



jamk.fi

Managing software project with GitLab

How it is done at Sparta Consulting Oy

Niko Mikael Lehtovirta

Bachelor's thesis

November 2017

Technology, communication and transport

Degree Programme in Software Engineering

Jyväskylän ammattikorkeakoulu

JAMK University of Applied Sciences

Author(s) Lehtovirta, Niko	Type of publication Bachelor's thesis	Date 15.11.2017 Language of publication: English
	Number of pages 29	Permission for web publication: x
Title of publication Managing software project with GitLab How is it done at Sparta Consulting Oy		
Degree programme Software Engineering		
Supervisor(s) Rantala, Ari		
Assigned by Sparta Consulting Oy		
Abstract <p>The objective of this thesis was to create initial base of GitLab process documentation for the Sparta Consulting Oy and to give suggestion to improve current process. Initial base means short introduction of different parts of GitLab.</p> <p>Sparta Consulting Oy has been using GitLab as version control system for the whole product development time and 1-and-half year as tool for project management. In this timespan process has been changed many times to find correct one for Sparta Consulting Oy development team. Due to these multiple changes documentation on the processes was never done. This situation wanted to be changed and that is why initial documentation on the GitLab usage was asked.</p> <p>Another wanted thing was suggestions for improvements. Current process has parts that challenging for product development and suggestion to improve these was requested. Ideas proposed in this thesis are mostly in use already, like template for merge requests. Larger changes like continuous integration are currently being implemented.</p>		
Keywords/tags (subjects) Project, Management, Development, Git, Kanban, GitLab		
Miscellaneous		

Tekijä(t) Lehtovirta, Niko	Julkaisun laji Opinnäytetyö, AMK	Päivämäärä 15.11.2017
	Sivumäärä 29	Julkaisun kieli Englanti
		Verkojulkaisulupa myönnetty: x
Työn nimi Managing software project with GitLab How is it done on Sparta Consulting Oy		
Tutkinto-ohjelma Ohjelmistotekniikan tutkinto-ohjelma		
Työn ohjaaja(t) Ari Rantala		
Toimeksiantaja(t) Sparta Consulting Oy		
<p>Tiivistelmä</p> <p>Opinnäytetyön tarkoituksena oli toteuttaa Sparta Consulting Oy:lle alustavaa pohjaa GitLab prosessidokumentaatiolle, sekä antaa kehitysehdotuksia nykyisiin prosesseihin. Alustavalla pohjalla tarkoitetaan lyhyttä esittelyä eri GitLabin osa-alueista.</p> <p>Sparta Consulting Oy:n tuotekehitys on käyttänyt koko tuotekehityksen ajan GitLabia versiohallintaan ja puolitoista vuotta projektinhallintaan. Tänä aikana prosesseja on useaan kertaan muutettu yritettäessä löytää sopivaa toimintamallia Sparta Consulting Oy kehitystiimille. Näiden muutoksien vuoksi dokumentaatiota prosesseista ja käytöstä ei koskaan saatu aikaiseksi. Tähän asiaan haluttiin muutosta ja siksi alustavaa dokumentaatiota GitLabin käytöstä toivottiin.</p> <p>Toinen haluttu asia olivat kehitysehdotukset. Nykyisessä prosessissa on kohtia, jotka hidastavat tai haittaavat tuotekehitystä ja näihin toivottiin ideoita. Näistä kehitysehdotuksista suurin osa on jo käytössä tuotekehitystiimillä, kuten pohjat merge requesteille. Isommat muutokset ovat vielä työn alla, kuten jatkuvan integraation muutokset.</p>		
Avainsanat (asiasanat) Project, Management, Development, Git, Kanban, GitLab		

Table of Contents

Table of Contents	1
Glossary	3
1 Introduction	5
1.1 Sparta Consulting Oy	5
1.2 Thesis objective	5
1.3 Version control	6
1.4 Scrum	6
1.5 Kanban	7
1.6 PlantUML	10
2 Objective	10
3 GitLab	10
3.1 Introduction	10
3.2 GitLab groups	11
3.3 Project	12
3.4 Milestones	13
3.5 Labels	14
3.6 Issue tickets	15
3.7 Issue board	17
3.8 Merge request	18
3.9 Pipeline	22

4	Proposed improvements	23
4.1	Milestones	23
4.2	Issue tickets	24
4.3	Issue board	26
4.4	Merge request.....	27
4.5	Pipeline.....	29
5	Results	30
	References.....	30

Glossary

Branch

Git repository can contain many simultaneous changes by using separated branches. Every code is always part of some branch in Git, and development team can decide which branch is the main branch to work on.

CI

Continuous integration allows running specified tests on Git repositories every time files are changed on server repository.

CD

Continuous deployment allows deploying project to a specified server or creating release build.

Docker

Tool used to build and run small virtual images where applications can be tested or run separately from the system that runs those virtual images.

Git

Distributed version control system used to store source code

GitLab CI

Continuous integration tool built to be used with GitLab. Allows running tests at any server with network connection to GitLab.

GitLab CD

Continuous deployment tools included in GitLab CI and GitLab, allow building a release file and accessing it from GitLab or deploying it to server.

Integration testing

Contains testing of user interface and whole features of program.

Merging

Changes from two branches are merged together to contain changes from both branches.

Project manager

Person who manages development team and tells them where to focus on given timeframe.

Product owner

Person whose job is to collect information on what customers want from a product and decide what features are wanted to be implemented and informs those to project manager.

Repository

Git repository contains all files that are version controlled by Git.

Unit testing

Tests that are meant for testing a single functionality in program.

1 Introduction

1.1 Sparta Consulting Oy

“Sparta was founded because we wanted to create for ourselves the kind of a company we always wanted to work for. We also wanted to do consulting in an ethical way, always delivering value to our customers. Sparta offers information management solutions that are different from the mainstream. We are proud of the way we work and operate. Sparta combines business understanding, information management and cyber security services in a unique way. We believe that business and information architecture should be a single entity, not forgetting security architecture and risk management.” (In light of better information, 2017).

Sparta was founded in 2012 and the first office was founded in Jyväskylä. The second office in Helsinki was founded in 2013. In 2015 Sparta Consulting Oy decided to invest in product development, and a product development team was founded in Jyväskylä in the same year. Currently, the office in Jyväskylä is marked as head office for Sparta Consulting Oy (Sparta Consulting Oy, 2017).

From this on point Sparta Consulting Oy is referred to as Sparta, which makes the writing and reading this thesis easier, since the full name of the company does not constantly have to be given.

1.2 Thesis objective

The objective for this thesis is to demonstrate how Sparta manages software project with GitLab and propose changes to things that are currently challenging. Background on why and how this thesis came to be is from two years back on 2015 when the author was hired to Sparta as trainee. There was limited information on how a software project should be managed at Sparta, and with the knowledge gotten from school, the author was assigned to create plan on how it would be done. This task has continued to this day.

There have been many large changes in the tools that were used; workflows have changed many times and even changes to methods were changed. For a year, Sparta

has stuck with GitLab as it has allowed to manage project without having multiple different tools. Using only GitLab has made the lives of developers and management easier as there is only one place everything can be found on.

1.3 Version control

Sparta uses Git in its version control for code. Git is a fast-distributed version control system that allows working without having a constant connection to the main server; which is important to allow developers working remotely. Another main reason for Git usage is its popularity. Almost all open source code in use uses Git for version control so it allows contributing changes back to open source.

For Git popularity, it is hard to give concrete numbers on how many projects are using Git and how many projects use something else. The only number that could be found is the number of projects hosted on GitHub. Currently there are over 71 million projects (About GitHub, 2017) hosted in GitHub. GitHub only support Git, so all those 71 million projects are Git projects.

1.4 Scrum

“Scrum’s early advocates were inspired by empirical *inspect and adapt* feedback loops to cope with complexity and risk. Scrum emphasizes decision making from real-world results rather than speculation. Time is divided into short work cadences, known as sprints, typically one week or two weeks long. The product is kept in a potentially shippable (properly integrated and tested) state at all times. At the end of each sprint, stakeholders and team members meet to see a demonstrated potentially shippable product increment and plan its next steps.

Scrum is a simple set of roles, responsibilities, and meetings that never change. By removing unnecessary unpredictability, we’re better able to cope with the necessary unpredictability of continuous discovery and learning.” (Scrum methodology)

Scrum is meant to add rules on how development should be handled in short two-week cycles so team; Scrum master and product owner can discuss how a project is going after every cycle. These cycles in Scrum are called sprints. Another idea in

Scrum is to have a short meeting every morning to with team so the Scrum master can gain information about developers situation in order to maybe assign some help for a developer that might be struggling with task.

This thesis only scrapes how Scrum works as Sparta did already use Scrum in the past. The basic understanding on how Scrum works will help when discussing history, and how and why some things were done with GitLab in the past.

1.5 Kanban

“Every business hopes to be efficient and cost-effective and to waste as few resources as possible: the essence of “lean manufacturing” That is why one finds that every lean manufacturing software solution (from multi-million-dollar, complex, integrated ERP systems to very simple and pragmatic, replenishment-based, supply-chain “kanban” setups) contends that it will eliminate waste, increase efficiency, and be easily cost-justified. In fact, without exception, there is an assertion that the ROI (return on investment) on all lean software systems is axiomatic and rapid.” (Examining Lean Manufacturing Promise)

In the late 1940s, Toyota started to study supermarkets trying to get an idea for how to apply shelf-stocking techniques to their factory floor. Customers generally get what they need from supermarket at the required time without any hassle. Supermarkets stock only what is expected to sell, and customers take what they need because the supply is assured. This led Toyota to view the customer as a port preceding the process and this preceding process as a kind of store.

Kanban aligns inventory levels with actual consumption. A signal informs suppliers to produce and deliver new material when it is being consumed. In software development, the customer tells the product owner what it wants, and those requirements are added to a board. These signals are tracked through the replenishment cycle, giving visibility to supplier, consumer and buyer. In software development, visibility is given to the development team and management so that they can report to customers what the status of their request is. (Kanban origins, 2017)

The rate of demand is used in Kanban to control the rate of production, so demand is being passed from the customer up through to customer-store processes. In 1953, Toyota applied this logic in their main plant machine shop. Kanban was originally invented by a Toyota engineer Taiichi Ohno (Taichi Ohno, 2017), who wanted to achieve at Toyota Just-in-time (JIT) manufacturing process to reduce the time from customer demand to production.

GitLab has so called issue boards that can be used to follow Kanban model. These boards are used with labels to split which column each issue ticket belongs to. Figure 1 illustrates how this looks like on GitLab. Figure 1 is also used to demonstrate how Kanban works if used in GitLab.

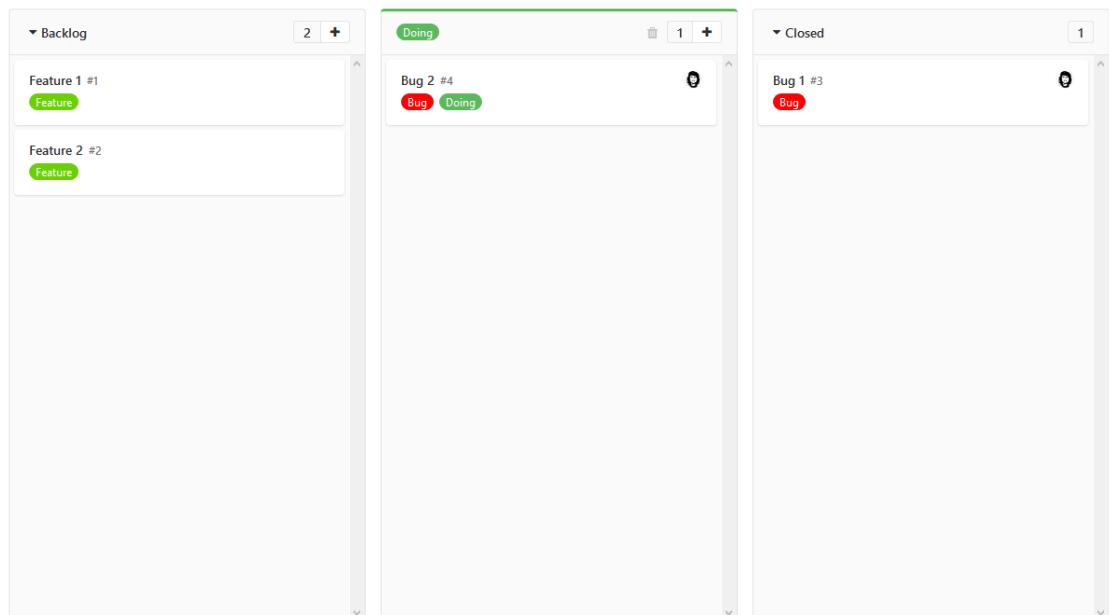


Figure 1. Simple GitLab Kanban board with four example tickets.

In Figure 1 board, there are three columns with a backlog, doing and closed and four tickets with two bugs and two feature tickets. The idea of Kanban is to simply move those tickets on the board from left to right with the designated person assigned to it. This allows seeing who is working currently on which task, and for history purposes, who worked on the ticket that is now closed.

To reduce time from demand to production, GitLab offers burndown charts for seeing how fast issue tickets are being closed as seen in Figure 2. The idea of a burndown chart is to look at the speed that tickets are being marked as closed within milestone start and end time. The dotted guideline shows how fast tickets should be

closed, and the progress line shows how well progress is really going. A wanted situation is that all tickets are closed before end time. Figure 2 already shows that developers are behind schedule and for some reason the progress has stopped on the middle.

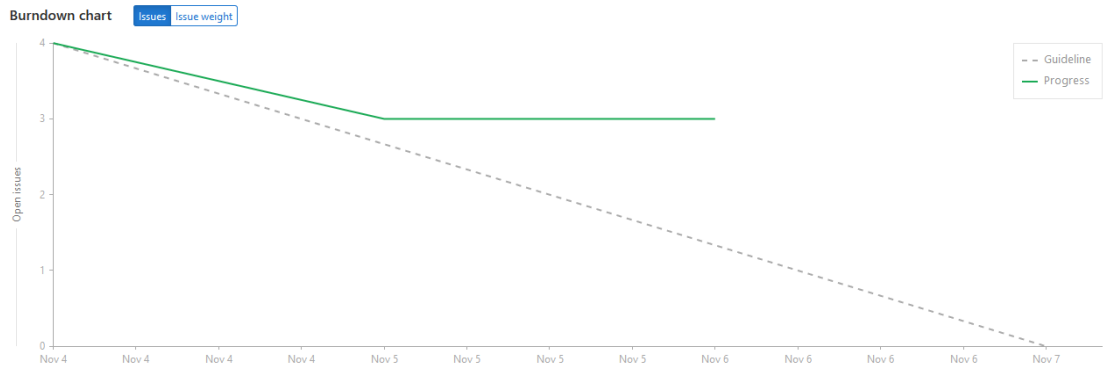


Figure 2. Burndown chart on Kanban board.

Another matter to note with issue tickets is that they can also have different weights assigned to them from zero to nine. The weight indicates how important that ticket is; if ticket has no weight there is no rush to implement it; however, if it has the maximum weight of nine, it needs immediate action from the team. GitLab offers burndown charts with weight calculated in them, which can be seen in Figure 3.

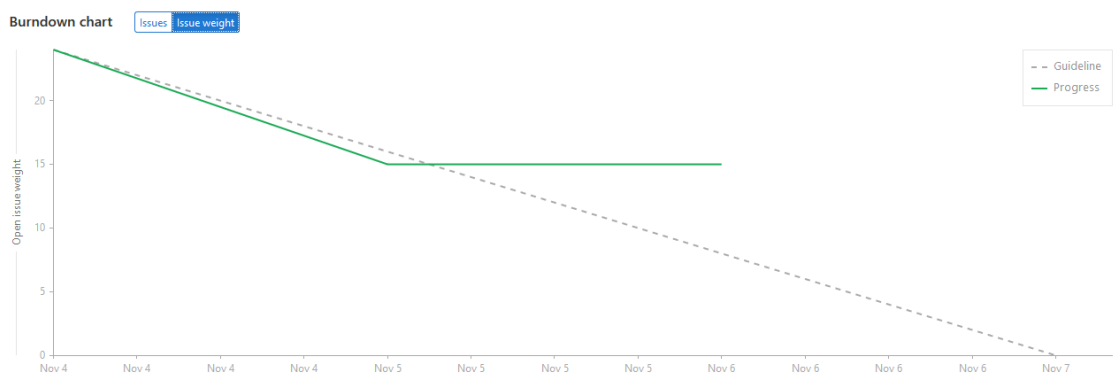


Figure 3. Burndown chart example on Kanban board with calculated weight

In Figure 2 was seen that the project was behind schedule; however, when adding the weight calculation shown in Figure 3, the project is ahead of the schedule as tickets with more weight were implemented or fixed first.

1.6 PlantUML

PlantUML is an application that allows creating UML diagrams from a text file using a simple and human readable syntax. All diagrams used in this thesis are generated with PlantUML Web Server provided by PlantUML project (PlantUML 2017). The links for the generated diagrams can be found in the References.

2 Objective

The objective of this thesis is to work as a base of documentation of the current GitLab usage in Sparta development and to propose changes that would improve it. The reason for using this as a base documentation is mainly due to using the thesis format which is quite different from a usage document. Proper usage documentation would also include a more in-depth view on Git usage, which is not included here. This thesis also introduces the history of how Sparta development team got to its current GitLab usage.

This thesis limits itself to only GitLab and does not include other tools, which was requested by Sparta as for them proposing other tools would be way further in the future. Looking at this much into the future is speculation as it would require speculating how the requirements of the development team and management change.

3 GitLab

3.1 Introduction

GitLab is an open source software offering an easy Git repository management from a web UI. GitLab also features project access management, a wiki for documentation, issues for task management, and milestones to set. GitLab comes with three editions. The first is free community edition (CE) with all basic features required for project management, including repository hosting, issue tickets, merge request, issue board. Next, there is Enterprise edition starter (EES), the version containing features required for a larger project such as issue boards limited to milestones, multiple issue boards, related issues, multiple reviews for merge request issue weights, burndown

charts and many more feature for managing GitLab instance. Lastly, there is Enterprise edition premium (EEP) which is meant for very large development teams giving feature like group board to add multiple project on same board and other features required for managing GitLab with hundreds of users. (GitLab products 2017)

There is also a public GitLab EEP instance hosted by GitLab, which is also used as their development platform. For this thesis, all images from GitLab are from that publicly hosted instance using the writer's personal profile. For demoing issues, merge request and other project related views have been created in the thesis project for the writer's personal user. Below, in Figure 4 is an image from the author's personal dashboard on a publicly hosted GitLab instance (Public GitLab instance). Note that even though this thesis is about GitLab usage at Sparta, there are no images from their GitLab instance. This is for privacy reasons to make sure that no private information is accidentally released on these images.

3.2 GitLab groups

Group allows multiple project under the same namespace. Large products usually have multiple related projects, for example, there is code for a website and a separate server project which would handle requests from that website. Both projects would be added to the same group. Figure 5 illustrates what having multiple projects in one group would look like. A group can also be used to manage which users can see that group and all projects under that group.

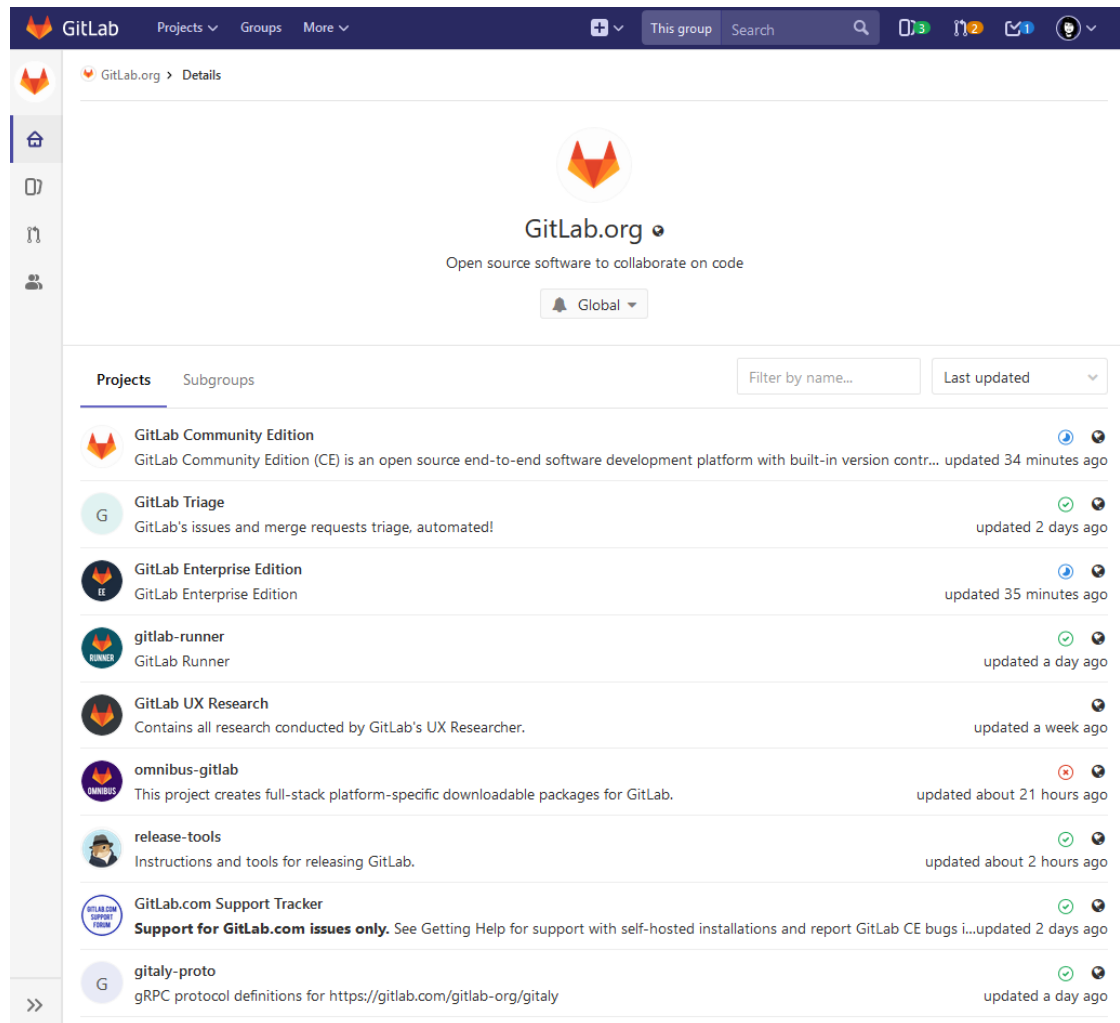


Figure 4. GitLab organization group page (GitLab, 2017)

Sparta uses one group per product where there are couple of projects related to that product. In that group, there is a project for the product itself, request for comment (RFC) and one project per customer deployment. These customer projects contain files and documentation related to that deployment.

3.3 Project

Project in GitLab is where all the files, documentation, issue tickets and other project related information are contained. For Git usage, every project is a repository where developers commit and send their code. Figure 6 illustrates how a project page looks for this thesis.

The screenshot shows the GitLab interface for a project named 'thesis' by user 'nikolauska'. The page includes a navigation sidebar on the left, a header with the GitLab logo and navigation links, and a main content area. The main content area displays the project name 'thesis', a star and fork count (both 0), and the SSH URL 'git@gitlab.com:nikolauska/thesis'. Below this, there are buttons for 'Add Changelog', 'Add License', and 'Add Contribution guide'. The file browser shows a list of files and folders with their last commit and update dates.

Name	Last commit	Last update
odt	Refactor more to latex model	3 weeks ago
old	Refactor more to latex model	3 weeks ago
src	Remove documentclass latex header	3 weeks ago
.gitignore	Move index.html to src folder and remove public	3 weeks ago
.gitlab-ci.yml	CD to src folder before generating pdf	2 weeks ago
README.md	Update readme for new org mode version	a month ago

Figure 5. Thesis project page for example.

3.4 Milestones

Milestones contain only three fields: description, start date and due date and multiple issues assigned to milestone itself. Milestones can be used to separate issues to their own projects and track how certain milestone is progressing. For example, burndown charts shown in Figures 2 and 3 are visible only in the milestone view. Figure 7 shows how milestone editor looks like on GitLab.

The image shows the GitLab milestone editor interface. It consists of several input fields and a rich text editor. At the top is a 'Title' field. Below it is a 'Description' field with a rich text editor containing a toolbar with options for bold (B), italic (I), quote, code, bulleted list, numbered list, link, and unlink. The text area contains the placeholder 'Write milestone description...'. Below the description field are two date selection fields: 'Start Date' and 'Due Date', each with a 'Clear' link. At the bottom left is a green 'Create milestone' button, and at the bottom right is a 'Cancel' button.

Figure 6. Empty milestone editor example on GitLab.

Milestones for Sparta have seen many changes in these two years. First, Sparta did not use them at all as developers, included the writer, since they did not see any reason for them. Later when issue ticket amount started to climb, and Sparta started to move towards Scrum development model, they first used them for sprints. For every sprint, there would be a milestone with end date as title. Later, there was the change on sprint length: it was to be longer with the release after every sprint. This is when milestones with version number started to appear and old sprint milestones with date were closed. The next big change for milestones took place with the change to Kanban model, which was done to better allow the product to be worked for the customer so there was no limit to how long something would take like in Scrum with sprints. This is when customer milestones appeared.

3.5 Labels

Labels are used for marking what kind of issue ticket or merge request it is or to show the status of the issue or merge requests. GitLab also uses these labels on an issue board as seen in Figure 1 where *doing* was just a simple label. The label itself does not contain other than three fields, name, description and background color. The example in Figure 8 describes what the label editor looks like on GitLab.

Title

Description

Background color

#5CB85C

Choose any color.
Or you can choose one of suggested colors below

Figure 7. Label editor on GitLab.

For Sparta, there have been multiple different ways of using labels. First, there was minimal usage with simple bug and feature. With more issue tickets and moving to Scrum and issue board usage, the status label started to appear. Later, there was a cleanup for label naming so every label with a status would be prefixed with a status, e.g. “Doing” would become “Status: Doing” and “Bug” would become “Kind: Bug” to separate what is status and what kind of issue it concerns. These *status* and *kind* labels are still in use and there is no reason for changing them. Some new labels are required for the changes proposed in issues. There should be *kind* labels for feature and task so those can be also filtered in issues list and board.

3.6 Issue tickets

Kanban tickets in GitLab are issue tickets. Issue ticket contain plenty of small extra information that can be used to manage them. The issues can come with time tracking on how long it took to implement, due date for when issue is needed to be implemented. The important part for this thesis is the possibility to assign issue ticket to milestone and adding weight to it. The issue can also contain labels. Figure 7 shows what the issue editor looks like on GitLab.

Title

Description

Write Preview B I

Write a comment or drag your files here...

Markdown and quick actions are supported Attach a file

This issue is confidential and should only be visible to team members with at least Reporter access.

Assignee [Assign to me](#)

Milestone

Labels

Weight

Due date

Figure 8. Empty issue example on GitLab.

Issues are the main part of project management and the hardest part to do correctly. At Sparta, these issues were always written without any template, so everybody could write anything from a nicely formatted issue with multiple subtitles and images to a one-line reminder to do something. What has changed in these two years is everything else related to these issues. First labels started to appear to these issue tickets. Then milestones were added to new tickets. Lately there has been activity to start adding milestones for older tickets that did not yet have them. When going through these old issues there was a new label added for issues that did not have enough information in them for the developer to understand what it requested.

3.7 Issue board

For information on how issue boards look and work the reader is advised to see Kanban Chapter and Figure 1. This chapter introduced how issue boards are used at Sparta and how their usage could be improved. From the beginning, Sparta has mainly used only one issue board with status labels as columns. The order of the first status labels in use was: “*to do*”, “*doing*”, “*blocked*”, “*ready for test*”, “*testing & QA*”, and a closed column was automatically added at the end. *To do* status would mean that the issue ticket is marked for this sprint. *Doing* meant that the developer was working on it. *Blocked* meant that the issues ticket cannot be worked on before something else was done. *Ready for test* meant that a merge request was made and is ready for testing. *Testing and QA* meant that a merge request has been merged, and it requires testing again after merging. An issue ticket could be closed after the developer has tested merge request changes after merging. Figure 13 shows how this looked: the colored columns are meant to show which columns do not use labels but use GitLab functionality.

| To do | Doing | Blocked | Ready for test | Testing & QA | Closed |

Figure 9. First version of issue boards (Niko Lehtovirta, 2017)

Later, when changing to use milestones filtering by milestone was added to the issue board, it only shows issue tickets marked to the current milestone. This is when “*to do*” label was removed and replaced with a backlog where issue tickets without status label are found. This was simpler as issue tickets that should be done in sprint did not require separate label but were automatically added to the backlog. Figure 11 demonstrates how this changed the board.

| Backlog | Doing | Blocked | Ready for test | Testing & QA | Closed |

Figure 10. Issue board after replacing “*to do*”. (Niko Lehtovirta, 2017)

The next change to status labels were new RFC labels being added to before doing the marking that there is an RFC ready for approval. This RFC would need to be approved before the developer can start implementing the proposed solution in it, which brings the current situation with issue boards demonstrated in Figure 12.

| Backlog | RFC | Doing | Blocked | Ready for test | Testing & QA | Closed |

Figure 11. Issue board after adding RFC. (Niko Lehtovirta, 2017)

3.8 Merge request

Merge request is a request containing changes made to the existing code base for review, testing and inclusion to an existing code base. The original way of sending code changes with Git was designed to be sent with an email as patches that contain changes made to be merged to main repository. This is a good solution for Linux kernel developers where there are thousands of them, however, for smaller projects and teams it is too complicated. For every change made after sending the patch one needs to send another email with an updated patch. For this reason, GitLab has merge request to make sending changes easier. Merge request can be created after pushing changes to branch and then creating a merge request from that branch. Updating merge request is easy, just pushing more changes to the branch where merge request originates from. Figure 13 shows how this works.

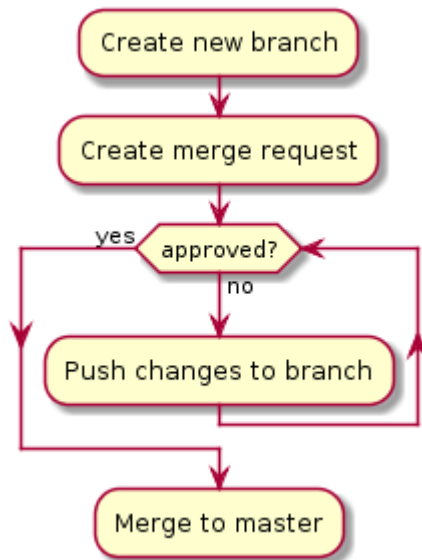


Figure 12. Merge request process (Niko Lehtovirta, 2017)

Merge request also contains other good functionality than just showing changes. It has commenting so developers can discuss the implementation, review functionality where the developer can add a comment to change and discuss the exact point in many changes. The latest GitLab 10.1 update has added the functionality to add review comments to images so this will most likely be used more often in the future. Figure 7 illustrates an example on what an empty merge request looks like on GitLab.


Title

[Start the title with WIP:](#) to prevent a **Work In Progress** merge request from being merged before it's ready.
Add [description templates](#) to help your contributors communicate effectively!

Description

Write Preview B I " </> ☰ ☷ ☑ ✕

Write a comment or drag your files here...

Markdown is supported  Attach a file

Assignee [Assign to me](#)

Milestone

Labels

Approvers

This merge request must be approved by these users. You can override the project settings by setting your own list of approvers.

This merge request must be approved by members of these groups. You can override the project settings by setting your own list of approvers.

Figure 13. Empty merge request editor example on GitLab.

On project settings it is possible to set that every merge request requires that CI tests are passed, and that the one other developer has accepted the changes before it could be merged. These options are currently enabled in the product project.

Merge request also allows linking to an issue ticket by just adding a link or an issue ticket id to the merge request description. There are two options how this can be used. The first is that there is just an issue ticket id without any specific text before that as in Figure 15 which GitLab calls as mentioning. This adds the merge request status to a linked issue ticket as shown in Figure 16. Another option is to add fixes or closes to the text before the issue ticket link as shown in Figure 17. When closes or fixes is used, the issue ticket is automatically closed when the merge request is merged, and this is shown also in issue ticket in Figure 18.

Implementation for feature

#1

Request to merge `fys` into `pages` (11 commits behind) Check out branch

Merge Remove source branch Squash commits Modify commit message

Mentions #1

You can merge this merge request manually using the [command line](#)

Figure 14. Merge request issue ticket mentioning

Feature 1

This is an example feature issue ticket

Edited 36 minutes ago by nikolauska

Related issues 1 +

#4 Task for feature 1 x

1 Related Merge Request

!1 [Implementation for feature](#) Open

Figure 15. Related merge request on issue ticket

Implementation for feature

fixes #1

Edited 2 minutes ago by nikolauska

Request to merge `fys` into `pages` (11 commits behind) Check out branch

Merge Remove source branch Squash commits Modify commit message

Closes #1

[Assign yourself to this issue](#)

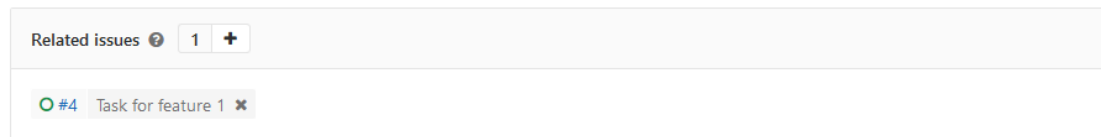
You can merge this merge request manually using the [command line](#)

Figure 16. Merge request when it closes issue

Feature 1

This is an example feature issue ticket

Edited 41 minutes ago by nikolauska



Related issues 1 +

#4 Task for feature 1 x

1 Related Merge Request

!1 [Implementation for feature](#) Open

When this merge request is accepted, this issue will be closed automatically.

Figure 17. Issue when related merge request closes it

3.9 Pipeline

“Investing in CI results in fast feedback on code changes. Fast as in “within minutes” fast. A team that relies primarily on manual testing *may* get feedback in a couple hours, but in reality, comprehensive test feedback comes a day—or *several days*—after the code gets changed. And by that time more changes have occurred, making bug-fixing an archeological expedition with developers digging through several layers of code to get at the root of the problem.

That is decidedly *not* fast.” (Dan Radigan, Continuous Integration)

Pipelines in GitLab mean a collection of CI tests that can be run in a determined order or simultaneously. From the start of the work on Sparta the writer has been managing the CI testing and studying new solutions to allow for even more automation using GitLab CI. This has meant learning to use Docker for virtualization to run tests separate from the server and ways how to secure things that need to be run on the server itself. This thesis will not go in detail about those but will go through their usage and how they could be improved.

The first versions of CI usage in Sparta were running unit tests. Unit tests mean testing a single functionality of an application which developers use to validate that their implementation works as expected. The next code linter test was added; which means the code was required to be of certain style, for example, a line should not exceed 80 characters. After this came the release package built tests, which means that every pushed commit creates a release build to see that it properly builds. This also

took place when release packages were introduced to be built as artifacts for download. Then came the first deployment from the release package. Every new commit on the master branch would be deployed to the local server and be available for developers and testers for testing. The next was the deployment for documentation and code coverage. Every commit to master would now also deploy automatically generated documentation and a coverage report to local server for developers to read. This is also when a test for the documentation generated was added to every merge request. Then came the support for integration testing on every new commit to the master branch. Integration testing means that tests are not run for single functionality as with unit tests but for a single feature. The single feature usually contains many functionalities that work together to make it work, e.g. creating a new user to a website. A unit test would test that the user creation works to database, but not if that information comes correctly from user browser to server and then to database. Lastly, the deployment got another update when support for GitLab environment deployment was added. GitLab has an integrated support for different types of deployment where one can give an environment name for deployment and follow which commit was used and when for that deployment. Later, GitLab versions added support for monitoring these environment deployments, however, that feature is currently not being used.

4 Proposed improvements

4.1 Milestones

The current situation is that there are milestones for release and customer. Having only these has created some challenges. First, having a larger change would need to be assigned to some of these milestones so they could be tracked, which might not really be a part of any release or be customer specific, e.g. code quality. There are issue tickets which improve code quality by adding more tests for a certain functionality. These issue tickets are not for any release or customer specific.

Milestones usage can be improved by creating more milestones for separate larger parts. The previous example could be added to a separate milestone called code

quality that can contain all code improvement issue tickets. Now they will not show on any release or customer milestone and can be worked on behind the scenes on a separate milestone. Simply put, more large background changes should always come with a separate milestone, so every issue ticket can have some milestone it belongs to, thus making it easier for developers.

4.2 Issue tickets

Issue tickets without templates are easy to write, however, without any formal template not all issue tickets might have enough information included in them. To be correct, having templates will not fix having enough information included in issue tickets; however, it would help.

What this thesis proposes is to have a default template that does not require too many different things but just enough to start a discussion where the rest of the information can then be added.

GitLab allows a project to have a default template for every issue ticket and this is not currently being used at Sparta. This default template should be designed to have only a few fields. The first field is description where information is given on what this issue ticket wants. The next field is for use cases which should contain information on for who this issue ticket is and why it is required. Figure 19 shows what this default template would look like.

The image shows a screenshot of a GitLab issue ticket form. At the top, there is a 'Title' field with a placeholder text 'Add description templates to help your contributors communicate effectively!'. Below the title field is the 'Description' field, which contains a template with the following structure:

```

Write Preview
B I " <> :≡ ≡ ☑ ☒
## Description
(What this issue proposes to be changed or added?)
## Use cases
(For who and why is this issue ticket for?)
Markdown and quick actions are supported
Attach a file

```

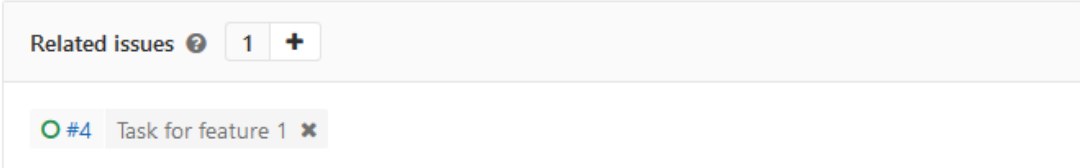
Figure 18. Default template for issue ticket.

Another change for issue tickets are not related to templates but linking issues with each other. GitLab 9.5 added the related issues feature, which allows having links to other issue tickets included in the issue ticket. Currently, only some of issues tickets have these links, which should be used more actively to follow which issue tickets are related to each other as they work both ways when added. For example, in Figure 20 and 21 the related issues were only added on the task issue ticket in Figure 21. As seen in Figure 20, it is also now visible there; so adding these does not require adding them to both issue tickets.

Feature 1

This is an example feature issue ticket

Edited less than a minute ago by nikolauska



Related issues ⓘ 1 +

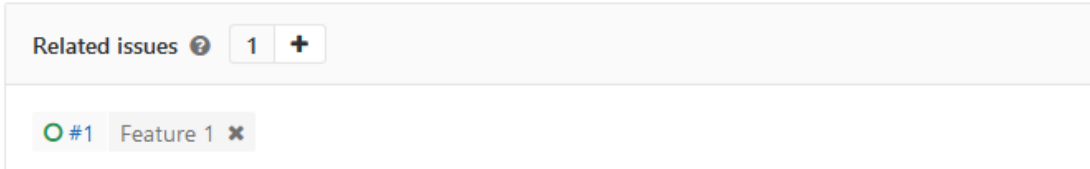
🟢 #4 Task for feature 1 ✕

Figure 19. Example feature issue ticket

Task for feature 1

This is a example task ticket

Edited less than a minute ago by nikolauska



Related issues ⓘ 1 +

🟢 #1 Feature 1 ✕

Figure 20. Example task issue ticket for feature

The last and biggest change for issues is to move all discussion to them. Currently there is another project called RFC (Request for comment) where large architecture

changes are documented. There is no rule currently on what is large enough a change to require an RFC so only some changes have RFC documentation written. Another challenge with the RFC is that it requires developers to create a merge request to separate a project, which takes time that could be better spent discussing ideas on how the feature could be implemented. Therefore, this thesis proposes removing the RFC and including documentation in issue comments where the whole discussion for code changes can be found in a single place. This change also makes it easier for developers to propose solutions as they can do it instantly and not assign themselves to creating merge request to an RFC project where the discussion is held.

4.3 Issue board

The current challenges with issue boards are that there are not enough of them. There is only one board which by default is filtered to a milestone. In addition, there are not enough other types of boards that could be useful in some situation, e.g. having a board with all different “kind” labels as columns, which would allow seeing how many issue tickets are marked as bug, for example.

What this thesis proposes is simply to have more boards; a default board that opens when viewing issue boards should be named as “all” with status labels such as the currently used board. This issue board would not be filtered to any milestone by default. Second board is so called “kind” board with all columns having different “kind” labels in them. This would allow easily seeing how many issue tickets are marked with a bug or cleanup, for example. The rest of the boards should be for milestones so that every milestone has a board attached to them. This can be done by filtering, however, by having a readymade board for milestone makes it easier for developers to add a link to it if required as seen in Figure 22 how this would look like.



| Backlog | Bug | UI | Cleanup | Closed |

Figure 21. Issue board with *kind* labels. (Niko Lehtovirta, 2017)

The next proposed change is to rename the RFC label to design. This is due to the proposal on the issue ticket chapter earlier to remove the RFC project and move discussion to the issue. This design status would mean developers are currently discussing and designing a solution on how issue ticket could be implemented. No other status label changes are proposed. Figure 23 illustrates how this would change the status issue board.

| Backlog | Design | Doing | Blocked | Ready for test | Testing & QA | Closed |

Figure 22. Issue board after changing RFC to design. (Niko Lehtovirta, 2017)

4.4 Merge request

Currently the merge request does not use any indication where it belongs to, which means, to see where a merge request belongs to, a developer must open the merge request and look for that information from related issues. This thesis proposes that all merge requests should start to include milestones. This allows with a quick look at the merge request list to tell where the merge request belongs to. The milestone for merge should be the same as the issue ticket it is related to.

Currently, the merge request also has the same kind of challenge that there is with the issue tickets with freely types description field without any templates. The challenge comes with how every merge request looks different so finding related issue tickets might be hard. Another proposal for the merge request is to add a default template to them. GitLab has allowed this from the version 6.9, however, it has not been used in Sparta projects yet. The default template should have only three fields to make it small as issue tickets are the main place. The first is the description where a developer writes a short summary of what this merge does. Next come the related issues where a developer adds a link to all issue tickets this merge request is related to. Lastly, there are tasks for work in progress merge requests, Figure 24 shows the image of this template. There is also Figure 25 which shows how the template would look, for example, for a merge request.

Title

Start the title with `WIP:` to prevent a **Work In Progress** merge request from being merged before it's ready.
Add [description templates](#) to help your contributors communicate effectively!

Description

Write **Preview**

Description

(Summary of what this merge request changes)

Related issues

(All relates issue ids should be here)

Tasks

First thing to do

Second thing to do

Figure 23. Merge request template

Implementation for feature

Description

This merge implements solution proposed on issue [#1](#) by adding new table to database.

Related issues

[#1](#)

Tasks

- Database changes implemented
- Application logic updated
- Unit tests added

Edited less than a minute ago by nikolauska

Request to merge `fys` into `pages` (11 commits behind) Check out branch

Merge Remove source branch Squash commits

Mentions [#1](#)

You can merge this merge request manually using the [command line](#)

Figure 24. Example merge request with template.

4.5 Pipeline

For every merge request there is unit, documentation, release and linter tests run. After merge, the same tests are run again to make sure that merging worked properly. Then a new version is deployed to the master environment and integration tests are run for that last deployment. Next, new documentation and code coverage are generated and deployed to the server. Lastly, there is release building where the release can be built manually by pressing the button on GitLab. Release build is not done on every commit as those are only sometimes needed, and this will save space on the server so there are not hundreds of release builds kept on server for download.

The current challenge with this setup is that integration tests are not being run on merge request, so some approved merge requests might include a bug that was not found on manual testing but is found with automated testing. This means the developers need to fix it in another merge request when it could have been easily found on the first merge request if integration testing would have run on merge request. So, the first proposal is to fix that issue and set up integration tests to run on every merge request.

Next, there is a feature in GitLab that is not used currently in any way. GitLab Pages allow hosting static websites with GitLab CI just by adding pages job and creating artifacts from those HTML files. This feature would be just right for documentation and code coverage as they are simply static HTML files. This would also make their management easier by removing manual management on the server.

The last and biggest change for CI is the deployment. Deployments are currently using environments properly, however, how deployments are made does not allow better integration to GitLab. Currently the deployment is done manually by running release scripts on a shell in the server. GitLab recommends using Kubernetes for deployment as its usage is integrated in GitLab. This would allow monitoring deployments and creating dynamic deployments. GitLab calls these dynamic deployments Review Apps. This means every merge request can be deployed and tests could be run there so a tester or developer would not have to run the product locally to test the changes in merge request work as intended. This has been marked as a large

change due to a major change to how the product is currently being deployed, and this is the change that would take many weeks to get properly working at even a level that the product is currently being deployed.

5 Results

After introduction of these changes, smaller ones have already been implemented. These include changes to board, milestone and template usage. Changes to milestones have already made it easier for issue tickets that did not belong to any currently used milestones to have one. New milestones have also helped with issue boards. Every milestone now has its own board with all status columns, so the developers and project manager can now easily see what the status of any milestone is. Custom boards are also implemented which are not meant for Kanban usage as those do not have status columns. There is a board for feature tickets where the project manager can easily see how many open feature requests there are with optionally limiting board to a certain milestone. Another custom board is customer board. This contains a column for each customer request, so the project manager and product owner can see the situation for how many requested features are still being worked on for every customer. Adding templates has also made minor improvement on standardizing how issue ticket and merge request should look like. There is not yet enough usage to tell if it will help with challenges regarding issue tickets without enough information of what is wanted. The proposed changes to CI have been accepted and are being implemented. These are larger changes which require much more time to be implemented properly, which means there is no information yet on how their implementation will change the usage of GitLab in the Sparta development team

References

About Github. 2017. Website. Accessed on 14.11.2017. Retrieved from <https://github.com/about>

Dan Radigan, Continuous integration. Accessed on 17.11.2017. Retrieved from <https://www.atlassian.com/agile/continuous-integration>

GitLab. 2017. Website. Accessed on 14.11.2017. Retrieved from <https://gitlab.com>

GitLab products. 2017. Website. Accessed on 16.11.2017. Retrieved from <https://about.gitlab.com/products/>

In the light of better information. 2017. Website. Accessed on 04.11.2017. Retrieved from <https://spartaconsulting.fi/en/company/>

Kanban origins. 2017. Website. Accessed on 27.11.2017. Retrieved from <https://en.wikipedia.org/wiki/Kanban>

The Internet Engineering Task Force. 2017. Request for Comments (RFC). Accessed on 14.11.2017. Retrieved from <http://www.ietf.org/rfc.html>

Michael James. 2015. An empirical framework for learning (not a methodology). Website. Accessed on 16.11.2017 <http://scrummethodology.com/>

Niko Lehtovirta. 2017. PlantUML design issue board. Website. Accessed on 17.11.2017. Retrieved from <http://www.plantuml.com/plantuml/uml/SoWkllm-gAStDuQfHTimholmiJGtFo2n9hN5AJCxEOk-luQf-nlltEJCy3CV8pyq02JZdvoTcfAK1MK6fYlgL2MdwHGabgSGbG80H80HKfg2a4EiQW6-cSaryCqkQGcfS2j140>

Niko Lehtovirta. 2017. PlantUML first issue board. Website. Accessed on 17.11.2017. Retrieved from http://www.plantuml.com/plantuml/uml/SoWkllm-gAStDuQe9oLT8oQ_Xgd79pynB1z9EEVd9sQafG5PGQc9AfK9QVf52IMfn2L0W14W15IceAGGw0hdAZdabcOMbgU4PclMfDSuv-SKWRGwfUIb0Wm40

Niko Lehtovirta. 2017. PlantUML homepage. Website. Accessed on 14.11.2017. Retrieved from <http://plantuml.com/>

Niko Lehtovirta. 2017. PlantUML merge request process. Website. Accessed on 16.11.2017. Retrieved from http://www.plantuml.com/plantuml/uml/FOun2iCm40JxUyMLrXUa8eRKWNn1iHVBO4dEtld6luy8mUqo-Eyoe5iLQtiZV1T701GU_99OfXMIba0KiU3Ue1Q8ZhHjuYFikzOFvvdXL7afnEoVJHD-FAPO5obOI6Shy_vrGH-GVFsidsbI311exqbE-

Niko Lehtovirta. 2017. PlantUML second issue board. Website. Accessed on 17.11.2017. Retrieved from http://www.plantuml.com/plantuml/uml/SoWkllm-gAStDuQfHTimholmiJGtFo2n9hN5AJCxEOk-luQfnoS_C0z5EEVd9sQafG5PGQc9AfK9QVf52IMfn2L0W14W15IceAGGwrg2Rv9oJNmplvf2QbmAg2000

Niko Lehtovirta. 2017. PlantUML third issue board. Website. Accessed on 17.11.2017. Retrieved from <http://www.plantuml.com/plantuml/uml/SoWkllmgAStDuQfHTimholmiJGtFo2n9hN5AJCxEOk-luQe9SdC6aYxvc-NaWdbDEVd9sQWf8F3KnfL8XBRz8eIlrk0Ge40AaW8eKr1G27MIGpNFEoly6QND8pKi1MWSO>

Niko Lehtovirta. 2017. PlantUML "kind" issue board. Website. Accessed on 17.11.2017. Retrieved from <http://www.plantuml.com/plantuml/uml/SoWkllm-gAStDuQfHTimholmiJGtFo2n9hN5AJCxEOk-luQfnAWM-HePv0mZadDJ6ljA06NR5pJdvnQaeDbqDgNWhGB000>

Rust RFCs. 2017. Website. Accessed on 14.11.2017. Retrieved from <http://rust-lang.github.io/rfcs/>

Sparta Consulting Oy. 2017. Website. Accessed on 11.11.2017. Retrieved from <https://tietopalvelu.ytj.fi/yritystiedot.aspx?yavain=2416943&tar-kiste=48EBEB20D93CE66B297B3BFB7ADAF4B202B625FE>

Taiichi Ohno. 2017. Website. Accessed on 16.11.2017. Retrieved from [https://en.wikipedia.org/wiki/Taiichi Ohno](https://en.wikipedia.org/wiki/Taiichi_Ohno)

Thomas R. Cutler. 2006. Examining Lean Manufacturing Promise. Accessed on 16.11.2017. Retrieved from <https://web.archive.org/web/20130526100928/http://www.softwaremag.com/content/ContentCT.asp?P=3193>