A proposal to use Github as PDM environment for a vision based sand analyzer

Jelle Spijker[1]* 495653

Abstract
During the development of a vision based sand analyzer or any other startup like product, it's important to manage the flow of information, in such a way that redundancy is limited and continuity of the product(range) is guarded. This is normally done within a product data management environment (PDM). These environments normally consists of dedicated and expensive software which is supported with complex network architecture. In this article Github is explored as a free alternative for such an environment. Github is an web-based git repository service mostly used during software development. It is found that this service can be used as a full fledge free alternative PDM environment, with the use of free opensource utilities. Thus allowing start up companies to implement PDM like options.

Keywords
Product Data Management — Vision Soil Analyzer — Github

[1] *Department of Engineering, HAN University of Applied Sciences, Arnhem, the Netherlands*
***Corresponding author**: spijker.jelle@gmail.com

Contents

Introduction

This article proposes a setup for a PDM[1] environment used during developed of a vision based sand analyzer[2] which analyzes sand by its optical characteristics. This new line of product(s) is currently being developed at IHC MTI B.V. which is a R&D based company and subsidiary of Royal IHC

The concept and initial design of this product finds it origin at the minor embedded vision design and is developed further during the graduation phase mechanical engineering , both taught at the HAN University of Applied Sciences. Design and production of this product leans on the following three disciplines: mechanical, electrical and software engineering. Due to its multidisciplinary properties it is relevant to implement a PDM environment which acts as a solid basis for all disciplines and allows for interaction and flexibility.

The initial focus of this product were vision algorithms written in C++, it therefore started as a software repository on Github, which is a web-based Git[3] repository. Due to the changing and growing demands of this project, the focus shifted towards other engineering disciplines like electrical and mechanical. Because Git is mostly used in the development of software it is largely unknown in the mechanical engineering branch. But it is gaining traction as an PDM environment according OLEG[5].

Github is strongly rooted in software development, it therefore has a strong toolbox supporting it. This article examines the use of third party opensource software and a supporting workflow which will ensure PDM like support for the none native disciplines: electrical and mechanical. This environment will also put safeguards in-place to ensure protection of IP[4] but still allows multiple users/parties to work on sub-projects. It does so by defining different user roles and set restrictions

[1]Product Data Management
[2]Working title VSA

[3]version control system for software development designed and developed by Linus Torvalds and used for the development of the Linux Kernels
[4]Intellectual Property

to sub projects.

This article illustrate how to use Github as a PDM tool during the development of a vision based sand analyzer or any startup like product. It does so by describing the normal workflow employed by Github. It will then describe the current product and give a foresight in to certain possible future developments. Afterwards it sets out the needed adjustments with respect to the normal workflow in order to support a whole startup like project.

This article has the following document structure: Github is described in section 1 following with a product breakdown of the vision based sand analyzer in section 2. Section 3 describes a normal PDM environment, this is used as matrix against the current Github environment. Hereafter, in section 4 a model is given how to make the translation to a full fledge PDM environment. Lastly in section 5 the pros and cons are discussed.

1. Github

Github is a web-based Git repository hosting service with SCM[5] and distributed revision control capabilities. It also provides access control and multiple collaboration features, such as bug tracking, feature request, task management and wikis. Github has a strong influence in the open-source community. This is because standard ”free” services doesn’t allow for private repositories. They’re accessible by anyone and allow everyone to view, copy and use them. A lot of company, governments and institutions such as NASA, CERN, Google and Netflix do so under the credo; Dare to share. For those whom want to protect their intellect property they offer paid accounts which allow for private repositories. It is also possible to host your own service using GitLab

Recent developments show a move from strictly software support to other disciplines, to cite Peter Bell and Brent Beer [6] While GitHub is still primarily used to collaborate on the development of software, it’s also a great way for a team to collaborate on a wide range of projects. From the authoring of books and the distribution of models for 3D printing to the crafting of legislation, whenever you have a team of people collaborating on a collection of documents, you should consider using Github to manage the process.

Github is based on the Git workflow, which was developed by Linus Torvald to help with the development of the Linux kernels. Git is a strict command-line tool which emphasizes on speed and data integrity, it allows for a distributed, non-linear workflow. This workflow is explained in the next section.

1.1 The normal Github workflow

As mentioned in the previous sections Github is Git-based and works with repository. These can roughly be described as directory structures with revision control. This workflow is depicted in figure 1 and can be described as follows: A new repository is created on the remote server, which is then pulled to a local host.

[5]Source Code Management

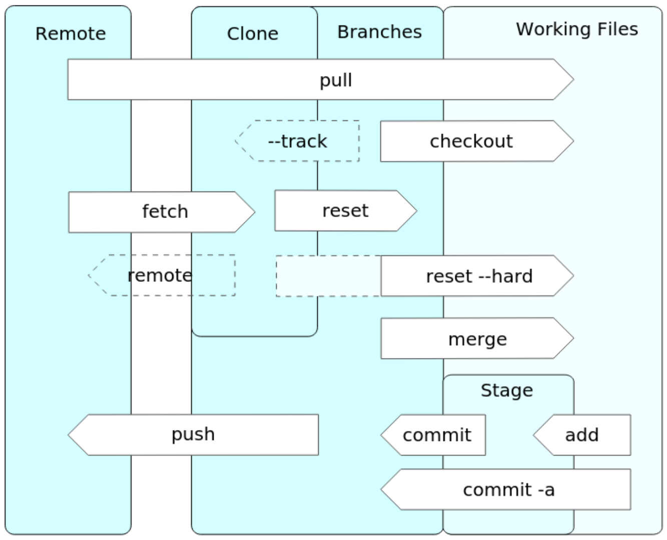


Figure 1. Git workflow. Source: [2]

A user can make changes in this directory by adding, deleting or modifying files and directories. Each alteration such as the addition of a few lines of code, which represent a new function are added to the staging area. These alterations are then committed to the current branch, which will be discussed further on. These commits are then pushed to the remote server, where they’re stored with meta-data such as comment, blame (who applied the change), and a date. Each commit only stores the change in the file and allows a user to review against previous commits (versions).

If another developer has committed new work, the developer first has to pull these changes to his local host before he is allowed to push his own changes. Git will automatically detect any conflicts and show them to the developer. He will then be allowed to solve these conflicts if needed.

As mentioned before the commits are first stored in a branch. These branches allow for distributive workflow. They do so be running parallel with the master branch. If a new feature is being developed, it is common practice to create a new branch, implement the new features, which probably consist of multiple commits and pushes to the remote server. Which allow other developers to work on the same feature as needed. All the while changes are still allowed on the master or any other branch. Once the feature is ready to be added to the master or an other branch, a pull request is made to the community.

This will compare the current branch against the other branch and allow for other developers to discus and review the code. If they find it satisfactory and they don’t foresee any major problems the pull request is granted and the branch is merged. This process is shown in figure 2.

The figure 2 also shows the use of tags. It is possible for a developer to tag a certain commit, in essence freezing the state of the project. This is mostly done when versions are

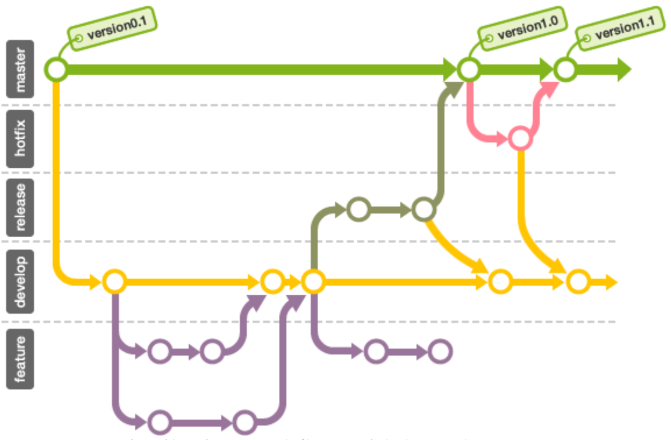


Figure 2. Distributive workflow with branches. Source: Backlog[1]

released to the public or classification agencies. These tags allow the state of the project to be recalled at that time.

1.2 Issues

Further active collaboration is supported and encouraged using Issues. These are generic items which could be idea's, task, bug reports or questions for instance, which can be submitted by a developer. An issue is created by describing it, illustrating it with images, links or examples. Assigning a label to it such as ToDo, bug, feature enhancement. Other developers can comment on these issues so as to give it direction. These issues can be assigned to individual developers and milestones. Which allows them to focus on further goals.

1.3 Project wiki

Each project has its own wiki[6] It's expected from the developers to maintain an active wiki. All knowledge regarding the project can be gathered here, shared and improved.

2. Vision based Sand Analyzer

The vision based sand analyzer is a product that analyzes sand sample using a digital camera sensor. It does so by taking a snapshot of a magnified soil sample and applying multiple software algorithms on an obtained digital image. A detailed and full description of the first two prototypes are described in the product documentations of the previous build prototypes [3] and [8], where the last iteration serves as the basis for the product breakdown structure depicted in the next subsection.

2.1 Product breakdown structure VSA

The product breakdown structure, depicted in figure 3 differentiates between three structures: hardware, software and a casing. This demarcation is made to better serve the different disciplines. This breakdown is purely done on individual parts, where each part serves a function.

[6]A wiki is a website which allow sharing of knowledge by collaborative modifications

Due to the scope of this document these functions are not specified. The software functionality is covered in detail in the Doxygen documentation. It is important to note that the software is developed with object orientated language C++ and is divided in different namespaces. Which are depicted in figure 3. Each of these namespaces is a bundle of object each serving its own function and by inheritance it's possible to create complex objects. Integration of the software development environment in Github is seamless.

Whilst function which are fulfilled by the casing and hardware are describes in the product documentations [3]. The individual parts serve as actuators, sensors and interfaces with the outside world.

2.2 The toolboxes

Each discipline uses it's own type of tools, it is therefore relevant to asses all used tools and make sure the are compatible with each other. Each of the tools below are stand-alone versions which run on a operating system, they store their data in files, which in turn can be seen as the output. At the basis of all three development environments lies the hardware. The specifications given below, describe the current development computer. It is guaranteed that the project can be recreated with a similar computer.

- Intel(R) Core(TM) i5-4210U CPU @ 1.70GHz
- 8gb memory
- Nvidia 820M
- SSD 128 gb
- HDD 500 gb
- Dual boot with Kubuntu 15.10 / Windows 10

Software engineering The VSA currently runs on a dedicated embedded Linux environment and is programmed in C++. The IDE[7] consist of a Linux Debian environment running Qt Creator, testing is done with Valgrind and code compiling is done with the GNU GCC compiler or LLVM Clang. The basic list of used libraries is given below.

- Standard C++ Library
- OpenCV 3.0 beta
- CUDA 7.0 SDK
- ZLib
- Boost 1.58
- Video4Linux
- GStreamer

All these packages are chosen such that the can easily be ported to other operating systems, such as Windows, Android or iOS. Although current prototypes are dedicated embedded devices, the ultimate goal is to create a cheap and portable device. In order to achieve that, it is important to make use of existing and readily available architecture such as phones and laptop. Software development has to be done with this foresight; Currently all these tools allow for this strategy.

[7]Integrated Development Environment

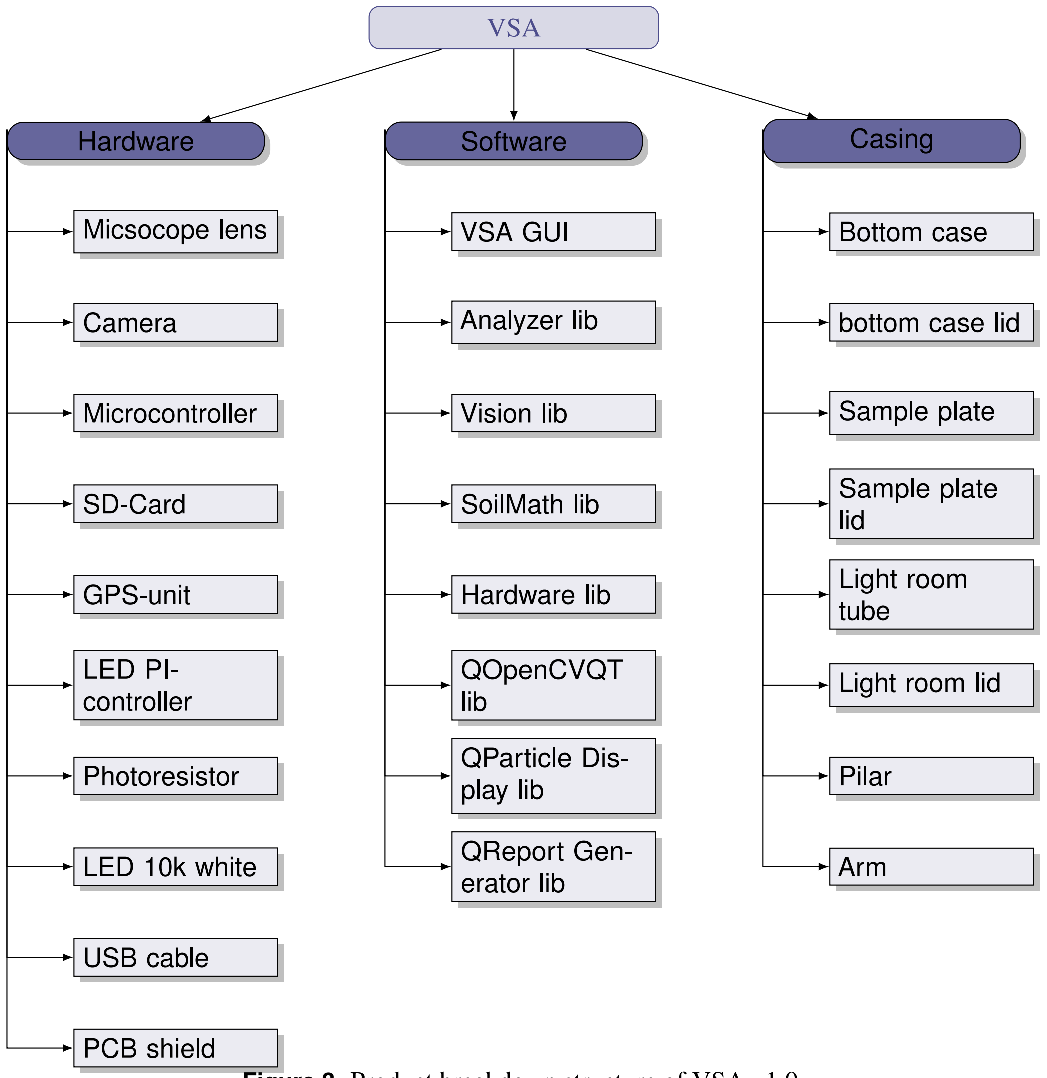


Figure 3. Product breakdown structure of VSA v1.0

The output of the software engineering toolbox will be mostly source code and binary files. The source code files are plain text format, which can be read and edited from most devices. While the binary files will consist of images and other additional resources needed by the program. This actual program needs to be compiled for each individual target system. This is done with each release and the files are bundled with the version tag.

Future release may save their data in databases, it is possible that these database run on the local host, but they could also run on remote host. Databases need special care to put under Git revision control. The best practice is to dump the database and schema and put the output – which is a plain text file – under git control.

Electrical engineering The actuators and sensors, such as the GPS-unit, LEDs and photoresistor are all driven from an embedded Linux device, The Beaglbone Black. This device has the capability to run an OS and can interact with the outside world by pinmuxing[8]. The PCB[9] and circuits are designed with the Circuit Design Suite from National Instruments.

This suite consist of the programs Multisim for circuit design & simulation and Ultiboard for PCB design. Both programs only run in a Windows environment. Multisim works with a binary file with the extension **.ms14*, but it also has an option to output SPICE[10]. Which are plain text files, and can be used with a multitude of different simulation programs. Ultiboard works with a binary file that has the extension **.ewprj*. It can export in the Gerber format, which is a plain text file and the de facto industry standard used with PCB.

Mechanical engineering Each part has a mechanical representation. These parts are modeled in Siemens NX 10, which

[8] Changing the state and function of pins on the CPU, such that it allows for different in- output protocols

[9] Printed Circuit Board

[10] Simulation Program with Integrated Circuit Emphasis

offers a rich interface for mechanical engineers. They allow for parametric modeling and complex FEM and CFD simulations. Siemens NX output all its file in binary formats. These files aren't backward compatible.

Github recently adopted a new viewer for 3D-models. It can now read and compare previous versions of **.stl* files, as shown in figure 4. These file are easily exported with NX and are used for CAM[11] such as CNC[12], 3D-printing or any other rapid-prototyping technique.

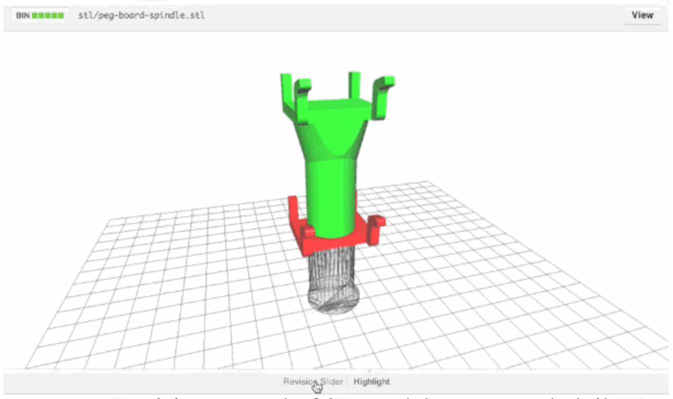


Figure 4. Revision control of 3D models, source: skalnik[7]

Simulations and calculations are done with Matlab 2015a, which has a Git integration. Most Matlab files are plain text files, but special care should be taken with the Simulink models. According to the Matlab website it is necessary to prevent Git from corrupting Simulink models, this is done by editing the *.gitattributes* file and adding the following lines:

```
*.slx -crlf -diff -merge
*.mdl -crlf -diff -merge
*.mat -crlf -diff -merge
*.mlx -crlf -diff -merge
```

Documentation All documentation is written with LaTeX, which is a word processor and a mark-up language. Documents are written in plain text files with macro command in between, which – when compiled with the LaTeXcompiler generate typeset documents. These output documents are stored as **.pdf* files. This document is an example of such an output file.

2.3 Interfaces

Each part has an interfaces, which describe how it interacts with other parts. This project defines interfaces in four domains; These are: mechanical, electrical, software and corporate. The last one represent the interface with (after-)sales and supply chain. Not all domains are represented for each part, but a part should have at least one interface, otherwise it is redundant.

Each part in the project has a fully defined interface. Due to the limited scope of this article only the interface used with the LED Luxeon Rebel is used as an example. This is shown in figure 5.

[11]Computer Aided Manufacturing
[12]Computer Numerical Control

3. What is a PDM environment

In order to truly understand a PDM environment, one has to start with the main definition, which is according to La, Hoogeboom, and Konst [4] a strategy which puts the right product and process related information in the right hands at the right time in a product life cycle. It also allows for the creation of processes which guard those data en allows other to use it. All the while generating en secure environment, at which all relevant information, and previous iterations is stored without redundancy.

Github is without a doubt a great tool when used as revision control it save all relevant information and stores it in a secure environment without redundancy. It has potential to be used as a full fledge PDM environment. But in order to maximize Github for this setup it is relevant to look at the mechanics of such an environment a little closer, so that it becomes clear what should be added or changed.

Data vault Each data item should be stored in one single location, where only authorized developers can access that data at a time when its needed. In order to support this system, roles should be defined. Each developer needs to be assigned one or more roles, which allows him access the for that role relevant data. The role should also specify if he has read or modification rights.

Metadata One of the main features of a PDM environment is a possibility to quickly look-up metadata[13], which comes in two flavors: structural and descriptive. The structural metadata tells something about the container e.g. a file, whilst descriptive meta-data gives a user information with regards to the data content. Searching for this data should be intuitive and simple; Using elementary queries.

Classification Classification of the files is important because not every file uses the same metadata fields. Classification can be done with the use of (sub)groups, narrowing the possibilities and making sure only the relevant items turn up in a search. It is advised to also create a classification system per functionality.

Check-in / -out Developers should not disturb each others workflow. They check-out their work, modify it, and check it in again; Accompanied with a small description of the changes.

Bill of materials A complete list of all materials should be easily obtained for each product, preferably from different role perspectives.

Configuration management Configuration management is registering and administrating all kinds of information, during the development process. It is central that certain states of the

[13]Data about data

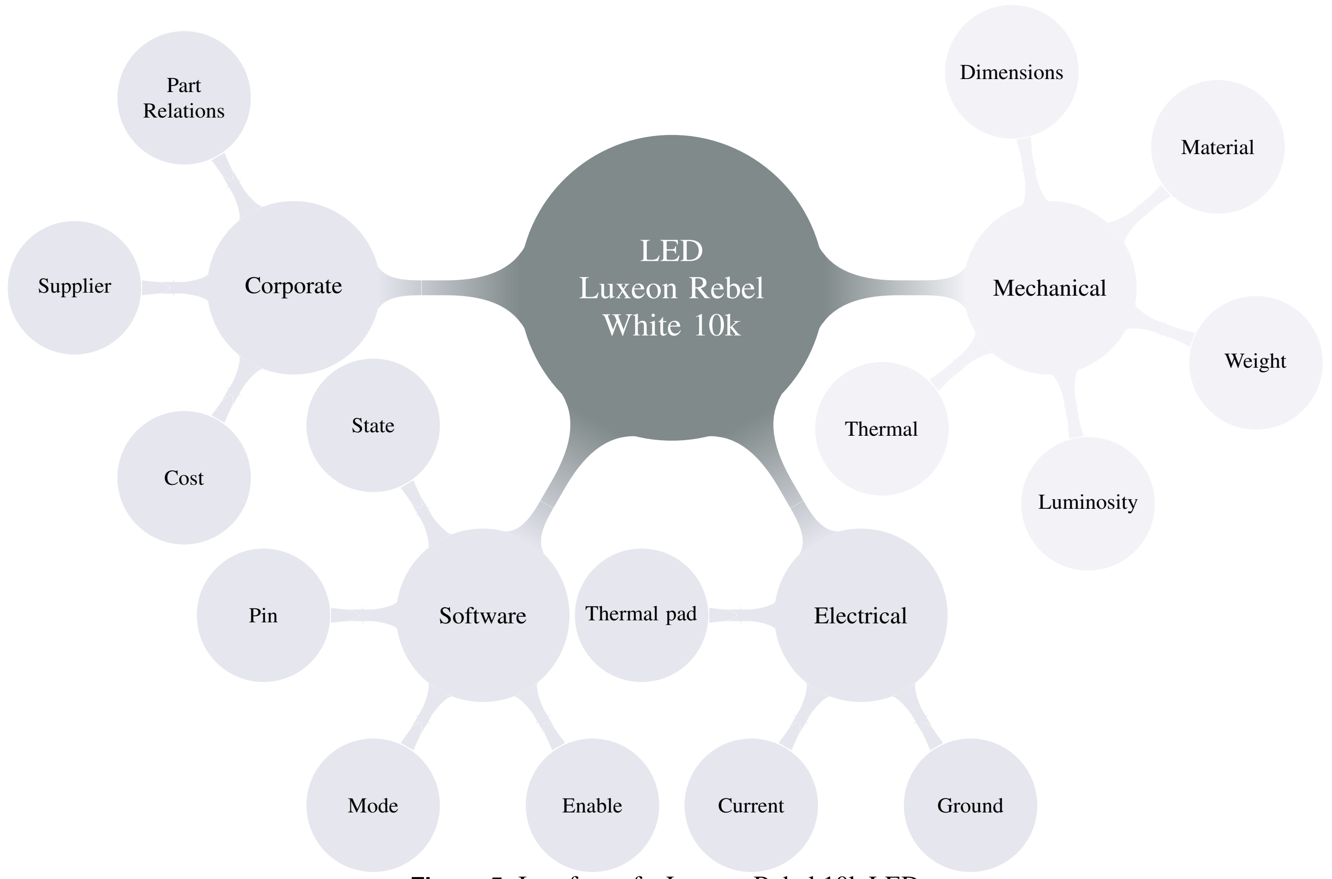


Figure 5. Interface of a Luxeon Rebel 10k LED

development process can be obtained when needed; So called snapshots.

Design review During the design process it is paramount that certain choices, comments and agreements are saved and can be accessed at any time. This helps when there is a need to review certain parts of the design.

Data visualization Not every user has all development programs accessible, but it could be possible that he needs information

4. Implement Github as PDM environment

When the traditional services of Github are set against the requirements for a PDM environment, certain deviations become apparent. This missing functionality can be implemented using free third-party opensource solutions such as TMSU. A utility which allows files to be tagged with meta data.

4.1 Meta tagging with TMSU

The tagged files can be accessed through a virtual mounted file system. Which allows the use of queries and lets them browse through the obtained results as if it where directories. This allows maximum compatibility with other software. TMSU also has the option to assign a value to meta tag. It's therefore possible to create a tag "Length" with a value for that a specific part. This allows for configuration management, such as the bash[14] print out below illustrates. TMSU is at is core a Linux program and runs on Debian based distributions. It can be compiled for Windows but without the option of a virtual mounted file system.

The use of TMSU tags and queries is illustrated below. In this example all metric $8[mm]$ parts are shown. With the first command all parts are shown, but the second query only shows the part that tests true for all values. The workings of TMSU are further illustrated in a screencast.

```
$ tmsu files "m=8"
./3Dmodel/parts_pdm/bolt_m8_50_316L
./3Dmodel/parts_pdm/bolt_m8_50_s235jr
./3Dmodel/parts_pdm/bolt_m8_60_316L
./3Dmodel/parts_pdm/bolt_m8_60_s235jr
./3Dmodel/parts_pdm/bolt_m8_70_316L
./3Dmodel/parts_pdm/bolt_m8_70_s235jr
./3Dmodel/parts_pdm/washer_m8_316L
./3Dmodel/parts_pdm/washer_m8_s235jr
./3Dmodel/parts_pdm/nut_m8_316L
./3Dmodel/parts_pdm/nut_m8_316L

$ tmsu files "m=8_and_material=s235jr_and_length=50"
./3Dmodel/parts_pdm/bolt_m8_50_s235jr
```

Figure 6. TMSU query on the command line

[14] linux console output

Best practice is to define all four domain interfaces (mechanical, electrical, software and corporate) as valued meta tags. This way all relevant interfaces can be found with a simple query. Each defined value in a domain interface, illustrates a meta tag in its own right. This assures compatibility with other parts. Since the structure for each file is defined. This will ensures easy linking and matching of individual parts.

By extending Github with TMSU it is possible to use classification, create a bill of materials, use configuration management and define inter-parts relations. Thus it becomes a full fledge PDM. But it's a fragile connection. Much depends on the naming of meta tags and the users work flow. The following workflow has proven to work.

4.2 Binary workflow new feature

In order to minimize the disturbance of the main product, new features are added with branches. As explained before in section 1.1 a branch is just branch; It is a offshoot of the master branch at a certain point such that modifications and changes are implemented. This branch can be merged at a later point with the master branch.

Although Github is steadily implementing certain features for binary files, such as **.stl* support it's still best to work with a different workflow, when working with binary files. This is because the difference can't be ascertained by Github. These need to manually determined by the developers. This process is shown in appendix A.

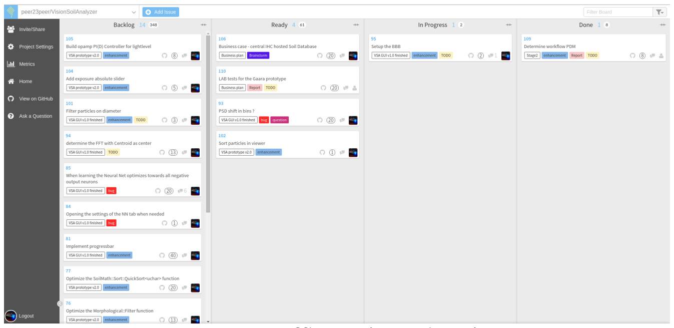
This is when the corporate interface domain comes in handy. Each file that is connected to another needs to be linked in the relations interface. This meta tag needs to be updated by the developer. That way a simple query can show the impact certain changes have on the whole design and which developers need to be notified in the suggested change. By opening up an issue, and assigning the work to the responsible developers. The dialog between them ensures an compatible new feature.

4.3 Project planning

The use of issues and milestones enable a fluent project planning which is up-to-date with the latest workload. In order to distribute this load it is possible to use the third party service Waffle, which integrates completely with Github. It shows the status of all issue, and present four column, namely: Backlog, Ready, In Progress and Done. All open issue start in the Backlog, when preparations are made to start working on them to can be moved to the Ready column. As soon as work is commenced the move to the In Progress column, when they're finished the move towards the Done column. The throughput of the project can be monitored closely allowing to actively steer on optimal team performance.

4.4 Protection of intellectual property

Information is a power unto itself, each piece needs to be assessed; What impact it has when not solely in our control – different parties have other interests – not always aligned with the interest of the original creator. It is therefore necessary to control information access.



Backlog 14 348

105 Build opamp PI(D) Controller for lightlevel — VSA prototype v2.0, enhancement — 8

104 Add exposure absolute slider — VSA prototype v2.0, enhancement — 5

101 Filter particles on diameter — VSA GUI v1.0 finished, enhancement, TODO — 3

94 determine the FFT with Centroid as center — VSA GUI v1.0 finished, TODO — 13

85 When learning the Neural Net optimizes towards all negative output neurons — VSA GUI v1.0 finished, bug — 20 — 6

84 Opening the settings of the NN tab when needed — VSA GUI v1.0 finished, bug — 1

81 Implement progressbar — VSA GUI v1.0 finished, enhancement — 40

77 Optimize the SoilMath::Sort::QuickSort<uchar> function — VSA prototype v2.0, enhancement — 20

76 Optimize the Morphological::Filter function — VSA prototype v2.0, enhancement — 13

Ready 4 61

106 Business case - central IHC hosted Soil Database — Business plan, Brainstorm — 20

110 LAB tests for the Gaara prototype — Business plan, Report, TODO — 20

93 PSD shift in bins ? — VSA GUI v1.0 finished, bug, question — 20

102 Sort particles in viewer — VSA prototype v2.0, enhancement — 1

In Progress 1 2

95 Setup the BBB — VSA GUI v1.0 finished, enhancement, TODO — 2 — 1

Done 1 8

109 Determine workflow PDM — Stage2, enhancement, Report, TODO — 8

Figure 7. Waffle project planning

Normally each developer, with access to a repository, can work on all files. When it's needed to have different access levels in a PDM environment it's advised to work with embedded repositories. Each repository can have it's own developers, only these can work on the files herein. These separate repositories are integrated in the whole project, whilst compartmentalizing the information. This setup is handy when third-parties need to work on parts of the project, but these parties are not part of the company.

5. Results and Discussion

This article explorers the possibility to use Github as a full fledge PDM environment. It does so by first describing a normal Github workflow, following with the expected out come – the description of an traditional PDM environment. These are set against each other and a workflow with additional tools is proposed to alleviate the difference.

It is found that Github can serve as a free alternative PDM environment; But it should be recognized that it is not a natural transition. Github is original a software development environment, it excels at handling plain text files, revision control and parallel feature implementation. This allows an efficient workflow. But it is still lacks a lot of features when handling binary files.

These limitations are alleviated with the use of third party opensource software such as TMSU and waffle. Which allow for project planning, meta tagging, search functionality and change management.

Although it is possible to mimic a PDM environment with Github it is important to note that it takes a certain type of users that can work comfortably in this environment; These users should feel at home working from the Linux command line. In other words, it requires nerds.

References

[1] Backlog. *Branching workflow using topic and integration branch [branch] | Git Beginner's Guide for Dummies | Backlog*. 2015. URL: `http://backlogtool.com/git-guide/en/stepup/stepup1_5.html` (visited on 11/21/2015) (cited on page 3).

[2] *Git (software)*. In: *Wikipedia, the free encyclopedia*. Page Version ID: 691077698. Nov. 17, 2015. URL: `https://en.wikipedia.org/w/index.php?title=Git_(software)&oldid=691077698` (visited on 11/21/2015) (cited on page 2).

[3] Jelle Spijker. *Vision Soil Analyzer: Product design of a vision based soil analyzer*. Oct. 10, 2015 (cited on page 3).

[4] Fontaine J. P. La, M. G. R. Hoogeboom, and J. S. Konst. *Product Data Management: A Strategic Perspective*. Geldermalsen: Maj Engineering Publishing, Aug. 24, 2009. 103 pages. ISBN: 978-90-79182-06-0 (cited on page 5).

[5] OLEG. *GitHub PDM: Is It For Real?* Beyond PLM (Product Lifecycle Management) Blog. Sept. 19, 2013. URL: `http://beyondplm.com/2013/09/19/github-pdm-is-it-for-real/` (visited on 11/18/2015) (cited on page 1).

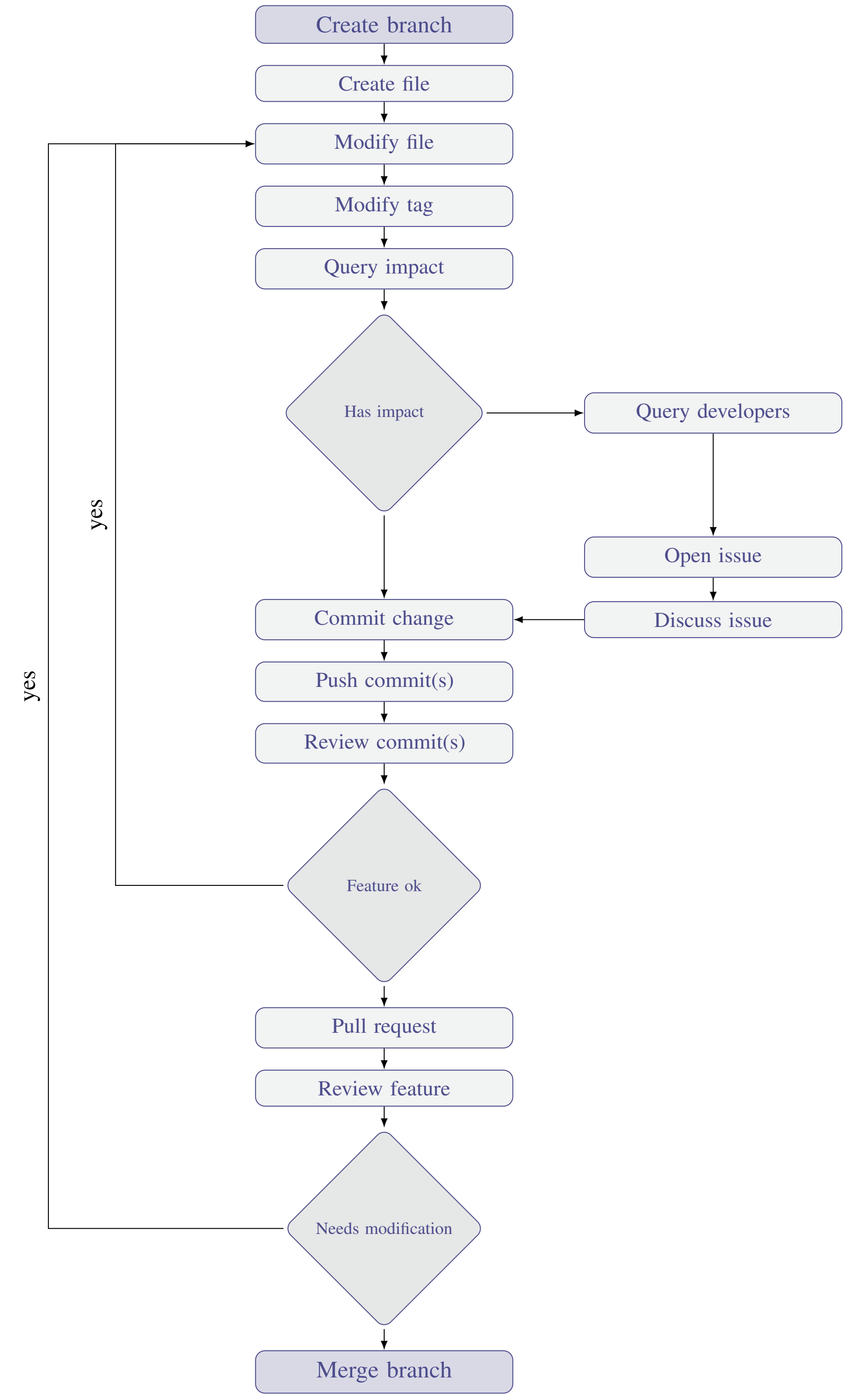
[6] Peter Bell and Brent Beer. *Introducing GitHub: a non-technical guide*. O'Reilly, 2015 (cited on page 2).

[7] skalnik. *3D File Diffs*. GitHub. Aug. 17, 2013. URL: `https://github.com/blog/1633-3d-file-diffs` (visited on 11/18/2015) (cited on page 5).

[8] Jelle Spijker. *Product documentatie: Test opstelling - Soil Analyzer*. Nov. 2, 2014 (cited on page 3).

Appendices

A Workflow add binary feature


Create branch
Create file
Modify file
Modify tag
Query impact
Has impact
Query developers
Open issue
Discuss issue
Commit change
Push commit(s)
Review commit(s)
Feature ok
yes
Pull request
Review feature
Needs modification
yes
Merge branch