Managing requirements in GitHub

The guideline by Risto Salo

Who am I?

- Risto Salo, risto.salo@uta.fi
- Fifth year CS student
- Project worker 2011-2012, pm 2012-2013
- Job experience 2 years (currently an ecommerce software developer)
- At the moment working on my master thesis

MASTER THESIS

Background

- GitHub experience from both project work courses I attended
- Especially as a pm I faced the problem how to handle requirements(/tasks) in a coherent way
  - We somewhat tried to utilize the GitHub for this, but since there were no guidelines or instructions, the results remained shallow
  - Even so, they made certain aspects better

What’s this all about?

- My goal is to create a guideline how to handle the requirement managing using purely GitHub’s own tools
- For this guideline, I try to take advantage of Lean software development ideology and take its principles into a practice

The guideline in GitHub

- The main tool that the guideline will utilize is the Issue tracker
- Other items are the version control itself (especially the commits & pushes) and the wiki
- Please bear in mind that this guideline focuses on managing requirements in GitHub
  - For example the steps related to creating requirements are not taken into account

What's there in for you?

- Somebody with an interest and (at least little) experience to help to take the best out of GitHub
- A guideline (that we didn't have in our projects!) how to handle things in a (semi) structured way
  - A word of an advice: I'm not trying to obstruct the way you work, but rather aim it in a certain direction
  - The guideline is not carved in stone, if necessary, it will be updated iteratively to achieve the best result
- The guideline will bring better visibility to the project, enhance the requirements handling and give a sense of a control, which unfortunately in many cases is lacking badly

What does it require from you?

- A will to try and really use the guideline with an open mind
  - Don’t judge it immediately!
- Certain processes must be taken into use to make the guideline to work, please stick with them
- Give your feedback!
  - As stated, the guideline is not finished and thus I hope to receive input; either with free talk or by answering the questions I come up with
  - If some aspect doesn’t work or the group doesn’t want to use it, tell me! If nothing else, it is an valuable piece of information which I can discuss in my thesis

Encouragement speech

- Especially at first, processes and practices from the guideline can seem devious or demanding.
- Nothing should be accepted without chewing it a few times but hear me out: these things are there for the benefit of the whole organization (customer, dev team, pms, other stakeholders)
  - Sometimes this can - and will - lead to individual situations where practices established feel like nonsense to some particular person
  - But don’t give it up! It will be worth it.

THE GUIDELINE (AKA MANAGING REQUIREMENTS IN GITHUB WITH LEAN PRINCIPLES)

Lean principles

- Eliminate waste
- Build quality in
- Create knowledge
- Defer commitment
- Deliver fast
- Respect people
- Optimize the whole

Lean principles

- Some of the principles can be utilized for requirements managing, some are better suited (or meant) for the software development process itself
- The world is not perfect, and probably never will be: this applies also to the processes and tools we have
- Already at this point, I have noted few such factors inside GitHub that don’t support the principles as well as they could
  - Fear not, this is a common case and should not be considered a big drawback
  - After all, as the last principle states, the goal is to optimize the whole, and as we know, whole is more than the summary of its components

ISSUES

GitHub’s issues

- Very vague concept, can basically be anything from requirements to tasks to sort of memos
- In GitHub’s context, Issues tool could be described as a lightweight task tracker and issues themself as tasks
- The guideline utilizes heavily this tool, and its usage is a vital part of the process

Issues - Labels

- A major factor to enhance the visibility!
- This guideline doesn't explicitly tell what labels should be used, rather it states the different categories that **can** be used (depending on the project)
  - Important thing here is to choose the categories based on the need of the project
- As said Issues tool is very plain which leads to a little more manual work (as of now, there really is no way of going around this)
  - This will be visible e.g. with labels

Categories

- The type of the issue
  - [Requirement | Sub requirement | Task | Sub task] (Shades of grey)
- The subtype of the issue (for task –type issues)
  - [Feature | Enhancement | Bug | Other] (Shades of blue)
- Status
  - [In progress | In customer acceptance | In testing | Rejected] (Shades of green)
  - The GitHub gives two native states for an issue: open and closed. These should of course be used in a combination with status labels
- Requirement level
  - [Required | Extra] (Shades of yellow)
- Priority
  - [High priority | Medium priority | Low priority] (Shades of red)
- Miscellaneous (~'flags')
  - [Blocked | Duplicate]

Labels

- Color coding should be used (every category with different color, every label inside that category with different lightness)
  - Example can be seen in my repo (https://github.com/Ripppe/GraduRepo)
  - Previous slide has suggestions about colors, but if you feel that they should be different, it's up to you
- Category identifier for every label (numbers in my example issues)
  - These depict the category they belong to (helps making them standout in combination with the color coding)
  - One way of doing this is to use letter prefixes like: PRI (priority), TYP (type) and so on
  - Choose wisely! It's a pain in the ass to change that later (same goes to the color scheme), although of course possible

Issues vs. Tasks vs. Requirements

- For this guideline, issue is a general definition for requirements and tasks
- Tasks are the actual jobs that (mainly but not limited to) the programmers do
- Requirements on the other hand are usually quite big and they require multiple tasks to be done
  - This means that every task should relate to some requirement! (If not, why it is being done?) But it doesn’t need to be strictly a parent – child like hierarchy

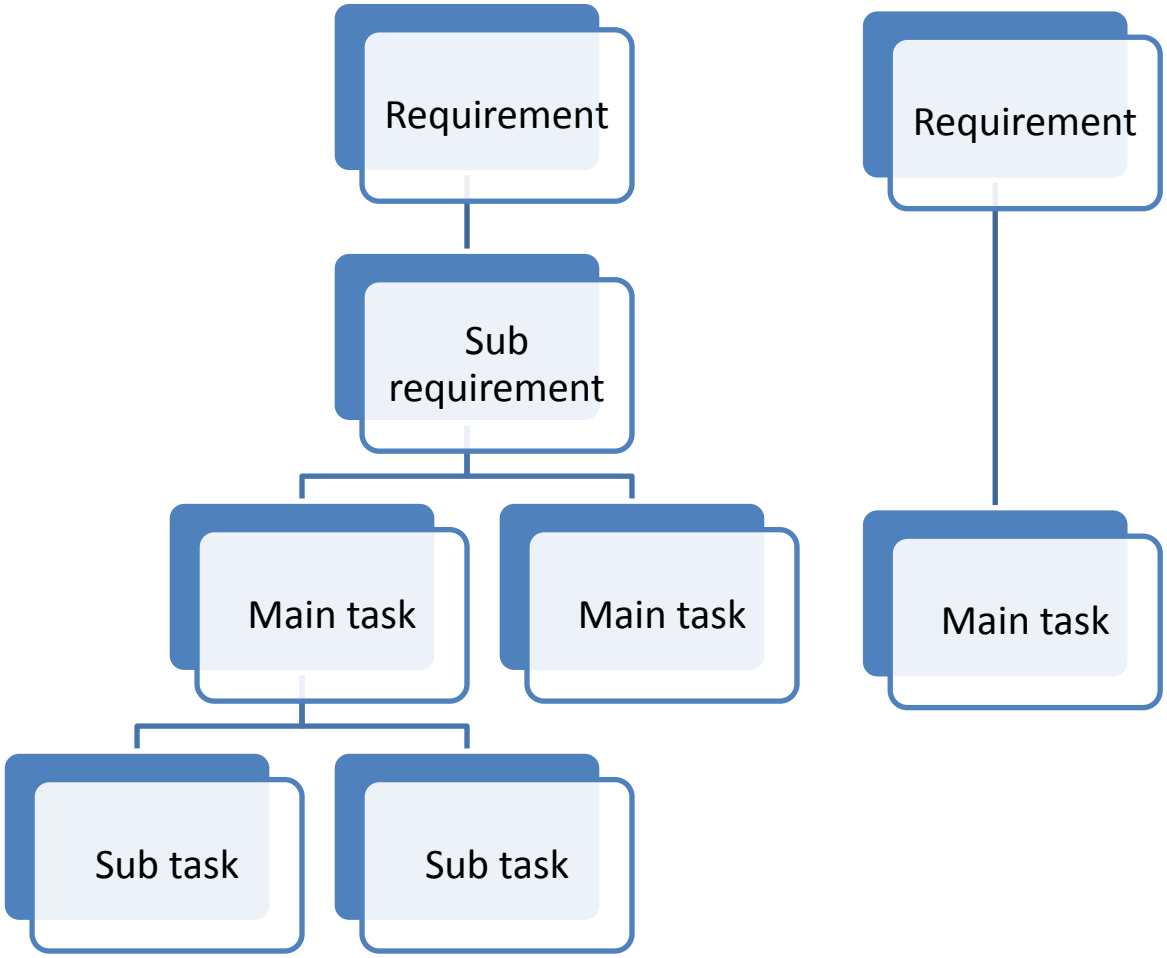
Main task/requirement & sub task/requirement

- There can be situations when a requirement or a task should or must be split into smaller chunks
- This brings some hierarchical problems to the table since Issues -tool doesn’t have this kind of built-in hierarchy handling, thus it must achieved manually

Example – R2.2

- R2.2 (There are a total of 10-15 puzzles)
  - This is a sub requirement, its parent requirement being R2
- It’s also quite obvious that there will be several main tasks related to R2.2, each depicting one puzzle
- One puzzle (main task) can further have sub tasks, like creating a needed logic, creating the UI, etc.
  - Splitting main task to sub tasks is discussed later
- This leads to a hierarchy visualized in the next slide
  - The next slide shows few possible hierarchical cases that can exist in the limits of this guideline

Hierarchy


Requirement
Sub requirement
Main task
Main task
Sub task
Sub task
Requirement
Main task

Hierarchy

- How is this achieved in GitHub?
  - With labels, references and naming conventions
- Well how the heck can this be surveyed in GitHub?
  - With a clever use of filters (more of those later)
- This is one of those aspects that GitHub could further develop to allow hierarchical views and relations
  - The only way of viewing issues is the plain list, and that makes it even more important to use labels and filters

Hierarchy

- Using correct labels is one part, the other is referencing issues from other issues
- I discuss this in the second step of creating an issue but the recommendation is as follows:
  - Every issue should refer to the **direct** parent or child issue (preferably both)
    - So a requirement can refer to sub requirements or main tasks (if there's no sub requirement between them) and a sub task can refer to main task (sub task cannot exist without a main task)
  - Since this causes a 'do this, fix that' problem (explained in the issue's creation), the minimum is that **child** issues refer to the parent issues, though I highly recommend that also parents refer to their children
  - Other issues should also be referred if there exists a viable connection between them

STEP BY STEP TUTORIAL HOW TO CREATE AN ISSUE

1. step

1. Title including possible reference number if such is used by the team (for example: (XXX-100): Do stuff or #123: Task name)
   - The naming convention should also reinforce the relation of issues. For example if main task is XXX-100, the sub task could be XXX-100-1 or XXX-100.1
   - Requirements should be named with different prefix than tasks, for example REQ 2 or R2

2. Step

2. Write description
   - Depending on how the team wants to work, either the whole description is given or possibly a link to GitHub wiki page
   - If task involves such subtasks that aren't converted to own issues, description should list them using Task list notation
   - If the issue has a close relation to another issue (like a sub task or the requirement that lead to this issue), that issue should be referenced.
     - Here the problem is that you cannot reference an issue, before it is created, thus you cannot reference a sub task from the main task if the sub task doesn't exist. There is three ways of solving this: 1) Parent tasks don't reference sub tasks, only sub tasks refer to main tasks, 2) Create first those issues that are referenced (first sub-tasks, then main tasks, sub-reqs, and so on) or 3) Update the issues afterwards.
     - I recommend the third option, since the first one is not valid for cases where you need to reference main task from main task and (if they are for example related) the second can cause kind of a recursion which breaks the work flow
   - If – for some reason – somebody who is not assigned to the task should be notified or mentioned, use the @-notation
   - Examples (also suggestions how to use the syntax) can be seen in these two issues: https://github.com/Ripppe/GraduRepo/issues/5 and https://github.com/Ripppe/GraduRepo/issues/6

Steps 3-7

3. Assign labels
4. Assign people
5. Assign milestone (if needed)
6. Create the issue
7. If the issue was supposed to be referenced from another issue, go and update that issue now!

Milestones

- Can be used two ways
  - For iteration tracking (to implement Sprints or other iterations)
  - For certain requirement or feature grouping (=grouping tasks related to the specific item)
- Milestones can be given a deadline (issues not)
- Something to remember is that an issue can belong only to one milestone!
  - Milestone is more or less like an exclusive filter for issues with a possible deadline
- The use of milestones is not required but especially in an iterative development it is highly recommended
- The creation is very straightforward. Just go to Issues -> Milestones and there you can manage the existing ones or create new ones

Issues in GitHub

- A live example: https://github.com/Ripppe/GraduRepo/issues?state=open
  - Issues are there to visualize this guideline's practices

Updating issues

- Issues can be updated two ways:
  - Either going to the issue's page and making the changes
  - …or selecting the issue from the list view and using the drop down options in the top of the list
    - This allows the update of:
      - Open/Close state
      - Labels (removing and adding)
      - Assignees
      - Milestones

An example of an issue's life cycle

- PM 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

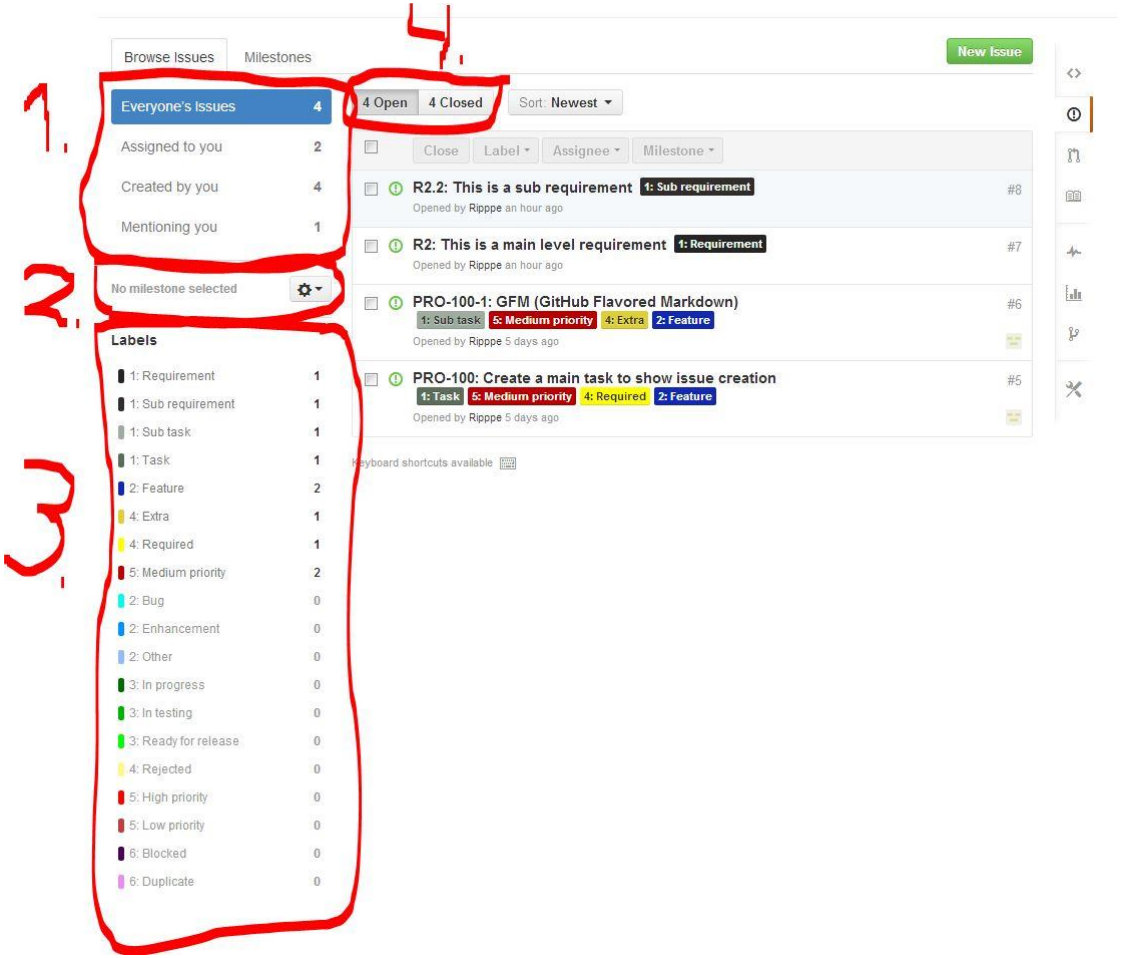
An example of an issue's life cycle

- The task is discussed in the comments and more widely in the f2f meeting
  - The description *must* be updated to reflect the outcomes!
- 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, mentions him)
- Task is ready for testing
  - Status is changed
- Y tests the task
  - Status is changed
- PM checks the task and accepts it
  - The issue is closed

How to efficiently use filters

- I cannot emphasize enough how important it is to use filters! Otherwise you will quickly be overwhelmed by the issue flood and it will most likely cause things to go foobar
- There exists four different filter groups in the Issues tool:
  - 1: General filters, only one can be selected at the time
  - 2: Milestone filter, only one can be selected
  - 3: Label filter, multiple selections allowed
  - 4: State, either open or closed ones are shown
- Filters are part of the url! So put those bookmarks of yours into an use
- Remember that there are also a sort option that can be used

Filters


1.
2.
3.
Browse Issues
Milestones
New Issue
Everyone's Issues 4
Assigned to you 2
Created by you 4
Mentioning you 1
No milestone selected
Labels
1: Requirement 1
1: Sub requirement 1
1: Sub task 1
1: Task 1
2: Feature 2
4: Extra 1
4: Required 1
5: Medium priority 2
2: Bug 0
2: Enhancement 0
2: Other 0
3: In progress 0
3: In testing 0
3: Ready for release 0
4: Rejected 0
5: High priority 0
5: Low priority 0
6: Blocked 0
6: Duplicate 0
4 Open
4 Closed
Sort: Newest
Close
Label
Assignee
Milestone
R2.2: This is a sub requirement 1: Sub requirement #8
Opened by Ripppe an hour ago
R2: This is a main level requirement 1: Requirement #7
Opened by Ripppe an hour ago
PRO-100-1: GFM (GitHub Flavored Markdown) #6
1: Sub task 5: Medium priority 4: Extra 2: Feature
Opened by Ripppe 5 days ago
PRO-100: Create a main task to show issue creation #5
1: Task 5: Medium priority 4: Required 2: Feature
Opened by Ripppe 5 days ago
Keyboard shortcuts available

Filters

- With these filters you can get quite much visibility, but of course they have their limits
- The following ones are valid filter scenarios:
  - Every task (main and sub) with low priority
  - Every task assigned to me which is marked as required
- More complex filters are invalid
  - Every main task that has a sub task
- In my experience in most of the cases, the GitHub's options for filters are sufficient enough

What kind of filters I need?

- This is a question that I cannot explicitly answer because it depends on the situation, the phase of the project and the project overall
- The following are my thoughts for programmers:
  - Should be aware of all issues that mention them + the issues that are assigned to them. Priority and milestone information is relevant.
- … and PMs:
  - PMs should watch closely all the issues being worked currently (this includes following task lists and how milestone deadlines are met)
  - Blocked issues must be responded without a delay
  - Bug reports and enhancement proposals should be monitored

Things to note about issues

- Status & miscellaneous labels can be used by anyone (and this is strongly encouraged). The rest should be left to the one responsible of the RM (like PMs)
- In the optimistic case, all the talk related to the certain issue (not including f2f talk) is in the issue's comments.
  - This can rarely be achieved, so my recommend is that if discussion regarding an issue occurs somewhere else (emails, chats, **f2f**, etc.), the issue's description is updated appropriately
    - This is also the case if the discussion in comments leads to updates!
    - The bottom line is that issue's description ***must*** always be up to date. This is imperative.

Things to note about issues

- Task lists should be used only in the issue's description
  - It's good to note that these lists are there only for cosmetic purposes. This means that GitHub itself doesn't mind if you close an issue with unchecked task list items. However, if an issue is referenced, the task lists state will be visible in the reference comment.
- When programmers make commits, they should always reference the issue related to commit (if such exists) in the commit comments.
- Programmers should take an active role for using issues. If nothing else, three things should be required: 1) Referencing the issue in commits, 2) updating labels accordingly and 3) updating task list
- Like keeping description up to date, it is imperative that the labels of each issue are also up to date! Those are the best and fastest way of knowing how things are going.
  - Think it like this: description tells what the issue is all about (what should be done to complete it) and labels visualize the different states of issues

Things to note about issues

- Depending on how the development process is planned to go (for example, is testing an issue wanted before closing it?) affects whether issues should be closed from commit messages:
  - If testing should occur first (like it should!), it is best that closing issues is done from GitHub itself
  - Otherwise commit comments can be used for closing (GitHub's special syntax for this)
- My suggestion is that the same person who does the implementation is not authorized to close the issue (thus someone else – like PM – must do it)
  - This means that issues shouldn't be closed from commits but manually from GitHub
  - IF we are precise, this doesn't belong to the RM, BUT generally it is a good idea, and it enhances the visibility (other developers or PMs are 'forced' to see what you have done)

Things to note about issues

- For my recommendation of syntax for issues, see the open example issues in https://github.com/Ripppe/GraduRepo/issues?state=open (especially the task issues)
- Requirements should only have the type and status labels and – if needed – misc labels
  - They could also have the priority label but it is more efficient to give it straight to tasks since they are the ones that people mainly work with
  - So if you want to emphasize certain requirement, give the tasks related to it higher priority. Of course priority label can be used as a remainder in a requirement issue

Creating a bug report

- A bug report is a special issue which is usually done when someone spots an error in an implemented feature
- Creating a bug report within this guidelines limits is quite simple, here are the steps:
  - Locate the main task (feature) to which the bug relates to (I discuss this more on the next slide!)
  - Create a new issue (use the aforementioned steps) and add a ’bug’ label

Creating a bug report

- Bug report can be done by anybody
- This causes an inconvenience since it may not be clear to the creator to which task the bug relates to. Requiring the creator to find out that may not be a good option
- Thus I suggest that if it is not clear for the creator, the bug can be done ‘outside’ the normal issue hierarchy.
  - However, the one responsible of issues should find out (since he should have the knowledge) the related task and reference it
  - It’s also a good practice that if the one who creates the issue doesn’t assign it to himself, the assignment is left pending the PMs decision (the creator should rather mention a PM as to notify him about the bug)

Creating a bug report

- Of course this is not the only way to handle bugs
- Other could be to use bug label as a flag
  - If during the testing a bug is found, the bug is commented to the issue's comments and bug 'flag' is given
  - However, this might obstruct the way how bugs are controlled (they are kind of lost in comments, so if we want to know how many bugs there has been, every issue must be gone through to find them)
- Bug issues can also be given a own prefix to better distinct them. Nevertheless, they should still refer to an issue

Creating an enhancement proposal

- Enhancement proposals are used quite sparingly, but they could be utilized with a greater efficiency
- Enhancement proposal is a suggestion relating to a feature or a requirement that (in most cases) comes from such stakeholders which are not authorized to make the decision by themselves
- These suggestions often consist of a thoughts related to improving certain aspects or functionalities
  - For example, it may have been specified with the customer that feature X will be done so and so. A programmer from the team has a better idea, and thus he makes an enhancement proposal

Creating an enhancement proposal

- For the guideline, an enhancement proposal is a main task or a sub task and relates to either a requirement or another main task. This depends about the nature of the enhancement.
- The creation is done with the normal steps with a ’Enhancement’ label
  - Making an enhancement proposal doesn’t mean that it will be implemented. It is a proposal and should be discussed with the appropriate stakeholders whether it is accepted or not.
  - If accepted, the description may need improving, but otherwise it is handled as a task
  - If rejected, it should receive a ’Rejected’ label and be closed

About wiki

- There's no reason why GitHub's wiki shouldn't be the major information platform
- However, I'll only discuss Wiki's usage related to the RM
- As mentioned below with the issues, it's not clear how much information should be written to the issue's description.
  - The bottom line is that by reading only the description a programmer should be able to tell what is wanted
  - My own experience is that for every task many, many small details exist, it's the matter of has anybody considered them that concerns us. Everything cannot be specified without some waste of time, but on the other hand, too vague specs will also waste time (because programmer either needs to figure out the missing stuff by himself or consult somebody). Of course in some cases programmer is given free hands to implement the feature within some borderlines. Vague specs may be accepted in such cases.
- Overall the golden middle way can only be achieved by trial and error
  - Check this example issue and its description: https://github.com/Ripppe/GraduRepo/issues/5

About wiki

- My overall suggestions for the wiki usage are as follows:
  - Every document should be linked to Wiki, for example a ”collection” page could be created to store the link to all the external resources
  - Major project related documents should be made to the wiki (as an own wiki page or as a link). GitHub’s wiki is so sophisticated (checkout the markup!) that every basic text handling activity can be achieved
    - Decide what markup notation you use in your wiki and stick with it – on all pages!
  - A collective page consisting of every preplanned requirement, should be established. The form of this page is free, but it should list shortly every requirement.
  - Info of a requirement in a wiki and in an issue must not conflict with each other!

WHERE SHOULD I START?

Steps I recommend

1. **Get familiar with the GitHub and especially Issues tool before taking this guideline into a practice**
   - This is imperative! Play around, make dummy labels, make issues, try updating them. Checkout the GFM syntax.
   - I strongly encourage creating an own repository for you to play with, because once an issue is created, it cannot be deleted (and we don't want the playground stuff getting into the project's repo)
   - You should at least be comfortable with the basics of GitHub before going forward

Steps I recommend

2. Read this hefty manual if you haven't already done so
3. Decide the label categories to use
4. Decide the label color schemes
5. Decide and create the labels themselves
   - This includes the prefix and naming
6. Create issues for the requirements
7. Decide the initial tasks (including sub tasks and task lists) and create them
8. Decide and create milestones if needed
9. Make a collection page to the wiki (usually this is the Homepage with links to the other wiki pages)
10. Start using the guideline!

Finally

- This is still a concept guideline (and presentation) and it will most likely be updated in the future
- The feedback is very much welcomed, my contact information can be found from the start of this presentation
  - The best way of contacting me is via email of Facebook
- Don’t be afraid to ask questions, I’m more than happy to answer and explain myself better!