.g. Docker, Ansible, etc
P2: The tool is developed in Python and for the front-end I use some of JavaScript, mostly Angular.
P3: We use Python for the programming language, we use other technologies like JavaScript, Flask, Docker, ElasticSearch and many others.
P4: We use Python, MySQL, JavaScript, and frameworks like Django, Angular, etc.
P5: For the main application, we use Go, and for another software that we use to deploy our software, we use a mix of Go and Ruby.

Question: How critical is the dependency management in your software?
P1: We use many libraries. So I think that the dependency management is pretty critical in our case.
P2: Some of the dependencies are not that important, but some of them are contributing to the main functionalities of the tool.
P3: It is very important, for example Kibana and ElasticSearch are always changing and our tool is based on them, so we have to keep up with their updating. Sometime it is hard for us to update from a major version to an other one, because new major versions have breaking changes.
P4: It depends on the type of dependencies. Our tool is built on top of another tool, so we try to keep an eye on its new releases. However, we also have dependencies that are not core ones, so we do not update these dependencies.
P5: The dependency management is the hottest spot in our project.

Updating Process:

Question: When and how do you update the software you depend on?
P1: It is a trade-off, sometimes we do the updating manually, sometimes we do it automatically. Like, if you do it manually, it may take some time to do it, but if you do it automatically, you may have a new update that has breaking changes. Also, doing it automatic means that you will always be in the bleeding edge, which means if a break happens you will need to deal with it every six months maybe, the effort will be distributed and will not be noticed by the business. While if you do it manually, maybe you will need to say that you need a six months of dependency refactoring. Anyway, it is always a trade-off.

P2: Actually, I did not update them in a while. But that is because I did not face any issues yet.
P3: Our dependencies are pinned to one specific version. In the case where there is a known vulnerability, we update the dependencies manually because we have to test them. For the main technologies that we use, we try to keep up with their updating process. We also have a case where we use an outdated but stable version of a dependency. The dependency is not so important for us, it only has one task. That's why we do not care about updating it.
P4: We always do the updating manually. When we know that there is a new release of one of the core dependencies that we use, we schedule a date of when we can do the migration/updating. Still, sometimes we can't keep up with the updating pace of the tools that we depend on. We are not a big team to be able to keep an eye and also keep up with the updating process of our dependencies.
P5: The tool that manages our dependencies looks for new updates and asks us if we update, so just one click and then everything is updated automatically. So, I can say it is automatic with a bit of human interaction. For some dependencies, even if we get a notification of a new available dependency release, we do not update.

Question: Why do you update?
P1: For security reasons, performance, novelty, just not to have all tools which are not used by anyone and not maintained anymore. In fact, I think that we should use a dependency that have many people working on it, and that has people enjoying working on it.
P2: I would update if there is some important vulnerability fix or some very good new feature.
P3: We mainly update for features.
P4: We do it for features.
P5: Mostly for security purposes, just to make sure that there are no open vulnerabilities.

Question: Do you keep tracking if your software dependency is outdated?
P1: I keep tracking the software for security updates. I sometime track features but not as important as security updates.
P2: No, I don't really do it.
P3: We are in contact with the developers of the core dependencies, so we receive notifications from time to time about the new releases.
P4: We are always having an eye on the tools that we use. But we don't have some automatic tracking.
P5: Yes, we have our tool that tracks the dependencies we use.

Question: Is there any guideline that tells you, when and how you have to update?
P1: I think it is quite psychological. I think the good thing for me is to see benefits of it.

P2: No there is no such guideline.
P3: No we don't, but yes, we always have that feeling, because the developers of the tools that we depend on are always adding many things and we are always asking these questions of should we update? and what are we going to gain if we update? and how much time we will need for that? We also have many transitive dependencies that need to be updated also if we update other tools, so yes we are always looking for metrics that tell us why we should update.
P4: No, we don't actually have any measurement that tells us how outdated we are.
P5: I don't think that we have any automated guideline for that. Even for our costumers we do not ask them to update our tool, because they are just two to three versions behind, so that's fine. So I can say that our guideline is that "update when you can". Obviously, if someone is lagging behind with one year then "good luck".

Question: Do you contact or consult with other people before updating?
P1: When you do something, you should always talk about it.
P2: I am the only developer.
P3: Yes, we contact each other to ask about each others opinions.
P4: Yes, nobody does the updating alone.
P5: I just do it. Because I have my own development environment.

Question: How much time does the updating take? Person time and real time (Human effort vs Computer effort)?
P1: I remember downgrading a dependency for availability purposes and because the version that we downgraded to was a good version. It took about one to two hours.
P2: Until now it is not important, it is just a matter of running the pip manager.
P3: Sometimes it takes just some minutes, and in some cases it really takes weeks because of the refactoring work behind.
P4: For the core tools, it takes so much time from starting to finishing the updating, sometimes it takes even months. The time needed is more about machine effort, to update we have to migrate some data and that's the part where it takes so much time.
P5: I would say, usually about 10 minutes of my time and a couple of hours for the services to download all requirements.

Question: Which are the most important characteristics of more recent versions for you to decide to which version to upgrade? Example: stability, absence of bugs, fixes of security vulnerabilities, backward compatibility, new functionality, performance, etc
P1: Security, that's for sure. Features, maybe performance but that's less relevant for me. If a new version has only new features but I will not need them, then I do not think I will update to it.
P2: I don't like to spend a lot of time in the front-end, so I think if there are some new features that will make my work easier, I would use it. Security is not an issue for me

since the tool is just internal, for now.
P3: Security is always important, especially when we work with critical and personal information. We have one case when we did a major update that took us a lot of time for security reasons. However, new features are important.
P4: The most important characteristics would be new functionalities. In some cases, the most stable one (absence of bugs) is also desirable.
P5: I think security is important. And also just getting access to the new features. For me the latest is more important than security, since it will have fixes for the open vulnerabilities. However, I think sometimes new vulnerabilities are emerged within the latest version.

Technical lag:

Before starting the questions of this part, we explained the technical lag concept and we gave examples similar to the ones presented in chapter 4.

Question: There are many different cost and benefit measurements that could be used when computing the technical lag, for example: new functionalities, resolved bugs, fixed vulnerabilities, lines of code, commits, version numbers, time, conflicts, etc. Which ones do you think are the most interesting to consider?
P1: I would consider the *changelog* between versions, what is new and what changed between the versions.
P2: I think intuitively, one should look for how many versions is the dependency missing, but I think it is a mix between all units, features, breaking changes and fixed vulnerabilities. Number of commits is also important because it shows how much effort people are putting into the project.
P3: Usually vulnerabilities, but of course bugs. Also, it depends on the tools, sometimes some bugs are not critical so you can live with them.
P4: In general, I think security is more important. So the number of vulnerabilities is more important.
P5: I think there are two important metrics, features and vulnerabilities. They can sum up everything, they are almost like a reverse *changelog*. But also I think what would be interesting is to know how many people had problems in order to have the ideal update. So it is kind of effort vs benefits.

Question: In the case of the software projects you are involved in: What do you think is the ideal dependency version that you would like to use?
P1: The ideal version that we would use is the most secure, however, we only use the bleeding edge version (i.e. the latest version), because it is easy.
P2: I think the one that is working and stable is pretty fine for me. I just want the

development to be as easy as possible for me.
P3: Well, since I am the leader, what I told you is what we do. However, in general I think it is a balance between the most secure and the latest versions.
P4: In our case, we prefer the latest versions (features).
P5: I think the latest major stable version would be good, but if you have appetite for security then do patches as well.

Question: What do you think of the concept of technical lag?
P1: I like it. I would consider cost. And I think it also motivates me to update or not. For example, knowing the most secure version and the features that I would miss if I use it is interesting. But also I would like to talk about the cost and benefits part, if you just go to the minor version, maybe you will get only minor stuff, but the updating will be easy. While if you go to a major version maybe you will have more stuff but the updating will require more cost, and more work.
P2: It is interesting. I think it would be more interesting to give different metrics when reporting the lag.
P3: I think it is good to know this kind of balance.
P4: I think it is helpful. If we know the ideal version that we can target, we can consider the concept of technical lag. Depending on the projects, it can be a motivation to update.
P5: I think it is great. It is definitely something we are missing. For the larger enterprise customers, the more we can say to them "this is why you should keep being up to date versus 10 years old version", the more is better.