# Handling issues – managing requirements in GitHub with lean principes

Guideline by Risto Salo

## What’s this all about?

- Requirements management (RM) is important topic in any software project
- To succeed in RM process one must have a good knowledge about the tools used and a systematical rules to follow
- This presentation aims to introduce a guideline that will help users accomplish these aspects using only GitHub’s native features

## GitHub and RM

- The more there are tools in a project, the more people need to remember things
  - Combining as such as possible reduces this
- GitHub is intuitive to use and offers some neat features, making it well-suited for many projects
- GitHub has a lightweight issue tracker that can easily be used for task handling, but what if the desire is more systematical RM process?
- There is very few instructions how this could be achieved
  - Instructions found are mainly experience reports from individual cases
- This guideline tries to address that issue

## What should you know before continuing?

- What agile software development means?
- What is GitHub and how its features work?
  - Especially issue tracker and its concepts and tools such as: milestones and labels
- Requirements engineering (what it is) and especially requirements management
  - Introduced very briefly

# THEORETICAL PART

## What is requirements management?

- The last step of requirements (RE) engineering process
  - RE studies and identifies the descriptions of services and its operational boundaries that are needed to adequately solve customer's problem or fulfill a contract (= build a working software product)
- The steps before RM collect, evaluate and document the requirements for a software
- RM is responsible for understanding, maintaining and controlling requirements

## What is requirements management?

- RM is usually divided to four activities:
  - Change control – how changes to requirements are handled.
  - Version control – Identifying and versioning requirements and associated documentation.
  - Tracing – Establishing links between other requirements and system components
  - Status tracking – Observing the progress of requirements.

# Agile RE

- RE is traditionally considered ”anti-agile” because it relies on defined processes and heavy documentation
- However agile projects *shouldn’t* neglect RE
  - RE process can be tailored to fulfill agile needs
- Important aspects for agile RE:
  - Customer/stakeholder collaboration and involvement
  - Replacing heavyweight documentation with a lightweight alternative
  - Approaching RE iteratively

## Lean software development

- Derived from Japanese car manufacturer’s (Toyota) production principles
- Lean manufacturing was invented in 1950s
- Toyota couldn’t compete with American mass production lines so they needed another way to be prolific
  - This lead to the invention of lean manufacturing formerly known as Toyota Production System (TPS)
- Focused on two core principles:
  - Just-in-time flow (JIT) – deliver the right amount in the right time
  - Stop-the-line (Jidoka) – stopping production line immediately when problem arose

## Lean software development

- Was later converted to an applicable form in software development by Mary and Tom Poppendieck
- They introduced seven core principles to utilize:
  - Eliminate waste – identify and remove every process that doesn’t generate direct value for customer or knowledge of how to provide that value
  - Optimizing the whole – Seeing the whole picture and acting towards it. Fix problems not just symptoms.

# Lean software development

- – Building quality in – Defect free product that functions as it is desgined to.
- – Learning constantly – Create knowledge and embed that into use.
- – Delivering fast – Rapid iteration and feedback cycles.
- – Respecting people – Empower and trust the team, they should be the ones making decisions
- – Deferring commitment – Defer commitment to the latest possible moment.

- Why lean explained in slide 15

## THE GUIDELINE

## Purpose

- To offer a set of recommendations and practices for RM in GitHub
- Individual parts can be used to some extend as such, but doing so may not provide the best result
- To make the guideline applicable to as many situations as possible, there is room for customization

## Purpose

- Guideline mainly concentrates on the issue tracker though some thoughts about the repository and wiki are also provided
- Therefore the guideline mainly focuses on aspects concerning issues: how they should be used, created and monitored to achieve a consistent RM process.

## Purpose

- The guideline aims for complementing the four activities of RM
- As GitHub’s issue tracker is quite lightweight according to its functionalities, it is very useful for projects using agile approach
- Lean software principles are used for assessing the guideline and its compatibility to the agile environment
- Why lean?
  - Lean software principles are abstract enough so that they don’t narrow down cases where the guideline would be useful, but still offer enough guiding for assessment against the principles

## Terms

- Issue
  - Vague concept within GitHub itself
  - For the guideline, the term issue is used as a higher level concept. It includes both requirements and tasks, which will be explained in the next paragraph.
- Requirement and task
  - Separated specialized issues
  - Task is a concrete item, whether it is implementation, designing or something else, that must be done in order to fulfill defined requirements
  - Requirements hold, in predefined formats, the goals and business objectives for the software

## How to use the guideline

- The guideline expects that its users are familiar with the features and functions of GitHub
  - If you are not, go and experience with GitHub!
  - You don’t need to master every single feature, but the basic knowledge is expected
- The guideline utilizes GitHub’s special hypertext formatting GFM (GitHub Flavored Markdown)
  - The syntax is quite simple so while you are testing GitHub, learn it as well

## How to use the guideline

- When you are familiar with GitHub, few preparations are needed:
  - Decide label categories, exact labels, their names and color coding
  - Decide overall naming conventions for issues
- The above aspects are discussed further down this guideline
- Come back to these decisions when you have read the guideline

## How to use the guideline

- Other important note is that the project team should decide the customized processes and practices **before** putting the guideline into use!
  - If the initial decisions prove to be wrong, they can always be changed
  - What is crucial is that the team agrees to follow the set rules consistently
- Therefore the team should appoint someone as responsible for RM
  - Usually the appointed person is project manager but it can be whoever monitors the requirements
  - *The whole team* is responsible of RM process -> the purpose of appointment is to clearly define from whom to ask should a problem with issue or the guideline arise

# How to use the guideline

- The guideline isn’t interested RE processes before RM
  - What concerns the guideline is that the requirements are somehow identified
- The optimal case is that when requirement is identified, it is immediately created to the issue tracker (see slides 30-43 for creating an issue)
  - This is not mandatory should the requirement be documented some other way first
  - Be aware that before requirements (or at least a sub set of them) is in issue tracker, the team can’t start the implementation
- As soon as requirements are created team can start splitting them to tasks
  - When first tasks are ready, implementation is ready to begin

## How to use the guideline

- Requirements are split to tasks in the order of their priority
  - This contributes for implementing the best business value from the get go
- Creating requirements and tasks is an ongoing process through the whole project
- When first tasks are under work, they must be monitored, maintained and updated if needed

# Separation of tasks and requirements

- Why the guideline differs tasks from requirements?
  - A single requirement can cover a large topic, making it hard to estimate time and other resources needed to complete it
  - In order to track this, we need to know what really needs to be done to fulfill the requirement
  - > Tasks are concrete actions needed to to accomplish the business objective (requirements)
  - Keeping both tasks and requirements in the issue tracker makes it possible to establish links between them

# Hierarchy between requirements and tasks

- Both requirements and tasks can be split to smaller pieces – sub-components
  - Splitting issues further helps eliminating wasted time that goes searching information from long descriptions
  - Smaller issues are easier to track
  - Splitting issues acts towards increasing the domain knowledge of the team
- This brings the total number of different issue types to four: main requirement and main task (or just requirement and task) + sub requirement and sub task
- Lightweight hierarchical structure increases the visibility of requirements and associated work

# Hierarchy between requirements and tasks

- An example:
  - The project has identified a main requirement ”The game is divided to puzzles” and its sub requirement ”There are total of 10 puzzles”
  - Now one main task could be made to depict every puzzle
  - Sub tasks could further define such elements as UI and business logic
- For the sub tasks to be usable, they must not be too fine grained
  - If sub task should be split to smaller parts, it must be ”promoted” to a main task -> this can cause other modifications to the other issues in the same hierarchy tree
- This enforces the maximum depth of issues to four (see slide 29)

## Sub tasks: yes or no?

- Sub tasks are not always needed
  - Too small tasks cause more work to create and maintain
  - Especially if developers are experienced, forcing them to create sub tasks might be waste of time
- A main task coupled with a task list might suffice
- On the other hand, splitting main tasks to sub tasks can aid in recognizing all the aspects associated to the main task

## Sub tasks: yes or no?

- What else is good with sub tasks?
  - The work related to them is displayed and traced better
  - If all the information is in the main task, it is harder to find
  - If project team has a lively communication, the comment section of a big main task can be quickly overrun, becoming hard to find the relevant comments
- What about not using sub tasks?
  - When an issue contains a task list, its status is visible in issue references, though it only shows how many of task list's items are completed, not what item exactly
  - Negates overhead caused by creating and maintaining sub tasks
- If developers really don't want to use sub tasks (even if there are valid reasons for using them), forcing them decreases motivation towards the guideline and its usefulness

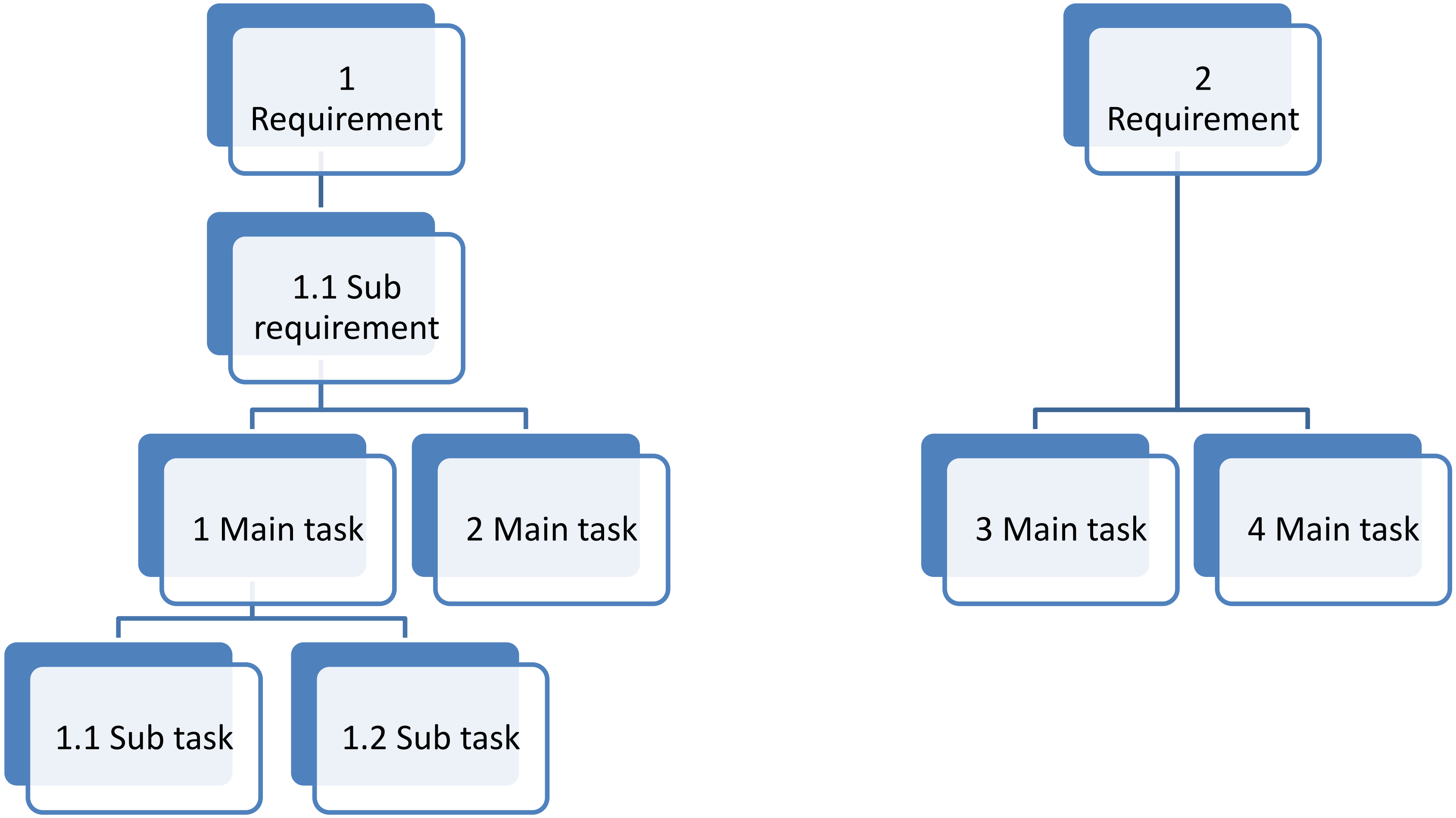
## Sub tasks: yes or no?

- There is no right answer for using or not using sub tasks
  - Decision should be made case by case
- Don’t overlook sub tasks because they require some more work - when information is separated to logical components, it is easier to find and interpret
- The important thing is that all the necessary information is present, logically structured and findable with minor effort

## Rules for issue hierarchies

- A main requirement must always exist.
- A sub requirement must always descend from a main requirement, same applies to tasks.
- A main task can be descendant of either a main or sub requirement.
- If both a main requirement and a sub requirement exist a main task must be descended of the sub requirement.
- A main requirement can have multiple sub requirement descendants. Multiple main tasks descendants are allowed to a single main or sub requirement and multiple sub tasks can be descendants of a single main task.
- All issues must be unambiguous (i.e. a single main task can be a descendant to only one main or sub requirement).
- Hierarchy is generated by naming conventions, labels and references

# Hierarchy



1 Requirement
1.1 Sub requirement
1 Main task
2 Main task
1.1 Sub task
1.2 Sub task
2 Requirement
3 Main task
4 Main task

# Creating issues

- Activity for the whole team!
  - Although requirements may be drafted with limited personnel, creating tasks should involve the whole team.
  - Benefits: seasoned developers might have useful ideas or views, letting developers contribute enhances the team spirit, broadens the domain knowledge of the team
- First requirements then tasks
- Team can start just a handful of requirements, converting them tasks and starting the work. The rest are gained from iterations.
- Other approach is to define a larger set of baseline requirements and only after that create the tasks

# Steps for creating a single issue

- **Step 1 – the title of the issue**
  - The title should contain some kind of reference prefix that is different for tasks and requirements
  - Examples: “(XXX-100): Title goes here” or “#123: Title goes here” for tasks and “REQ 2 Title goes here” or “R2: Title goes here” for requirements
  - Should support the hierarchy structure - if a main task has prefix format “XXX-100”, the sub task could have “XXX-100-1” or “XXX-100.1”

## Steps for creating a single issue

- **Step 2 – Writing the description**
  - The main question is: how extensive should the description be?
    - A lot of influencing factors: the experience of the team, the size and domain of the project, who creates and manages issues, is another documentation tool used?
    - Too much information makes reading and finding information cumbersome – too vague description might cause guessing, misleading or a need to consult somebody
    - Leaving room for individual thinking and implementation endorses the team

## Steps for creating a single issue

    - Requirements must be as unambiguous as possible to make sure that every stakeholder understands them the same way
- The bottom line is that issue should always have a description which is tangible
    - Anyone from the project team should be able to tell what the task truly means just by reading the description
- As much of the task specific information as possible should be kept in the description
- If sub tasks are not used make sure that the description covers also those aspects -> the relevant information must not be lost or ignored
- A large amount of pictures, documents or other relatively static material should be archived somewhere else
    - For example mockups of UI are usually related to multiple requirements or tasks, and therefore should be only in one place

# Steps for creating a single issue

- For requirements it might be a good practice – especially if they have a lot of information themselves – to document them to the wiki. To the requirement issue, a short description and link is sufficient in this case.
- A predefined format for description should be used
- The guideline suggests the following sections for the description:
  - Description – The core idea in few sentences
  - Information – All the relevant information -> use links if data is stored somewhere else
  - Task list – Only present in main task or sub task -> list followable items that are not split to own (sub)tasks
  - References – List all the issues that this issue references

## Steps for creating a single issue

- Manual work is needed with references because this is not automated in GitHub
  - Only existing issues can be referenced, so the main task’s references must be updated after its sub tasks are created
  - Three plausible references: parent issues, child issue and related issue
    - Related issue is an issue that has a close relationship with the referencing issue but they are not direct ancestors or descendants in the hierarchy tree
- Use mentioning (“@-notation”) to inform people/teams that should be aware of the issue

## Steps for creating a single issue

- **Step 3 – Assign labels**
  - Essential part of the guideline
  - Main solution for creating visibility and status tracking
  - The guideline doesn’t explicitly state what exact labels should be used; rather it states possible categories for them
  - There are a total of six categories suggested by the guideline, they are presented on the next slides

## Steps for creating a single issue

- The type of the issue
  - Values: Requirement, Sub requirement, Task, Sub task
  - Explanation: States what higher level type the issue represents.
- The sub type of the issue (for task and sub task only)
  - Possible values: Feature, Bug, Enhancement, Other
  - Explanation: The sub type of the task. This allows for example a quick filtering for bug reports. The list of values can be a lot longer depending on what kind of parts the tasks are split. Sub type also opens the nature of the issue without a need to go through the description.

# Steps for creating a single issue

- Status
  - Possible values: In progress, In testing, In customer acceptance, Rejected
  - Explanation: In what part of the process the issue is going. These are additions to GitHub’s native statuses: open and closed. Values must be used based on the processes in the project.
- Requirement level (for requirement and sub requirement only)
  - Possible values: Required, extra
  - Explanation: In certain cases, a requirement can be closer to “nice to have feature” or “if there is time”. In such cases, this label can be used to visually distinct these requirements.

## Steps for creating a single issue

- Priority
  - Possible values: High, medium, low
  - Explanation: In conjunction with the requirement level, this label tells on how high of the prioritization list the issue is. The guideline suggests that this is mainly used with tasks and sub tasks, since these are the concrete actions needed to fulfill a requirement. If certain requirement needs to be prioritized, the priority of tasks associated should be raised to better reflect the urgency of the work. Of course the label can be assigned for a requirement or sub requirement as a reminder of requirement's prioritization but for example milestones are more suitable for this purpose.
- Miscellaneous (also called as flags)
  - Possible values: Blocked, duplicate
  - Explanation: This label shares additional information regarding issues status. For example flagging an issue as "Blocked" tells immediately, that there is something causing interference for the issue, and it must be investigated.

# Steps for creating a single issue

- Every label category should have its distinct color
  - Labels inside the category have different shades of the category color
- A short prefix in the beginning of the label name complements recognizing the labels and their meaning
- Generally every issue should have a maximum of one label from each category, miscellaneous category is an exception
- Excluding the first category the matter is not whether aforementioned are the exact categories, but that categories are decided in the first place and then used accordingly.

# Steps for creating a single issue

- **Step 4 – Assign people**
  - Who are the ones implementing this task?
- **Step 5 – Assign milestone**
  - Complement the status and priority categories of the labels
  - Use iteratively: every milestone represents an iteration to which certain requirements and their tasks are associated
  - … or to group issue that form some bigger feature

# Steps for creating a single issue

  - These are not the only ones – users are encouraged to use their creativity
    - For example have three milestones: one holds the issues that are currently worked with, one for issues that are next in line and one for issues that need more specification or wait for something else
  - Milestones can have a deadline – issues not! Issue can belong to only one milestone
- **Step 6 & 7 – Publish and reference update**
  - The issue is ready to be published!
  - After it is created, go and update the references of relevant issues (for example parent task or requirement)

# Status tracking and traceability

- Core activities for RM
- Traceability is mainly gained by the naming conventions and references, but also the labels from type and sub type categories enhance it
- The status tracking however relies on milestones and other labels, like status, priority and miscellaneous
- Monitoring these requires use of filters
  - Remember that the filters are part of the URL, so use those bookmarks of yours!

## Status tracking and traceability

- Exact filter combinations are up to the users to decide but few remarks are made by the guideline
  - Developers should follow issues that 1) mention them and 2) are assigned to them. Priority and milestones information are relevant.
  - Project managers should pay attention to tasks that are currently worked with. Milestone deadlines and task list progression are relevant.
  - Project managers should also keep an eye out issues needing special actions – like the ones flagged as ”blocked” – and bug reports and enhancement proposals

## A bug report and an enhancement proposal

- Special issues used for specific needs
- A bug report is used when somebody spots an incorrect behavior or an error (bug) in the software
  - Reporting a bug is quite easy: follow the issue creation steps but assign only labels “task” and “bug” (or corresponding labels if these are not used)
  - Should always relate to either a task or requirement – this link should be referenced in the bug report
  - If creator can’t establish a relation, someone with better understanding can complement this information
  - An own prefix for titles of bug reports is recommended
  - Bug label can also be used as a flag to indicate that a task has an error, but this is not recommended since bug information is easily lost

## A bug report and an enhancement proposal

- An enhancement proposal is a suggestion relating to a feature or a requirement that (in most cases) comes from such stakeholders who are not authorized to make the decision by themselves or the decision is up for a discussion
  - Relates to improving aspects or functionalities of the software product
  - In an agile environment, enhancements don’t need to follow a defined process
    - If proposal is larger or needs further discussion or specification an enhancement proposal issue should be created
  - Creation is done the same way as with the bug report, the “bug” label is just replaced with “enhancement”

# A bug report and an enhancement proposal

- If an enhancement proposal is accepted a new task or sub task is created for the actual work or implementation that is required
  - The issues that are affected by the enhancement should be updated to keep up with the new situation
  - The enhancement proposal issue itself is closed
- Should it get declined, it receives the “rejected” label and is closed

## Wiki and version control

- Wiki should be used to collect static documentation, though it can contain requirement specific information even if requirements are volatile
- As wiki is incorporated to GitHub repository it is easily accessed
- Should a wiki be utilized, establishing a logic structure for its pages is recommended
- A collection page can be created to gather every external document to the wiki
  - The page lists every relevant link with a short description, preferably grouped under headlines
- Should the wiki contain information regarding requirements, careful attention is to be paid that this information doesn't conflict with the issue and tasks of the requirement

# Wiki and version control

- Version control offers some interesting interaction possibilities
  - An issue can be referenced in commit messages
  - Also mentioning people is possible
- To enhance visibility of what is going on developers are encouraged to always reference the issues their commit affects in commit messages

# Updating and maintaining issues

- Issue is going to have updates during its lifecycle inside the issue tracker
- The most common forms of updates are: changing labels, assigning milestone, assigning people, commenting issue, referencing it from commit messages and closing it.
- Stakeholder communication regarding the issues should be an ever-going process
  - Challenge is that this communication can happen in different medium and cause something to change in the issue or they may reveal some new information or insights that help the implementation
  - It is absolutely imperative that wherever the communication happens, the description of the issue is always up to date and reflects the latest information

## Updating and maintaining issues

- Comment section in an issue should be actively used and encouraged
  - Benefit: leaves a history behind
  - Disadvantage: can become clogged up -> it is even more important that the description is kept up to date
  - If comment section contains debates, the outcome should always be stated in a “finishing” comment
- Be careful when deleting an old information!
  - The old information should be kept for traceability purposes should the new information for example become under a questioning for some reason
  - The description should always contain the newest information in a compact form (discussed in the issue creation steps) -> the old information can be moved to the wiki and leave just a short remark of this update to the issue

# An example of an issue’s lifecycle

- Project manager creates a sub task and assigns it to a developer X
  - Labels and description are set, people and milestones are assigned
- The priority changes by the request of a customer
  - The priority label is changed
- Developer X starts working with the issue
  - Changes the status label
- Developer X finds out a blocker problem with the task
  - Flags the issue with ’Blocked’ label
- The task is discussed in the comments and more widely in the face-to-face meeting
  - The description is be updated to reflect the outcome

# An example of an issue’s lifecycle

- Blocked-status is removed, X continues the work
  - Changes the status label as needed
  - If the issue has a task list, X updates it to keep other posted on the status
- X makes a commit and publishes it
  - References the issue in comments (and if tester is assigned/known, mentions him)
- Task is ready for testing
  - Status is changed
- Y tests the task
  - Status is changed
- Project manager checks the task and accepts it
  - The issue is closed

# Updating and maintaining issues cont.

- Issues can be closed from commit messages but this is not recommended
  - It might save the developer from going to the issue tracker, it is prone to leave the issue (for example label information) outdated
- When an issue is closed, it is wise to submit the last comment that explains why the issue is being closed
  - Helps recognizing and keeping the team informed whether the issue was closed because it was done or was there some other reason like rejection
- As important as it is to keep the description updated, is to make sure that labels are used and updated
  - The importance of the labels is to visualize issues and different aspects of them in one view -> Should labels be misused or not updated, an unnecessary waste is generated
  - Outdated information causes mistrust to used practices and interferes with RM and its objectives

## FINALLY

## After words

- This presentation was meant to introduce the principles, practices and recommendations of the guideline for handling RM in GitHub
- If you are interested in reading how well it achieves RM objectives and following lean principles, you can read my master thesis of the topic (the link is to follow when the study is published)
- I have an example repository set up in GitHub (https://github.com/Ripppe/GraduRepo) – you are free to go and take a look. The information in the repository is complementary for this presentation