An Analysis of Different Git Strategies

UNDERSØKELSESMETODER / VÅR 2019

Kandidatnummer: 101030, 101007

1.0 Introduction

Git is an industry standard versioning control system–or source control management software–that is distributed rather than centralized. In its core, git allows you to keep track of changes to text files, most often source code–but it can also be used to track any raw text files. These changes are recorded in a *commit,* which is a snapshot of files tracked in a working index along with a message from the author, that is stored in a *repository*. The recorded changes can be compared, reverted and checked out to the working area.

The author always works on a local repository, but changes can be pushed to a shared central repository that can be used by multiple authors to collaborate. Managing multiple authors, often thousands, working on a single repository can be challenging when all changes are stored linearly. This is solved by git storing all commits as a graph, where a commit has either a root, a parent, or two parents. When two commits are stored that make changes to the same text line, where a common ancestor (parent commit) cannot be determined, there will be a conflict over which changes should be made and the authors have to agree on a common resolution.

Previously described is what is historically referred to as working on the trunk. However, git can create separate timelines of new commits separately from the trunk, which are called branches.

On its own, the versioning software has no opinion on how you use it. There are a lot of different strategies in use, with varying efficiency and purposes.
This paper provides an analysis of many of the more popular Git workflows and branching strategies in use within the industry.

1.1 Hypothesis

Due to these differing strategies being in use, it can be difficult to know which ones work, and for what problem that specific strategy is trying to solve. The best strategy for a situation can also be affected by both the project and team size.

In this paper we aim to identify industry usage of different git branching strategies.

2.0 Data

2.1 Methodology

Survey

To reach a broader range of respondents within a narrow field of study, a survey was a great fit with us. A questionnaire is a commonly used tool for doing surveys (Oates *2006, 281*), where we can compare practises across multiple companies. To gather data on different philosophies and how they might be adopted and implemented in various environments, we need to gather data from a large number of people in the industry.

Other optional data sources include interviews, which could be beneficial due to the ability for clarifying uncertainties, however with limited time and required saturation of responses this is not possible in the time frame the project is set within.

Literature / article review

It is important to gather data on what has been published earlier, both in academic context and specific use-cases from established companies.

Concept 1	Concept 2	Concept 3	Concept 4
Strategy	Git	Collaboration	Considerations
Approach		Teamwork	Review
Blueprint		Combination	Plan
Method			

Figure 2.1.1 According to Oates (*2006*), splitting out key search terms first is a good start to a literature review. Here we have analyzed key concepts and alternative search terms for further research.

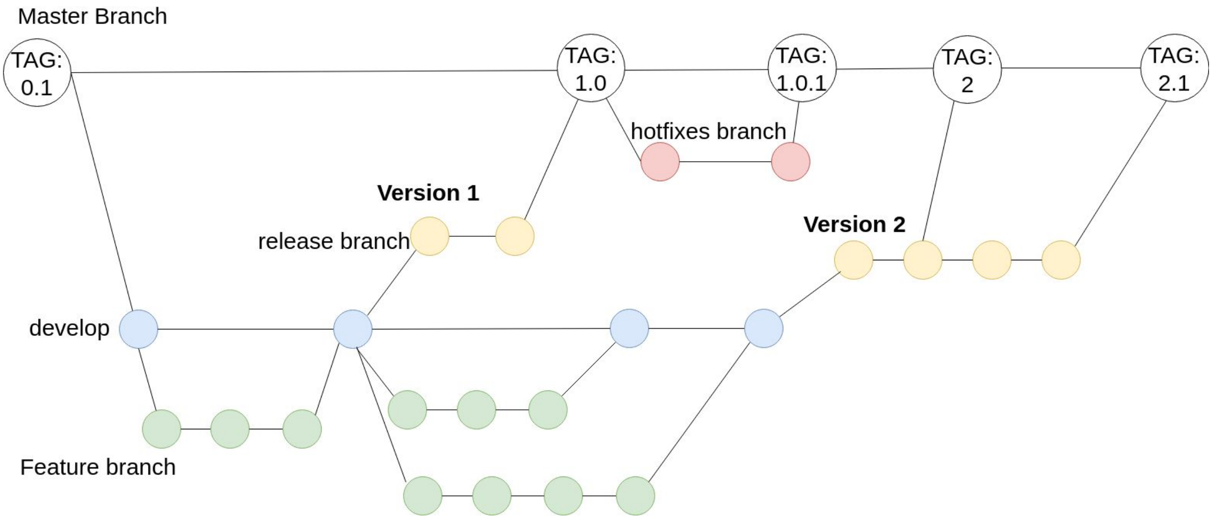
Many larger corporations such as Microsoft, Google, Facebook, Netflix and GitHub are writing about their practises and findings using various management techniques. Due to the influence of larger companies such as these, with thousands of contributors of code to one (or many) repository, they can be used as valuable comparators to complement data gathered from local technology industry.

2.2 Literature review data

Git Flow

One of the older and more well known git branching models “Git Flow”, published and popularized by Vincent Driessen (2010), is a well renowned git workflow that is used as a foundation for many more modern workflows.

Git Flow uses a master branch (previously referred to as trunk) where only the latest versions lives. Other branches are only merged into this branch when the product (multiple cherry-picked features and fixes) is ready for a new stable version to be released to users.



Required branches:

- Releases (often named according to Semantic Versioning, (Preston-Werner, 2013))
- Hotfixes
- Develop
- Features

The develop branch receives all features first, and is pulled from when creating feature branches. When it's ready for a release, it is merged into a release branch. At this stage the code in the release branch is being tested and bug fixed only until it is ready for a release. After testing, it has to be approved and merged into the master branch and deployed to the end user. Should an issue arise, then the issue is pulled to a hotfixes branch where the issue is resolved and merged back into the master.

This strategy can be good for larger teams, considering you have a common develop branch and release branches that work as a safety net before reaching master where the product reach the consumer.

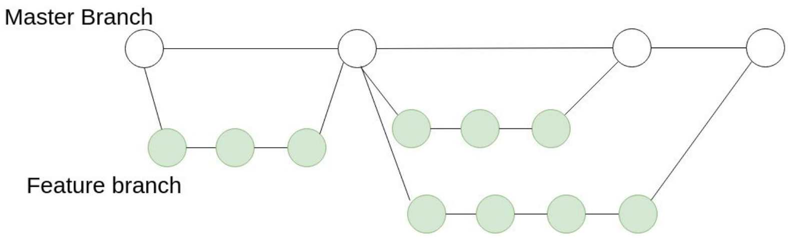
Case Tensorflow

TensorFlow is large scale project working on machine learning, https://github.com/tensorflow/tensorflow/branches/active. We can identify that they use properties from git flow by the active release branches, as well as properties from GitHub Flow. We can observe this from the active release branches, while at the same time a branch being actively pulled from and features merged into. However where this differs from pure Git Flow is in the use of the master branch instead of a separate develop branch.

Git Flow does not specify when it is beneficial to use this workflow over the others. One can argue that this is because this one of the older flows and therefor there was not many other standardized flows around specifically for git.

GitHub Flow

GitHub Flow is a significantly simplified derivative of Git Flow, as GitHub Flow operates with a master branch instead of a dedicated development branch (GitHub, 2017). When a feature branch is complete, it should be pull-requested into the master where the team discuss the proposed changes and make changes accordingly before merging. The feature branch is deleted after merging.



Common branches:

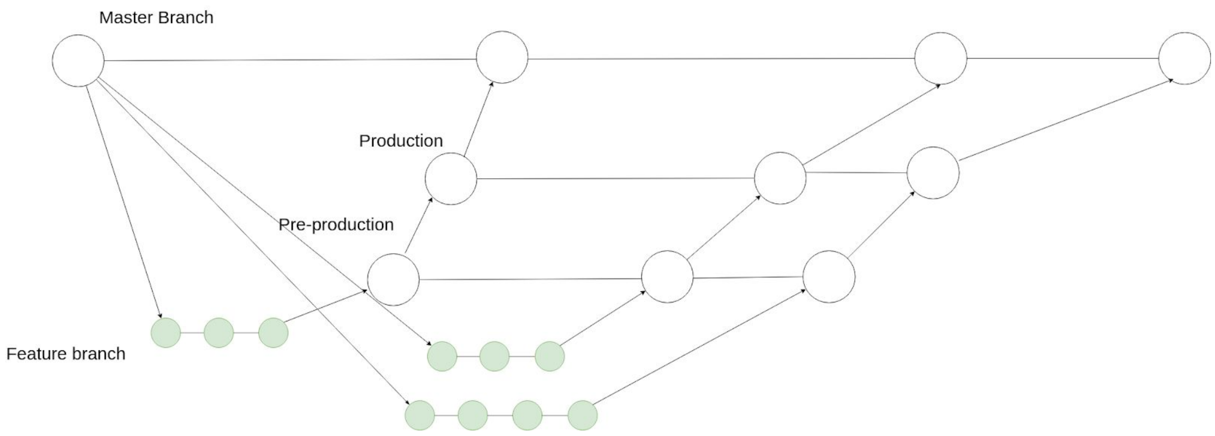
- Features

Case Visual Studio Code

The Visual Studio Code open source project is using GitHub Flow, https://github.com/Microsoft/vscode/wiki/Coding-Guidelines. We can identify this by a common master branch, where *forks* and feature branches are rebased in, and feature branches are short-lived. In this example it’s even more extreme as a rebase is performed, rather than a merge.

GitLab Flow

GitLab Flow is another strategy derived version of Git Flow, but focus on changes to make CI/CD easier (GitLab, 2014). GitLab Flow uses environment branches that are next to the master branch. They all live in separate environments and can be monitored for bugs individually.



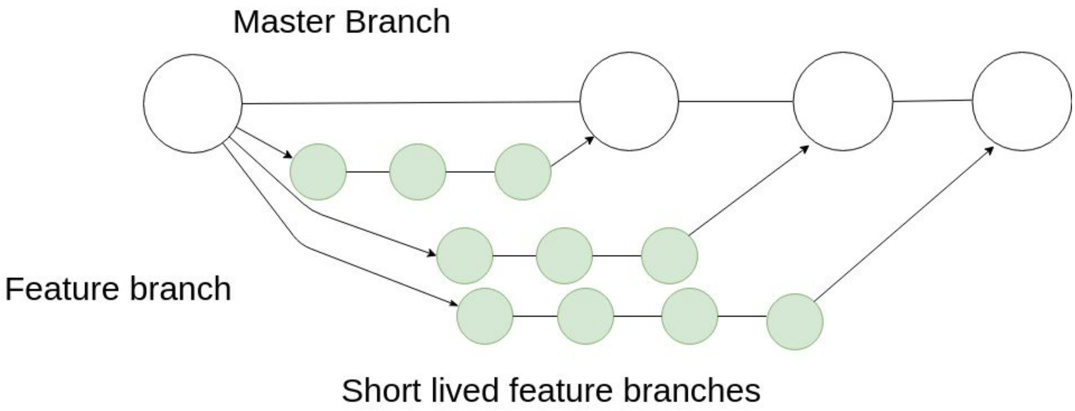
Common branches:

- Environment (example; production, pre-production)
- Releases (often named according to Semantic Versioning, (Preston-Werner, 2013))
- Hotfixes (cherry-picked into release branches)
- Features

A feature is merge-requested into master, and a CI-build is performed to verify that it integrates. After it is approved then it is cherry-picked into a release branch (if it exists), or directly to an environment branch. Most commonly to either a production environment or a pre-production, then followed by production if it is approved in pre-production. It is common practise with this strategy to have the environment branches reflect the live environment [testing | production] available to the end-user.

Trunk Based Development

Trunk Based Development (TBD) is made with CI in mind, where all changes are stored in a main trunk, referred to earlier as the master branch (Hammant, 2017). All changes are reviewed then merged into the trunk only after passing CI-builds. This creates a promise that the codebase always is releasable on demand and simplifies CD. Instead of long running feature and release branches which are in Git Flow and derivatives, TBD uses short lived feature branches. With a smaller team, changes committed directly to the trunk is allowed.



Common branches:

- Features

According to Google (Potvin, 2015), who at the time had 25 000 collaborators and 45 000 commits per day to a single repository following TBD, this strategy is also well suited for very large projects. Facebook also uses use this approach to manage many million lines of code (Rossi, 2011).

2.3 Ethics

Reflecting back on the techniques on how we gathered data, there are a few mentionable things done in this research that could be considered unethical. We used Google Forms, and do not fully control the metadata that Google gathers. As such the first thing we consider unethical is not considering how the data that our subjects sends to us are stored at Google.
Another ethical quarrel we should have considered is open-ended questions should have an option for opting out or skipping. This could be considered unethical and should have been reasonably easy to implement in our questionnaire prior to releasing.

3.0 Analysis

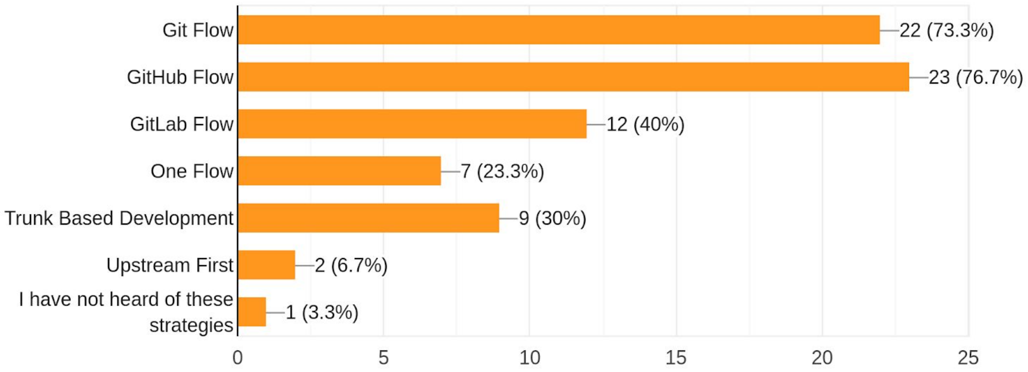
The data we have gathered has been analyzed was both qualitative and quantitative. Our observations from usage at large open source repositories gave us qualitative data, where we saw their use of branching strategies. Using a quantitative analysis over this qualitative data using the method proposed by Oates (*2006, 327*), looking for specific features we can identify properties from most of the strategies discussed above. Our questionnaire gathered both quantitative and qualitative, where we can compare answers from open-ended questions with other other answers or bisect based on other quantitative answers such as multiple choice or ‘pick one’ questions.

For processing the quantitative data from the questionnaires we used the data analysis tool provided by within Google Forms. This helped assistance analyzing and visualizing the data. Google is being used daily by a large amount of users to create surveys, and utilizes Machine Learning to fit the data in representational models from the question and answers (Google, 2017). We use this as well as filtering the data spreadsheet for qualitative comparisons in addition to automatic analysis.

The case studies were performed by reviewing the branching and commit history of that repository. By looking over how these repositories accept contributions we can interpret what branching strategy they were using, or at least properties from different strategies. By doing this we could quantify usage, in public repositories, of different strategies.

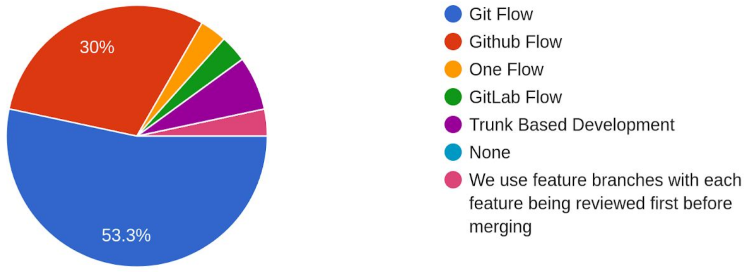
Which of the following git workflows and branching strategies are you familiar with?

30 responses



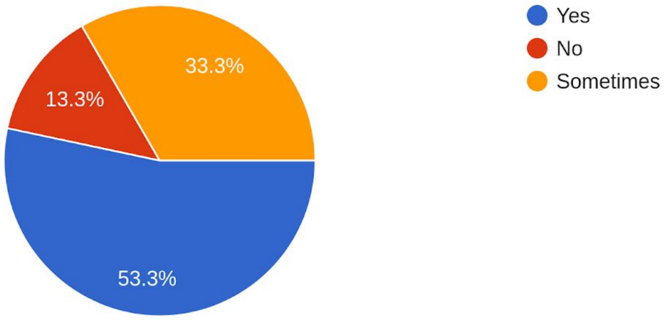
What strategy do you most commonly use?

30 responses



Do you consider CI/CD when choosing a branching strategy?

30 responses



4.0 Discussion

4.1 Reflections on method choices

Survey

As stated in earlier, we needed a considerable (50-100 >) amount of data to be able to discern patterns across multiple companies. Both from senior developer, team leaders and project managers, to regular developers and team members. We sent out emails to multiple HR and open mail boxes for larger tech firms in the area, such as Sopra Steria, Bekk, Telenor, Netcompany, Intility, Microsoft, Accenture, Visma, as well as direct contacts within multiple other companies. But it proved more difficult than anticipated to receive what we feel would be an adequate amount of data, and we ended the survey with 30 responses.

Interviews could be performed on a fewer set of subjects to have a lower requirement on data sources as we could potentially gain more insight in the company (from each individual), however interviews require a lot of time from both us and the interviewees. With a limited time to deliver the project, and from the subjects that by nature are usually very busy during normal operating hours, this made it improbable that we could finish in a reasonable time.

After further analysis of case studies, it became clear that open source could influence answers, and could provide valuable data if we included questions about project conditions such as if it was closed or open-source, inhouse versus community projects.

Literature review

The bulk of our baseline data has come in the form of literature research where we analyze what strategies are commonly used online. With sources from well known contributors to large companies that set or challenge industry standards. Companies such as Facebook with thousands of developers and talent, that is maintaining the very popular package react, which at the time of writing has 4.9m weekly downloads for software packages. Facebook, Google, Netflix and Microsoft are among the largest companies within the software industry, and produce a lot of open source and public services. As such, they are good candidates for study on how they maintain their code bases. Public git repositories such as GitHub and GitLab are also good resources, as they host an array of different software and publish multiple articles on practises in use as well as guidelines.

However, despite this, we do not have access to some of the proprietary or closed code repositories internal to these companies. Because of this, the data we have acquired could be argued to be skewed towards open-source compared to internal closed-source since we only had access to open source projects for our case studies.

4.2 Conclusion

From our questionnaire, we can deduce that most people in the industry either uses Git Flow, GitHub Flow or GitLab Flow, and from our case studies we can also confirm that many big open source projects uses either GitHub flow, Git Flow or some custom flow where the project managers has chosen to take parts they like from Git Flow and GitHub Flow to maximize their CD/CI capabilities.

Even though our literature review concludes that GitLab Flow and TBD are the most common strategies for CI, our data show that Git Flow and GitHub Flow are the most used in the vicinity of Oslo. However large companies such as Google, Facebook and Microsoft use TBD for in house development, and strategies such as GitHub flow are mainly used for community / open-source projects maintained by said companies. This is reflected in the responses on the questionnaire where only a third are familiar with TBD.

TBD appears to be widely used in Microsoft (Cooper, 2018), Google (Potvin, 2015) and Facebook (Rossi, 2011), but mainly for their inhouse products. All of their open source repositories that we have studied are not using this. The reason for this might be because it is extremely hard having a large amount of collaborators around the world contributing while also follow the required iteration sprints that makes TBD great to use.

GitHub Flow appears to be used more frequently for open-source, and seems to be deliberately made for this as it allows third party collaborators to fork the master branch and pull-request changes back to original master branch, without being a first party. Pull-request can then be reviewed by maintainers with the responsibility of gatekeeping what is accepted as additional contributions to the project.

From our case studies there seems to be a pattern of using GitHub Flow or Git Flow that we can't confirm is the same when companies are working in house due to the nature of closed source. From our case studies of open source projects on GitHub, we noticed that all of the projects we researched use Git Flow, GitHub Flow or a hybrid version between the two.

As mentioned earlier; this seems to be because these flows are well suited for third party collaborators that do not need to be ‘in the loop’. The data collected from the questionnaire does not directly tell us if the participant worked on open source or in house projects, but from the target demographic we can deduce that some of these people worked on in house projects. We can assume that Git Flow and GitHub Flow is being used in house as well, as a majority of our participants answered that this was their prefered branching flow.

The majority of projects in the industry seem to be following a combination of Git Flow and GitHub flow, where a minority is familiar with and uses TBD and GitLab flow for inhouse projects.

5.0 Glossary

CI - Continuous Integration
CD - Continuous Delivery
TBD - Trunk Based Development
Cherry Picking - Picking what is included.
Merge-request and Pull-request are used interchangeably (platform specific), and are the same.

6.0 References

Cooper, Matt. 2018. *How We Use Git at Microsoft.* https://docs.microsoft.com/en-us/azure/devops/learn/devops-at-microsoft/use-git-microsoft, accessed 18. February 2019.

Driessen, Vincent. 2010. *A successful Git branching model.* https://nvie.com/posts/a-successful-git-branching-model/, accessed 9. February 2019.

GitHub. 2017. *Understanding the GitHub Flow.* https://guides.github.com/introduction/flow/, accessed 6. February 2019.

GitLab. 2014. *Introduction to GitLab Flow.* https://docs.gitlab.com/ee/workflow/gitlab_flow.html, accessed 6. February 2019.

Hammant, Paul. 2017. *Introduction.* https://trunkbaseddevelopment.com/, accessed 9. February 2019.

Google Inc. 2017, accessed 26. February 2019. *Smarter Google Forms to save you time.* https://gsuiteupdates.googleblog.com/2017/07/new-features-in-google-forms.html, accessed 26. February 2019.

Oates, Briony J. 2006. *Researching Information Systems and Computing.* SAGE Publications Ltd.

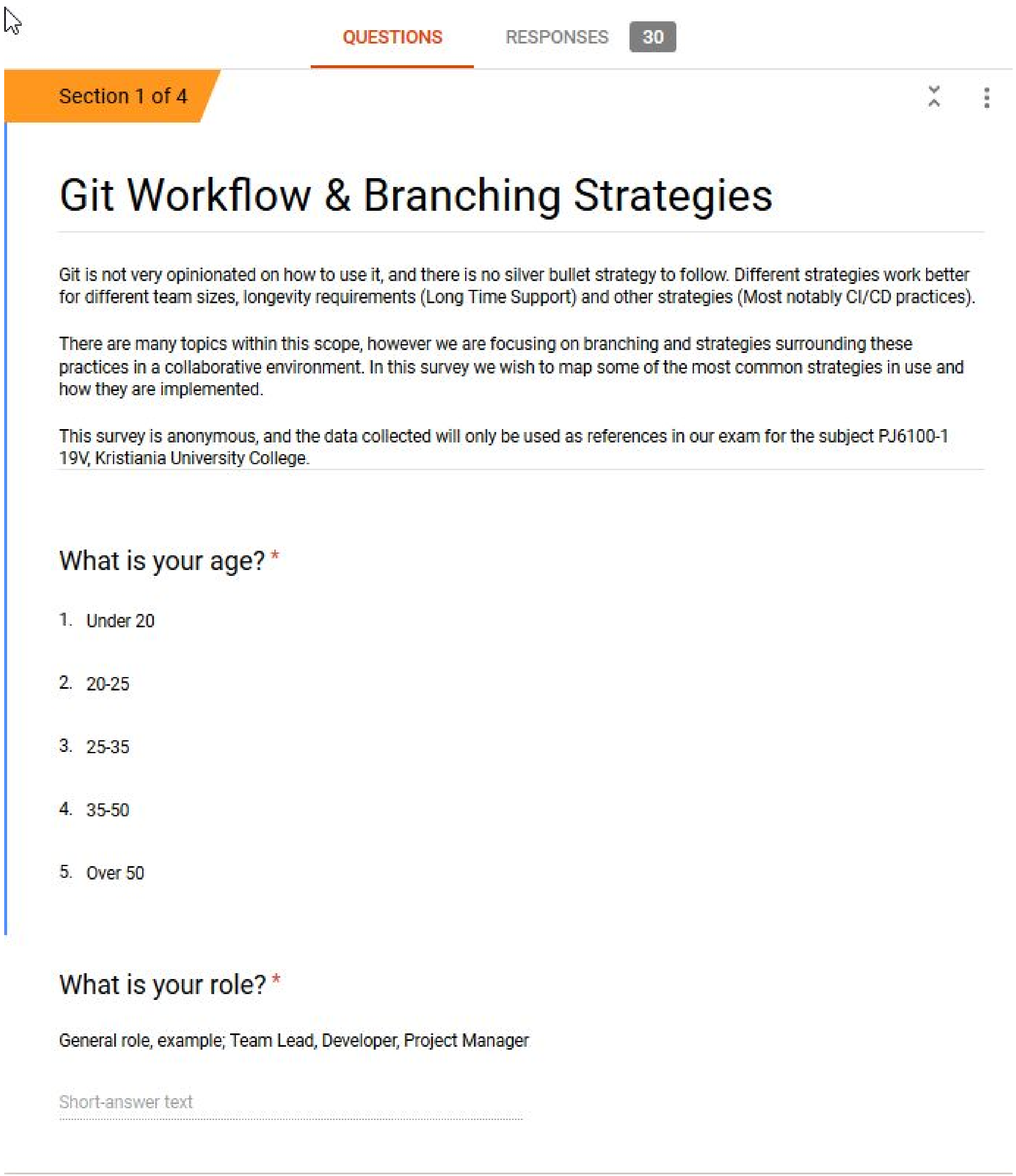
Potvin, Rachel. 2015. *The Motivation for a Monolithic Codebase.* https://www.youtube.com/watch?v=W71BTkUbdqE, accessed 22. February 2019.

Preston-Werner, Tom. 2013. *Semantic Versioning 2.0.0.* https://semver.org/, accessed 24. February 2019.

Rossi, Chris. 2011. *Pushing Millions of Lines of Code Five Days a Week.* https://www.facebook.com/Engineering/videos/10100259101684977, accessed 20. February 2019.

7.0 Appendix

Questionnaire



Git Workflow & Branching Strategies

Git is not very opinionated on how to use it, and there is no silver bullet strategy to follow. Different strategies work better for different team sizes, longevity requirements (Long Time Support) and other strategies (Most notably CI/CD practices).

There are many topics within this scope, however we are focusing on branching and strategies surrounding these practices in a collaborative environment. In this survey we wish to map some of the most common strategies in use and how they are implemented.

This survey is anonymous, and the data collected will only be used as references in our exam for the subject PJ6100-1 19V, Kristiania University College.

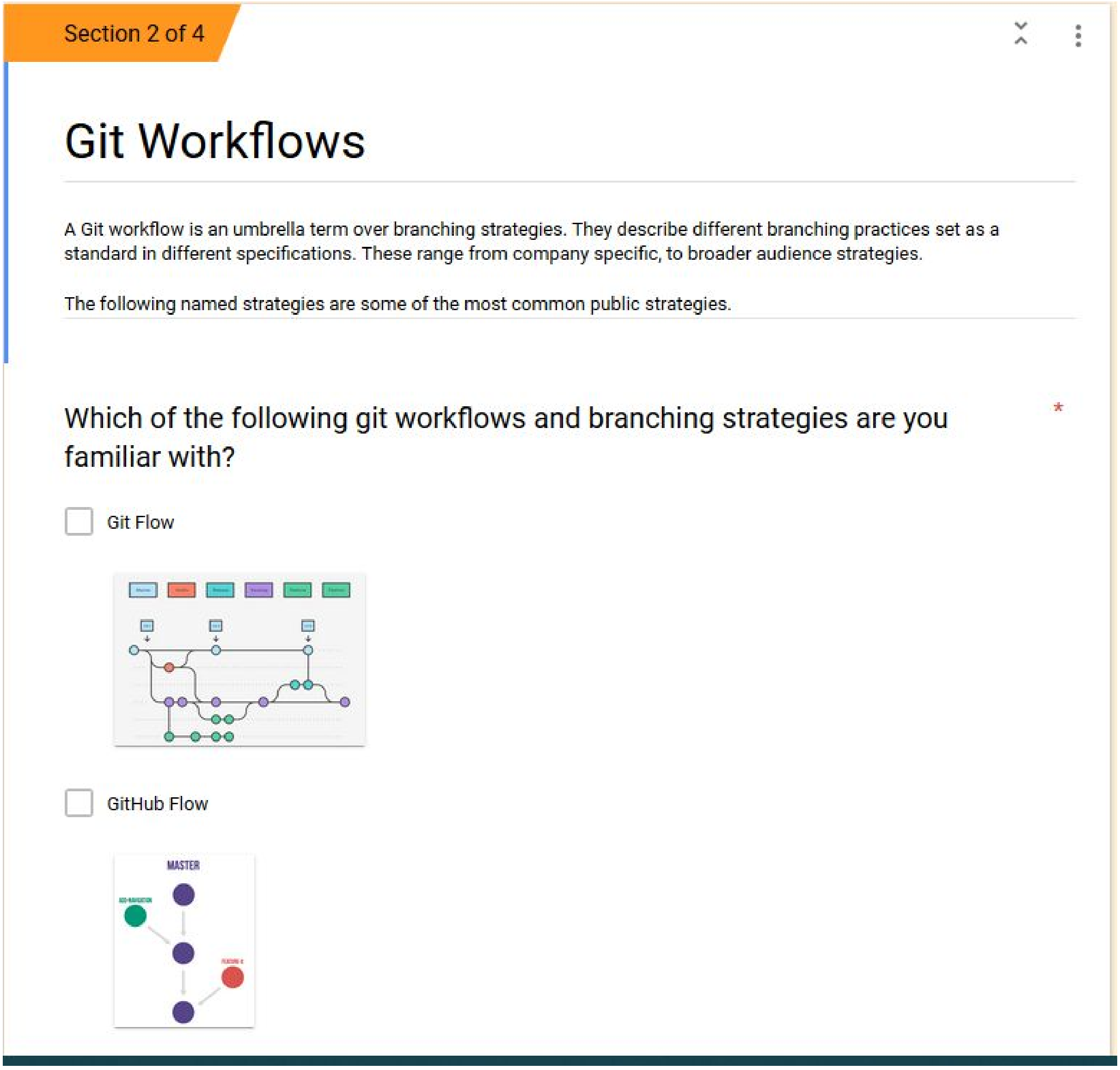
What is your age? *

1. Under 20
2. 20-25
3. 25-35
4. 35-50
5. Over 50

What is your role? *

General role, example; Team Lead, Developer, Project Manager

Short-answer text



Section 2 of 4

Git Workflows

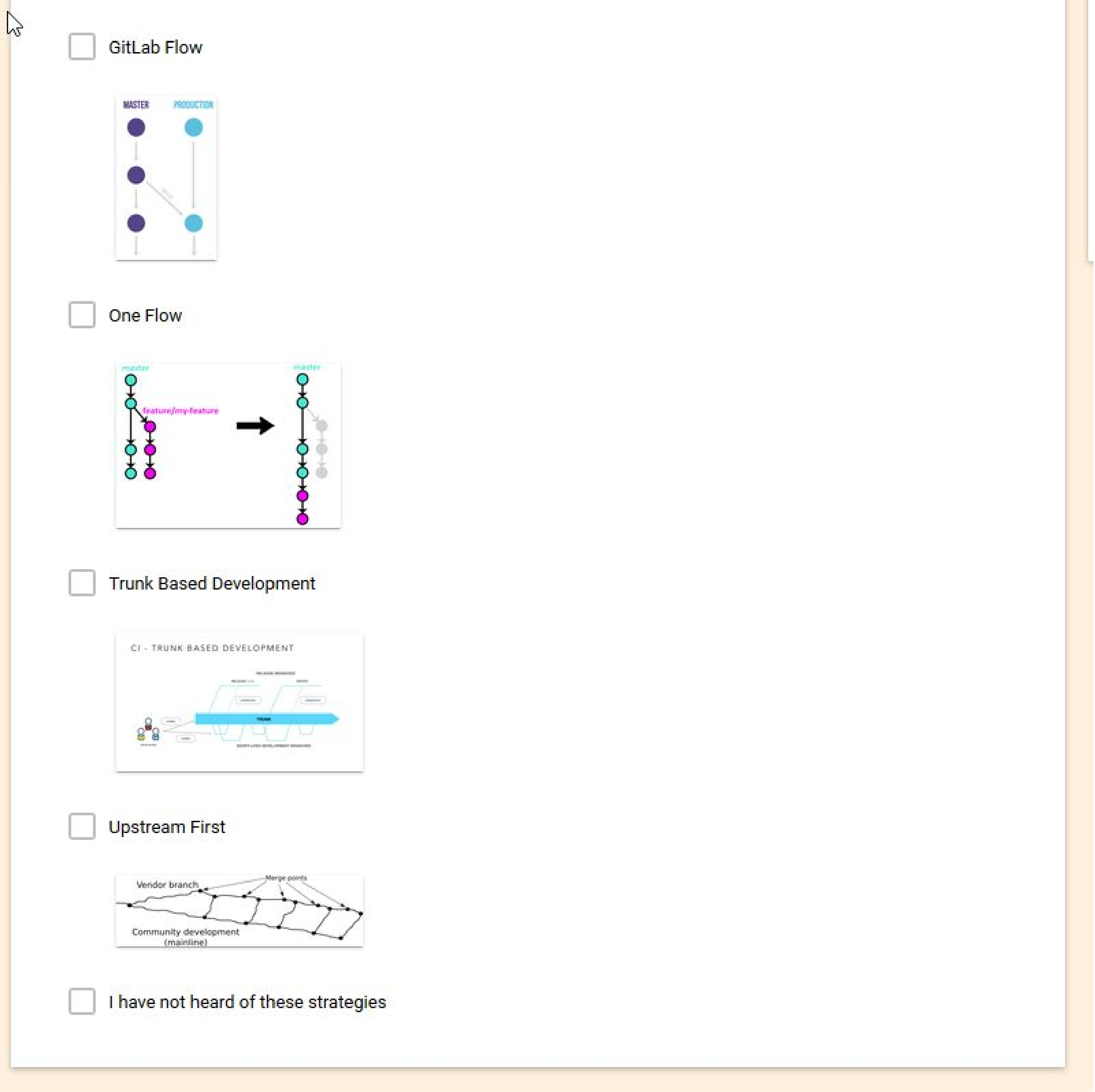
A Git workflow is an umbrella term over branching strategies. They describe different branching practices set as a standard in different specifications. These range from company specific, to broader audience strategies.

The following named strategies are some of the most common public strategies.

Which of the following git workflows and branching strategies are you familiar with? *

□ Git Flow

□ GitHub Flow



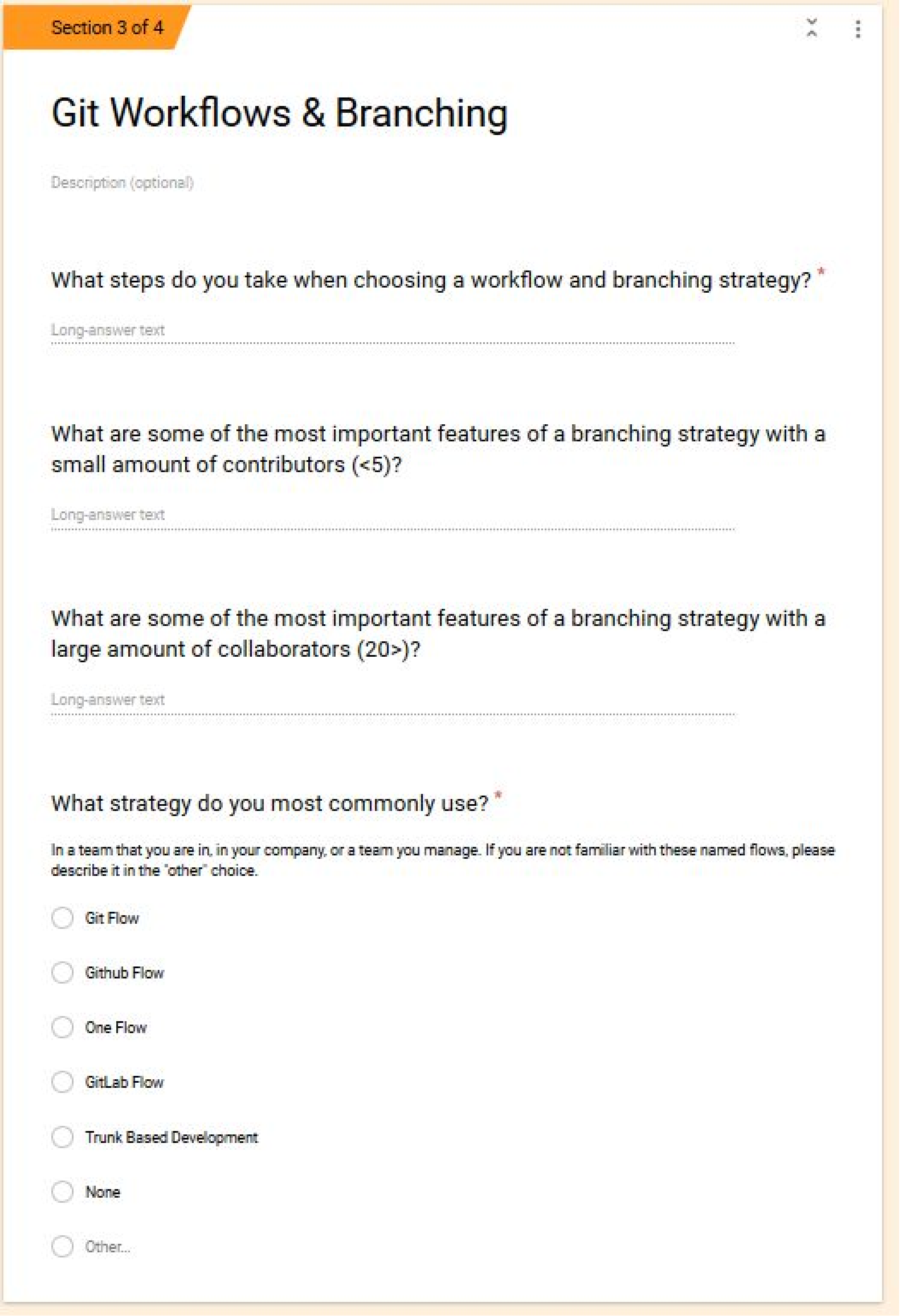
□ GitLab Flow

□ One Flow

□ Trunk Based Development

□ Upstream First

□ I have not heard of these strategies



Section 3 of 4

Git Workflows & Branching

Description (optional)

What steps do you take when choosing a workflow and branching strategy? *

Long-answer text

What are some of the most important features of a branching strategy with a small amount of contributors (<5)?

Long-answer text

What are some of the most important features of a branching strategy with a large amount of collaborators (20>)?

Long-answer text

What strategy do you most commonly use? *

In a team that you are in, in your company, or a team you manage. If you are not familiar with these named flows, please describe it in the "other" choice.

○ Git Flow

○ Github Flow

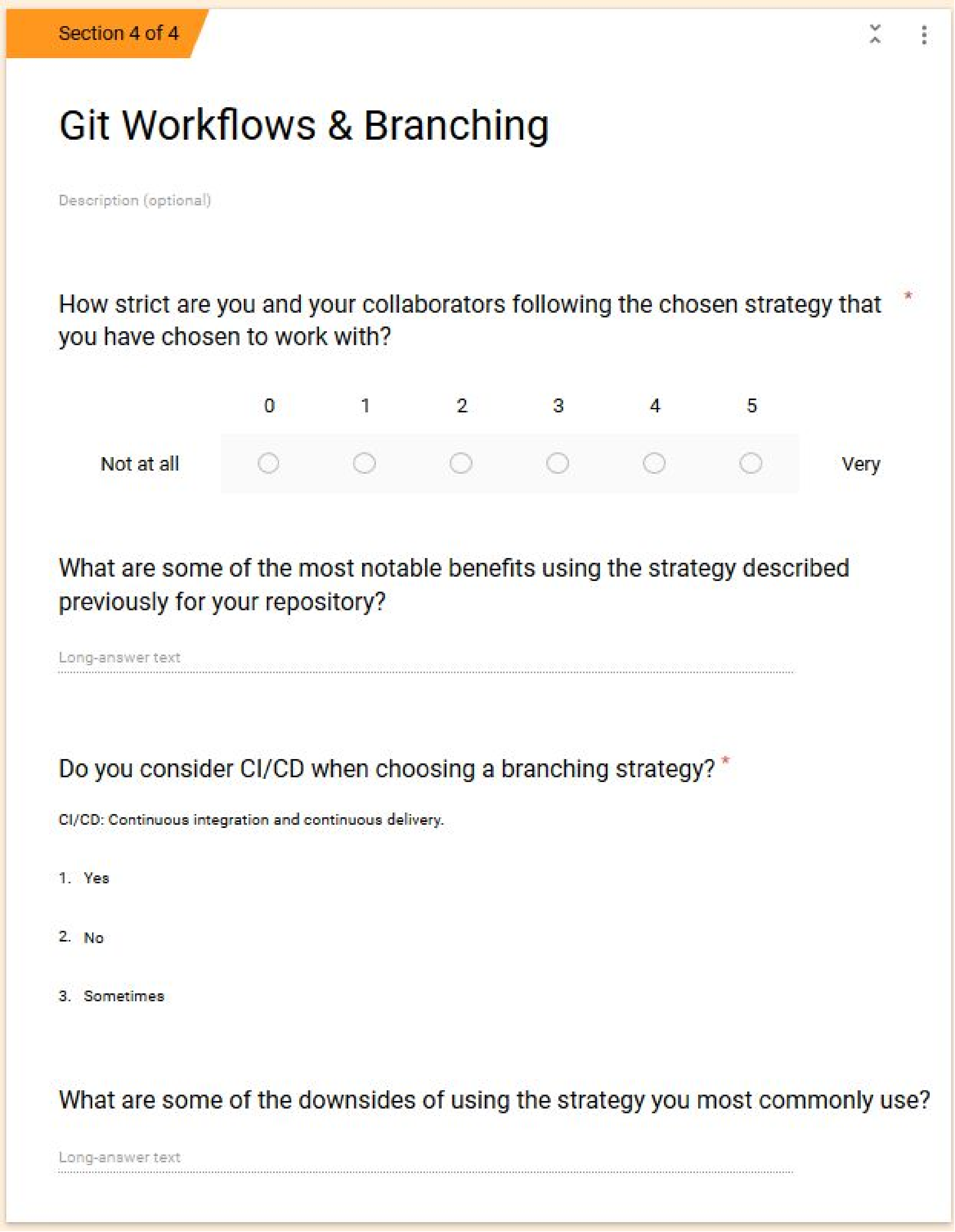
○ One Flow

○ GitLab Flow

○ Trunk Based Development

○ None

○ Other...



Section 4 of 4

Git Workflows & Branching

Description (optional)

How strict are you and your collaborators following the chosen strategy that you have chosen to work with? *

	0	1	2	3	4	5	
Not at all	○	○	○	○	○	○	Very

What are some of the most notable benefits using the strategy described previously for your repository?

Long-answer text

Do you consider CI/CD when choosing a branching strategy? *

CI/CD: Continuous integration and continuous delivery.

1. Yes
2. No
3. Sometimes

What are some of the downsides of using the strategy you most commonly use?

Long-answer text