



POLITECNICO DI BARI

DIPARTIMENTO DI INGEGNERIA ELETTRICA E
DELL'INFORMAZIONE

CORSO DI LAUREA MAGISTRALE IN INGEGNERIA
INFORMATICA

Thesis in

ADVANCED SOFTWARE ENGINEERING

Software sustainability and maintenance in industries

Supervisor:

Prof.ssa **Marina MONGIELLO**

Co-supervisor:

Ing. **Francesco NOCERA**

Prof.re **Carlos CARRILLO**

Prof.re **Rafael CAPILLA**

Author:

Michele SCARIMBOLO

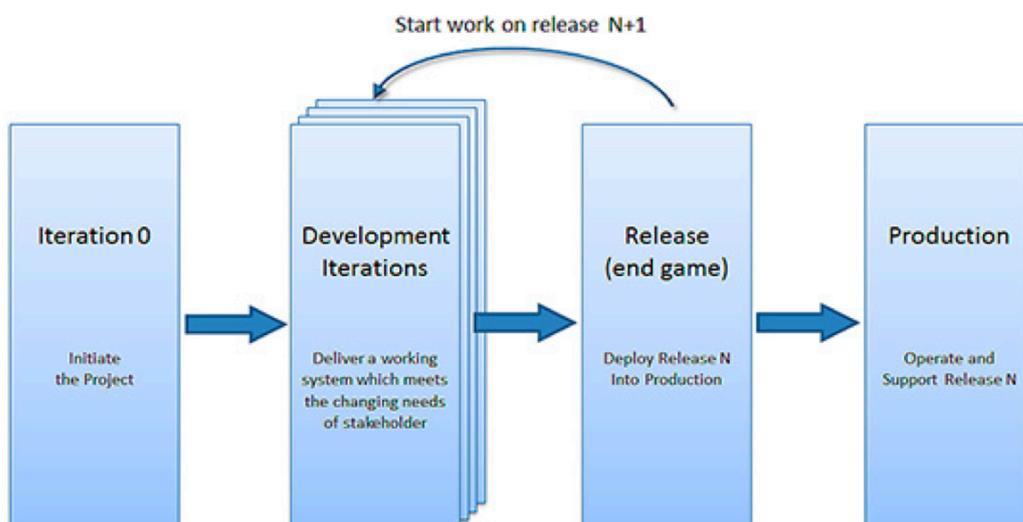
Academic Year 2018/2019

Index

I. Introduction	5
Software sustainability	7
Sustainable architectures.....	9
Long-Lived decisions	10
Kanban software development	13
Git systems.....	16
Branching and merging.....	16
Small and fast	18
Continuous software engineering.....	21
Continuous Integration.....	22
Overview and detail CI process	23
Continuous Deployment.....	28
II. Software developed	31
Functionalities	32
Home page.....	32
Creation of project	33
Project page	34
Design decision page.....	38
Code: main artifacts	42
Example	42
Model View Presenter (MVP).....	44
Benchmarking	49
Trello	49
Comparison.....	50
Asana	52
Comparison.....	53
Asana vs Trello	56
III. Results and future developments	57
Continuous engineering in our software	58
Using GitLab for CI/CD	59
Git and social developments.....	61
IV. Bibliografy	62

I. Introduction

The software maintenance cost has risen to 70% of the data processing budget in some corporations. Some of the crucial factors contributing to the software maintenance problems are identified and discussed and some management and technical solutions to the pressing problems are proposed. Over the past 20 years the classic project management methodologies for software development have been recognized as being unsuitable for bringing a sufficient percentage of projects to complete success. Traditional engineering, with its practices, has not proved effective in the production of software as with other industrial products. The best methodology used in last years is the Agile software development. We can say that the traditional software development processes are not able to deal effectively with the two main questions concerning cost and time. The literal meaning of the word agile is "characterized by rapidity, lightness and ease of movement". This therefore means that the agile development process consists of a rapid delivery of software characterized by greater ease and flexibility of development. Using a more classical definition, we could say that: "Agile is a development process that emphasizes customer satisfaction through the continuous supply of working software". The life cycle of Agile development is divided in: iteration 0, where project is initiated; Development iterations; Release; Production. [1]



The main investment done by companies that produce software is during the fase of Development iterations. The goal of this work is focused on helping to reduce the gap between designers and programmers. The main aspect was to create a collaborative tool that tried to bring together as many features as possible, also using external tools (such as Github): for designers was examined the theme of software sustainability to help him to manage architectural design decision in efficiently way (as done by Trello and Asana) based on Kanban software development. Instead for programmers was united the force of versioning and continuous integration and continuous deployment to increase the quality of developing phase.

stable architectures that are based on good, well-understood design decisions that extend the longevity of those architectures. However, in many situations software architects and developers struggle to cope with the impact of unpredictable changes (e.g., changes in technology platforms or surprising changes in the organization's business) that need extensive refactoring to implement them. We believe that this is often because architectural sustainability is not considered during the system design.

A system's longevity affects its sustainability—these two factors are essentially two sides of the same software quality problem. Architectural sustainability can be achieved through good design decisions that retain their validity and influence over the long term. So, we define *architectural sustainability* as the set of factors that promote an architecture's stability and longevity during system evolution. Capturing architecturally good design decisions can provide a basis for assessing their stability over time if changes in the decisions don't affect the resulting architecture's core. Consequently, architects should ask these key questions:

- When is an architecture considered sustainable?
- When is a design decision considered stable? What's a decision's ideal lifetime?
- How can we measure AK sustainability?

Given the number of long-lived systems in all domains today, these questions are relevant to software architects who want to raise or measure their architectures' quality, longevity, and stability.

Sustainable architectures

Estimating the sustainability of an architecture might not be easy, and we need to detect and identify “architecture smells”, showing that something is wrong or no longer adequate in the architecture. These smells often arise as consequence of architecture-related technical debt (TD), architectural mismatch or problems with the design decisions that have been made. Such loss of quality in the architecture usually affects the overall quality of the system, too; hence, we need metrics to estimate the ongoing quality of an architecture so that we can see when it starts to decay.

Another factor that clearly affects the architecture sustainability is how the evolution of the system is managed. In order to keep technical debt at acceptable levels, compensating technical changes must constantly be performed to repay the debt; otherwise, it will get out of control. We have found that there are a number of different categories of metrics that can be used to estimate the impact of changes on an architecture, which act as indicators of its sustainability:

- ripple effect metrics used to understand to what extent a change in a design decision affects other decisions (the higher the ripple effect, the poorer the architecture sustainability);
- instability, as opposed to stability, computed based on theory, as the probability of an architecture to change, while change proneness is computed empirically, and measures the effect of changes in architecture elements. Predicting instability or when a software module could change in a future version can be estimated as a probability function based on past changes and the percentage of ripple effects propagated from other modules;

- code metrics used to detect anti-patterns in the architecture that consequently might lead to architecture changes. For instance, code metrics can provide indicators about complexity like coupling and cohesion and enable detecting, for instance, god classes that are clear signs of the Blob anti-pattern.

Long-Lived decisions

The sustainability of architectural knowledge can be achieved more easily if the knowledge is explicitly documented. Ideally this would be through the use of formally documented decision models, where the key design decisions could be captured and stored to be shared or even reused in new software projects. While formal models are rarely seen in industry, more lightweight, informal documentation can be used instead (e.g., capturing architectural knowledge in textual form perhaps using a template for guidance). Our experience leads us to strongly believe that the number of design decisions that have to be maintained (and the effort to maintain them) is a key indicator of the likely level of architecture sustainability of a system. This is primarily because the process of modifying a decision is not an isolated action and often influences other related decisions.

As a result on our experience in this area, we base our assessment of the sustainability of an architecture on the following factors:

- the number of refactoring and frequency of changes performed over a period of time;
- the amount of significant design decisions changed;

- the adequacy of the trace links between design decisions and other software artifacts that eases tracking when decisions are changed.

Good design decisions usually endure over a long period of time and enable the architecture to remain stable. Regarding the lifetime of good design decisions, these must be revisited in case of large architectural changes, but should not be constantly changed, as the system evolves. If we use appropriate metrics to monitor the size of the decision network, the number of changes to the decisions during evolution cycles, and the impact of refactoring, we can better estimate the extent to which an architecture can be considered to be sustainable. Therefore, the longevity of decisions is an indicator as to whether the architecture of a system is stable. Many contemporary decision model implementations contain a timestamp and a decision history to record when decisions are modified, which allows users to know the last time a decision changed. This is also useful when calculating metrics like the ones listed above and ones relating to the frequency of change.

Figure A shows the factors affecting architectural sustainability and longevity. These factors— derived from the complexity of a decision network or the stability of decisions—indicate TD; we use them to identify the loss of architectural quality and thus design erosion.

The bottom box in the figure shows metrics and items that can be combined to estimate AK sustainability. [2]

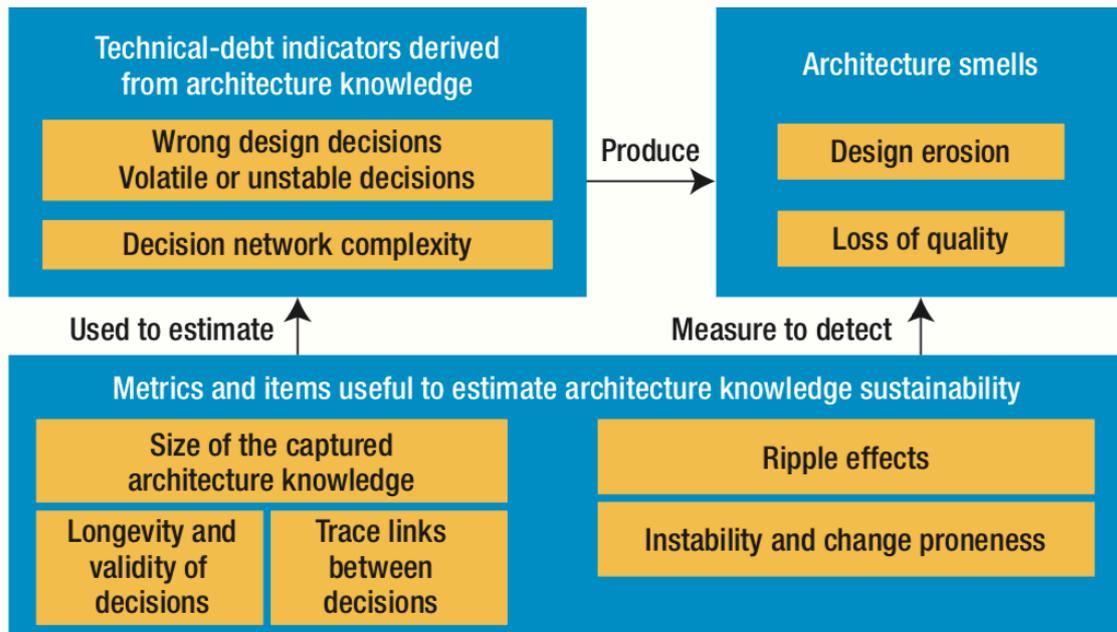


Figure A - Architecture smells derived from metrics to estimate the sustainability of architecture knowledge.

Kanban software development

The Japanese word *kanban* refers to a signboard. When the term is used in manufacturing, it means a scheduling system that hints what, when, and how much to produce.

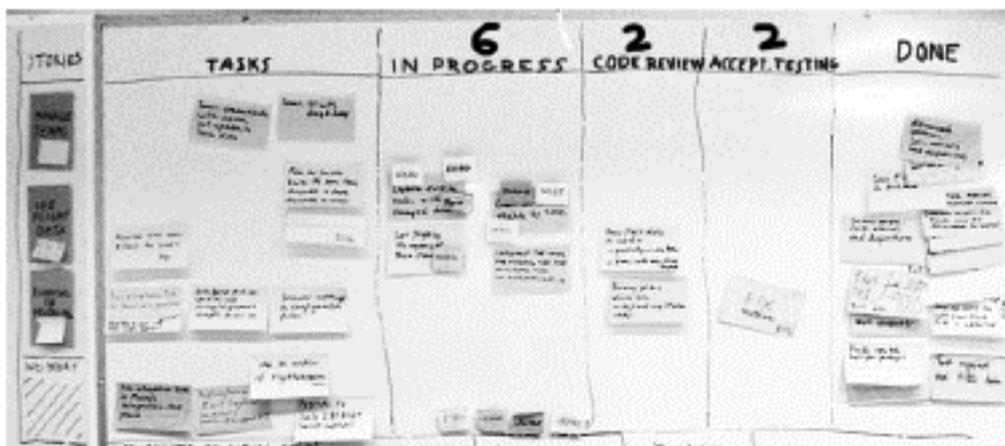


Figure B - An example of one implementation of the Kanban board used in a software development project.

It is claimed that Kanban is one of the important elements that executes Lean thinking in practice. Kanban is also one of the key operation management tools in Lean manufacturing. It drives project teams to visualize the workflow, limit work in progress (WIP) at each workflow stage, and measure the cycle time (i.e., average time to complete one task).

Despite the demand to visualize the workflow, there are no particular rules concerning how to implement the content of the Kanban board. In their basic form in production environments, Kanban controls are typically implemented with physical index cards (usually called *tickets*) moving along with the material. The cards then act as the flow-control tickets between the different work stations or processes. Figure B illustrates one realization of Kanban as a table (such as a wallpaper with sticky notes). Each project task (card) flows from one state to

another (from left to right) as it progresses. Hence, the overall project situation can be seen at a glance while the dynamic moving of the task cards indicates the project progress (or blocking) over time.

Benefits of Kanban scheduling are reduced inventory (simultaneous WIP), improved flow, prevented overproduction, operations-level control, visualized schedule and management of the process, improved responsiveness to changes in demand, minimized risks of inventory obsolescence, and increased ability to manage the supply change. As a result, Kanban attempts to lower production costs, increase quality, and accelerate cycle time. Meanwhile, inventories and problems caused by sudden changes will become more apparent.

However, while Kanban attempts to clarify workers' awareness of the current production issues and forthcoming tasks, it does not recommend any particular project phases, milestones, or partitioning tasks. Due to this liberty, it is up to a project team to build and customize the appropriate practices for its project. Some such practices have been suggested in the literature. Ladas, for example, suggests that the amount of tasks in progress simultaneously should be adjusted to the reasonable capacity in use. Middleton claims that this amount should be minimized in order to keep quality high. Shinkle, instead, argues that minimizing this amount is not the best solution.

Project flow stages wherein the tasks progress from stage to stage have been suggested as well. If each stage, such as *to do*, *design*, and *coding*, gets its own "ready" stage, the blockages in the workflow become more visible. State that tasks should be prioritized. Moreover, laborious tasks should be partitioned before setting them as assigned.

Aspect of Work	Expected Influences of Kanban
Documentation	Only if the customer needs and favors it
Problem solving	Unexpected problems are found quickly on the Kanban board; they are thus tackled immediately
Visualization	The work is visualized on the Kanban board
Understanding the whole	One of the Lean key principles; Kanban does not offer tools though
Communication	Rapid and plenty
Embracing the method	An intuitive method (e.g., visualization)
Feedback	Rapid and plenty; Supports also regular meetings with the customer
Approval process	Best expertise is at each workstation or developer; no complex approval processes
Selecting work assignments	Developers select and pull their work voluntary and individually

Table A - Nine aspects of Kanban development.

While the operations management field has investigated Kanban-based production for years, not much empirical evidence has been published in software development. It follows that the presumed advantages and suggested control rules, such as WIP sizes, remain largely anecdotal and not necessarily generally applicable.

Figure B demonstrates the meaning of WIPs. In the “Code Review” column, the WIP number has been set at two. This limit means that no more than two tickets are allowed to be in the column simultaneously. If a task being code reviewed were problematic, carrying it out without the assistance of others would restrain the flow. Other people, once they have finished their tasks, cannot put another ticket into the column because it is already full (due to the WIP limit). They rather have to help with the problematic ticket in order to free some space in the column for new tickets. In this way, the flow is supposed to be smooth and bottlenecks avoidable. [12]

Git systems

Git is a distributed version-control system for tracking changes in source code during software development. It is designed for coordinating work among programmers, but it can be used to track changes in any set of files. Its goals include speed, data integrity, and support for distributed, non-linear workflows.

As with most other distributed version-control systems, and unlike most client-server systems, every Git directory on every computer is a full-fledged repository with complete history and full version-tracking abilities, independent of network access or a central server.

Git is free and open-source software distributed under the terms of the GNU General Public License version 2.

Branching and merging

The Git feature that really makes it stand apart from nearly every other SCM out there is its branching model.

Git allows and encourages you to have multiple local branches that can be entirely independent of each other. The creation, merging, and deletion of those lines of development takes seconds.

This means that you can do things like:

- **Frictionless Context Switching.** Create a branch to try out an idea, commit a few times, switch back to where you branched from, apply a patch, switch back to where you are experimenting, and merge it in.
- **Role-Based Code-lines.** Have a branch that always contains only what goes to production, another that you merge work into for testing, and several smaller ones for day to day work.
- **Feature Based Workflow.** Create new branches for each new feature you're working on so you can seamlessly switch back and

Small and fast

Git is fast. With Git, nearly all operations are performed locally, giving it a huge speed advantage on centralized systems that constantly have to communicate with a server somewhere.

Git was built to work on the Linux kernel, meaning that it has had to effectively handle large repositories from day one. Git is written in C, reducing the overhead of runtimes associated with higher-level languages. Speed and performance has been a primary design goal of the Git from the start.

Let's see how common operations stack up against Subversion, a common centralized version control system that is similar to CVS or Perforce. *Smaller is faster.*



For testing, large AWS instances were set up in the same availability zone. Git and SVN were installed on both machines, the Ruby repository was copied to both Git and SVN servers, and common operations were performed on both.

In some cases the commands don't match up exactly. Here, matching on the lowest common denominator was attempted. For example, the 'commit' tests also include the time to push for Git, though most of the time you would not actually be pushing to the server immediately after a commit where the two commands cannot be separated in SVN.

All of these times are in seconds.

Operation		Git	SVN	
Commit Files (A)	Add, commit and push 113 modified files (2164+, 2259-)	0. 64	2.6 0	4x
Commit Images (B)	Add, commit and push 1000 1k images	1. 53	24. 70	16 x
Diff Current	Diff 187 changed files (1664+, 4859-) against last commit	0. 25	1.0 9	4x
Diff Recent	Diff against 4 commits back (269 changed/3609+,6898-)	0. 25	3.9 9	16 x
Diff Tags	Diff two tags against each other (v1.9.1.0/v1.9.3.0)	1. 17	83. 57	71 x
Log (50)	Log of the last 50 commits (19k of output)	0. 01	0.3 8	31 x
Log (All)	Log of all commits (26,056 commits - 9.4M of output)	0. 52	169 .20	32 5x
Log (File)	Log of the history of a single file (array.c - 483 revs)	0. 60	82. 84	13 8x
Update	Pull of Commit A scenario (113 files changed, 2164+, 2259-)	0. 90	2.8 2	3x
Blame	Line annotation of a single file (array.c)	1. 91	3.0 4	1x

Note that this is the best case scenario for SVN - a server with no load with an 80MB/s bandwidth connection to the client machine. Nearly all of these times would be even worse for SVN if that connection was slower, while many of the Git times would not be affected.

Clearly, in many of these common version control operations, **Git is one or two orders of magnitude faster than SVN**, even under ideal conditions for SVN.

One place where Git is slower is in the initial clone operation. Here, Git is downloading the entire history rather than only the latest version. As seen in the above charts, it's not considerably slower for an operation that is only performed once.

Operation		Git*	Git	SVN
Clone	Clone and shallow clone(*) in Git vs checkout in SVN	21 .0	107 .5	14. 0
Size (M)	Size of total client side data and files after clone/checkout (in M)		181 .0	132 .0

It's also interesting to note that the size of the data on the client side is very similar even though Git also has every version of every file for the entire history of the project. This illustrates how efficient it is at compressing and storing data on the client side. [3]

Continuous software engineering

Continuous software engineering is an emerging area of research and practice. It refers to develop, deploy and get quick feedback from software and customer in a very rapid cycle. Continuous software engineering involves three phases: Business Strategy and Planning, Development and Operations. This study focuses on only three software development activities: continuous integration, continuous delivery and continuous deployment. Figure 1 shows the relationship between these concepts.

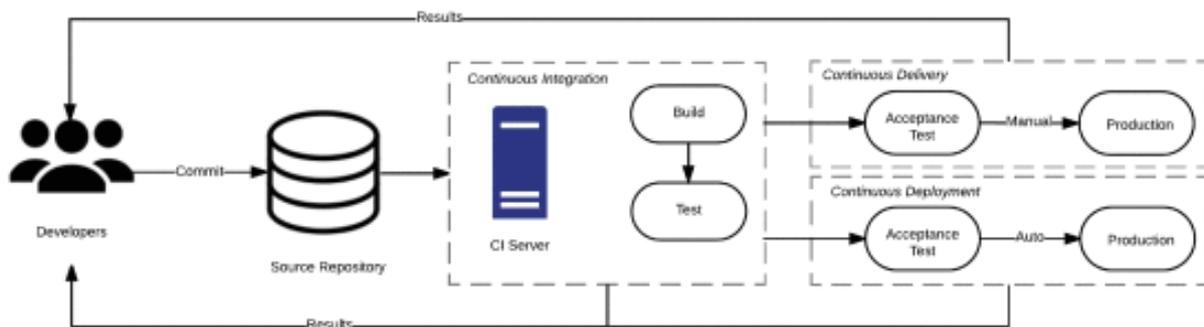


Figure 1 - The relationship between continuous integration, delivery and deployment.

Continuous Integration

In the area of modern client-side web development, there are lots of communities and organizations which have offered JavaScript frameworks and plugins on the internet. Those frameworks and plugins have many useful modules and functionalities that help to speed up the development process, yet user' requirements still keep changing over time (e.g. better interactivity, better performance or better security). These points are constantly increasing the complexity of application development. This is a reason why developers have to rely on more than one framework or a plugin within an application which of course often leads to conflicts or even defects. Usually, defects are caught during the integration process and it is very hard to fix them quickly because web applications need to be tested on many different environments including multiple operating systems and even more browsers.

The issues described above are demanding a proper solution, because unexpected and unforeseeable problems might affect project deadlines. This might result in unsolvable bugs for more complex applications. To improve this situation and enhance development quality and efficiency, Continuous Integration (CI) is a very promising approach to apply.

Hence, to reduce the number of defects during the integration process, some managing processes are required to work automatically.

Continuous Integration (CI) is a software practice that requires software to be integrated in a shared repository up to several times a day. Each integration is verified by an automated build, which includes testing to detect integration errors as quickly as possible. The first step in a CI Process is developers committing their code to a Version Control Systems (VCS). A CI server detects changes in the repository (e.g. by polling every few minutes). When a change is detected the project is pulled into the CI server itself. This will trigger a build process which

builds the software. Then the built-result report will be generated by CI server and sent to project manager (or members).

The CI server continues to check for further changes and repeat this cycle. The goal is to convert from manual integration to automated integration. Yet, common CI is just a basic concept which still needs to be customized in order to smoothly adapt with modern client-side web application's testing systems.

Overview and detail CI process

A. Overview CI Process

Figure 2 presents the system overview and the components of our proposed CI process that are involved in the whole CI development. The main actors are: developers, a Version Control System (VCS), an on demand cloud testing service, and a testing server. The process is consistent with the common CI approach, yet there are some specific tasks that might work differently. For example, in the CI server program contains four mandatory processes which need to be run sequentially. The four processes are: “Test before build”, “Build”, “Test after Build”, and “Deploy”. And, the “Test after Build” is selected to perform all tests on a Cloud Service.

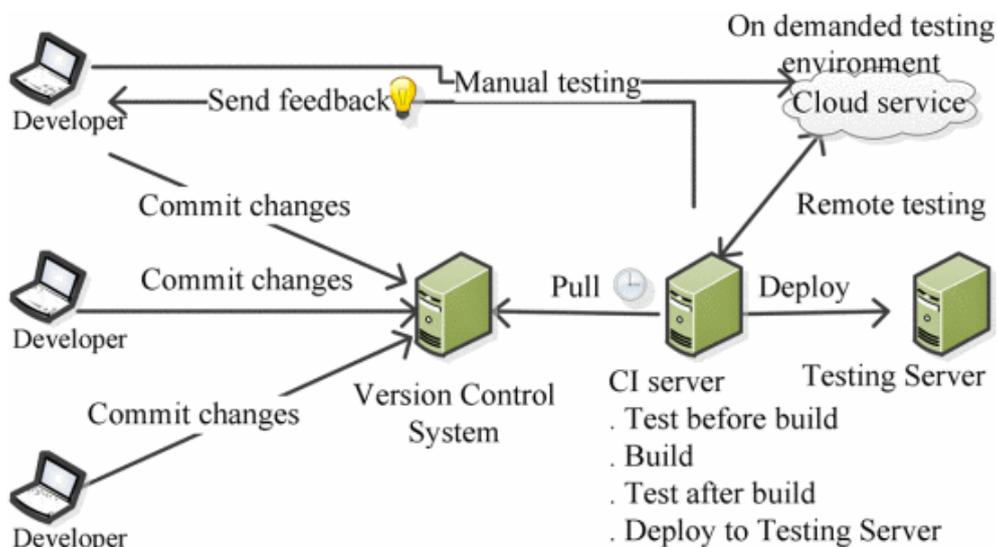


Figure 2 - CI components in modern client-side web application.

B. Detail CI Process

1) Development and Manual Testing

Local development is an important part in the CI approach. Every new commit from a local development is a key signal which triggers the CI server to execute its automation processes (i.e. test, build, and etc.). A well-defined and executed local development helps CI approach to work effectively. The developers need to be aware of the basic practical flow during their development.

This process starts with developers checking out the project from VCS, so that they can work on the assigned tasks. The specific tasks of developers consist of maintaining source code and tests (unit test and functional test). Unit tests are simply used to verify the behavior of single elements (or units) in a software system. Those are, most of the time, single methods (functions) or classes. It is a type of test which works by executing a piece of code directly and expecting a specific result. On the other hand, Functional tests work by issuing the commands to a device/browser that mimic actual user interaction. Functional tests do not always call other APIs directly; hence mockup data needs to be provided to test such behavior.

Every day, developers also perform manual tests on their local machine in order to make sure that all tests are passed and the source code work correctly. Then at the end of this cycle developers commit their source code and tests to the VCS.

2) Automated Process on a CI Server

Setting up CI server can be done by manual or using its tool like CruiseControl or CruiseControl.NET. Due to the fact that web application's behavior works different from other applications written by java, ruby or. net, we decided to set up CI server by ourselves. These CI server are responsible for the four mandatory automated processes.

Hence, this server will automatically check new commit from VCS according to our pre-defined period of time, and once a change is detected, the processes will be executed as consequence.

a) Test Before Build

The aim of “Test before Build” is to catch the errors as soon as a new commit is made. Fortunately, with modern web applications we do not need to compile our code before running it. It means browsers act as the platform to interpret JavaScript code. So, if any test fails after running in the browsers, the project is fragged, indicating to the developers that it contains errors.

We know that the CI server is running on a predefined time-based trigger. When the trigger goes off, the CI program checks-out the project from the VCS. After that, the program CI determines if there is a new commit? If so, the CI server will automatically run “Test before Build” script, otherwise it will do nothing in this cycle. The “Test before Build” script will invoke browsers (local browsers which are installed in CI server) and perform all tests. When it finishes, it will summarize the result. Then, the CI program will determine if the result contains errors. If so, the program will send an email to notify the project manager (or a team member) with a log file. Otherwise, the CI program ends this process and the CI server will wait for the next trigger event. The Figure 3 shows the process of the CI program.

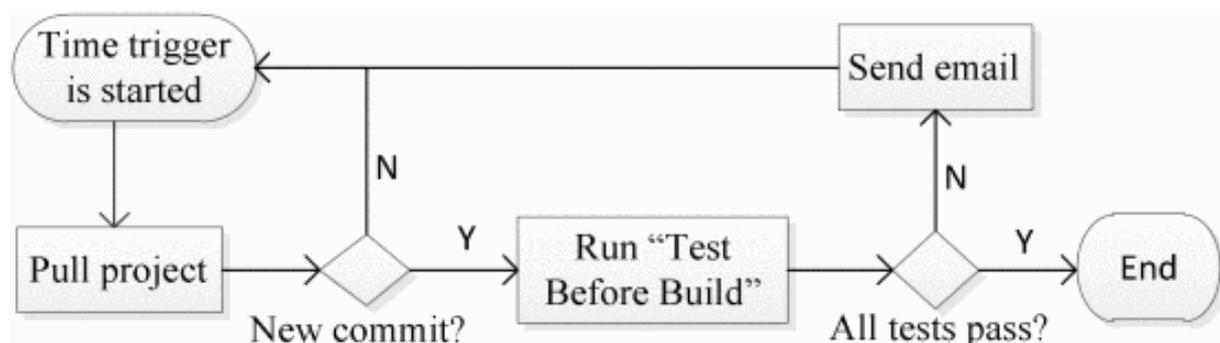


Figure 3 - CI program on “test before build” process.

b) Build

Usually, the word “build” in software development landscape is used for compiling from source code to machine code. However in web application development, it means source code minification and source code obfuscation. Minification is a process of combining multiple project files into few combined files then removing all unnecessary characters or dead-code without changing its functionality. Notwithstanding, the obfuscation process is used to hide the actual code intention. So that, the process will rename variable and function names to meaningless names, and remove all extra white space and line breaks.

The build process is performed automatically on the CI server, after a successful “Test before Build”. Then after the build process finishes, the CI program finds the error in log file. If there is any error, the program will send an email to the project manager with log file. Otherwise, the CI program ends the build process and the CI server will wait for the next trigger event.

c) Test After Build

Normally, a web application might run after the build process is finished, but it does not mean it does all right thing. In a real practice, the errors might encounter. Therefore, performing automated test is the best way to detect and announce the errors quickly. And this process is named “Test after Build” and it will be executed after the previous build process is done without any errors.

The “Test after Build” process is worked with external platform. It means all tests will be run on a cloud-based service. The cloud-based service allow us to perform the tests on a number of different environments, i.e. operating systems, browsers, system resources, and etc., which are not existed on our local. This is very helpful because performing tests on a local machine is very costly to include many environments.

Recently, using cloud services has become common in the software development. The testing process is controlled by the CI server. The CI

server opens the connection to the cloud-based service. Then, it sends following commands to the cloud service: authentication, initialize the operating systems with the corresponding browsers that we want to run our tests. The cloud-based service listens to the commands and works accordingly. Once it finishes it will send the result back to the CI server, and then the result will be saved in a log file. The whole process can be easily illustrated in the Figure 4.

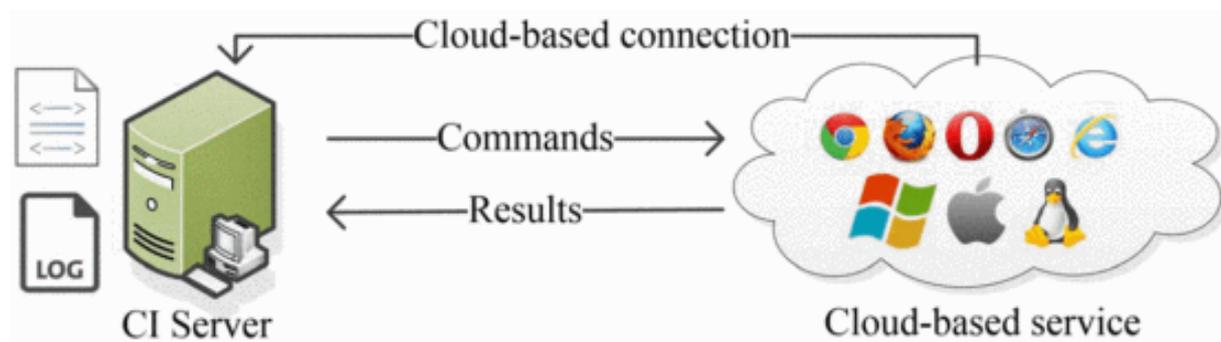


Figure 4 - Simple cloud-based process.

d) Deploy

Deploying a web application on a test server ensures that the application can be manually tested. Since the programmers deploy multiple times during the development process, it is preferred to automate this job. As a consequence, the deploying process will be started once the “Test after Build” process finishes without any defects. The process is such a good approach for testers (QA, other developers, project manager, and etc). They could perform manual testing on the latest version of the application as fast as the whole CI cycle is done. Especially, with modern client-side web applications, this can be easily done by copying all your project files to web directory of the test server. After the automated deployment process is finished, we successfully completed CI cycle and the next cycle will start by the time trigger. [4]

Continuous Deployment

With increasing competition in software market, organizations pay significant attention and allocate resources to develop and deliver high-quality software at much accelerated pace. Continuous Integration (CI), Continuous DElivery (CDE), and Continuous Deployment (CD), called continuous practices for this study, are some of the practices aimed at helping organizations to accelerate their development and delivery of software features without compromising quality. Whilst CI advocates integrating work-in-progress multiple times per day, CDE and CD are about ability to quickly and reliably release values to customers by bringing automation support as much as possible.

Continuous practices are expected to provide several benefits such as: (1) getting more and quick feedback from the software development process and customers; (2) having frequent and reliable releases, which lead to improved customer satisfaction and product quality; (3) through CD, the connection between development and operations teams is strengthened and manual tasks can be eliminated. A growing number of industrial cases indicate that the continuous practices are making inroad in software development industrial practices across various domains and sizes of organizations. At the same time, adopting continuous practices is not a trivial task since organizational processes, practices, and tool may not be ready to support the highly complex and challenging nature of these practices.

Due to the growing importance of continuous practices, an increasing amount of literature describing approaches, tools, practices, and challenges has been published through diverse venues. An evidence for this trend is the existence of five secondary studies on CI, rapid release, CDE and CD. These practices are highly correlated and intertwined, in which distinguishing these practices are sometimes hard and their meanings highly depends on how a given organization interprets and employs them. Whilst CI is considered the first step towards adopting CDE practice, truly implementing CDE practice is

necessary to support automatically and continuously deploying software to production or customer environments (i.e., CD practice). There was no dedicated effort to systematically analyze and rigorously synthesize the literature on continuous practices in an integrated manner. By integrated manner we mean simultaneously investigating approaches, tools, challenges, and practices of CI, CDE, and CD, which aims to explore and understand the relationship between them and what steps should be followed to successfully and smoothly move from one practice to another.

Continuous Integration (CI) is a widely established development practice in software development industry, in which members of a team integrate and merge development work (e.g., code) frequently, for example multiple times per day. CI enables software companies to have shorter and frequent release cycle, improve software quality, and increase their teams' productivity. This practice includes automated software building and testing.

Continuous Delivery (CDE) is aimed at ensuring an application is always at production-ready state after successfully passing automated tests and quality checks. CDE employs a set of practices e.g., CI, and deployment automation to deliver software automatically to a production-like environment. This practice offers several benefits such as reduced deployment risk, lower costs and getting user feedback faster. Figure 1 indicates that having continuous delivery practice requires continuous integration practice.

Continuous Deployment (CD) practice goes a step further and automatically and continuously deploys the application to production or customer environments. There is robust debate in academic and industrial circles about defining and distinguishing between continuous deployment and continuous delivery. What differentiates continuous deployment from continuous delivery is a production environment (i.e.,

actual customers): the goal of continuous deployment practice is to automatically and steadily deploy every change into the production environment. It is important to note that CD practice implies CDE practice but the converse is not true. Whilst the final deployment in CDE is a manual step, there should be no manual steps in CD, in which as soon as developers commit a change, the change is deployed to production through a deployment pipeline. CDE practice is a pull-based approach for which a business decides what and when to deploy; CD practice is a push-based approach. In other words, the scope of CDE does not include frequent and automated release, and CD is consequently a continuation of CDE. Whilst CDE practice can be applied for all types of systems and organizations, CD practice may only be suitable for certain types of organizations or systems. [5]

II. Software developed

The developed software tries to combine all these issues described to help companies that produce software, keeping track of everything that happens within each development team, from solving a small bug to a total recreation of the project.

A web-application was made:

- Front-end: Bootstrap [6] for the User Interface (html5, css3), with a small customization of the basic components; for the dynamism of the site it was consequently used javascript with jQuery framework [7], especially for APIs calls to the server through Ajax.
- Back end: the infrastructure is active on an apache server [8], using PHP&MySQL. The framework used to create endpoints and interact with the client is Slim3 [9] (PHP micro-framework).

The choice to use this type of infrastructure was born from the need to try to combine speed of use by the teams and computing power that did not depend on the devices with which it was accessed, but the whole depends on the power of a single server that manages everything . Another basic reason is that being a collaborative software, a structure like a web application has no comparisons with other types of types. Once this infrastructure has been terminated, it will be possible to create any type of application (mobile, desktop) able to connect to the central server in order to allow expanding the functions currently offered and described below.

Functionalities

Home page

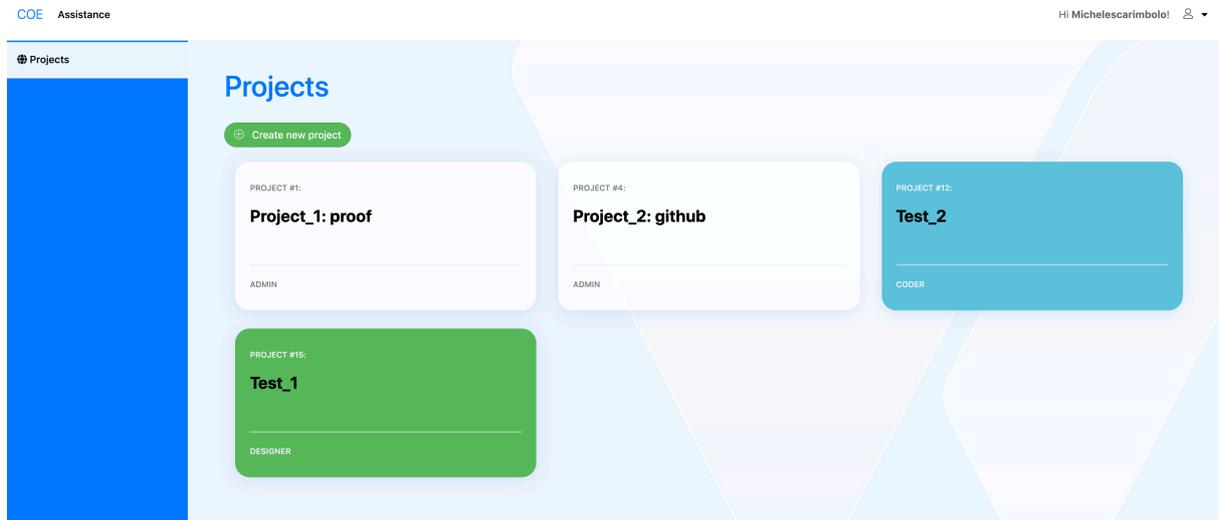


Figure 5 - Home page

The home page (fig. 5) of the software is presented with the list of projects in which one participates (created or invited). A color pattern has been defined relative to the role the user has within each project: white for the role of ADMIN, green for the role of DESIGNER and blue for the role of CODER.

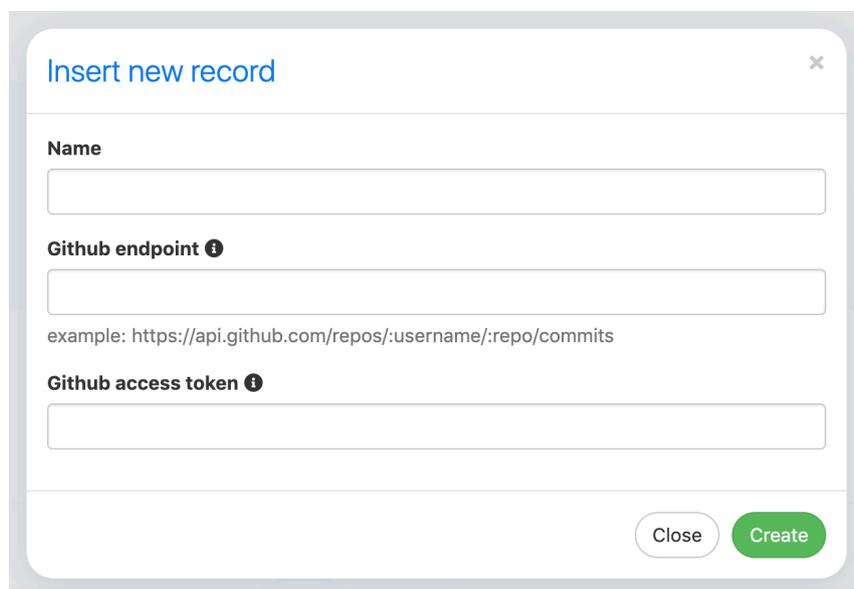
The difference in these roles consists of:

- ADMIN: being the creator, he has the possibility to do everything related to the project functions (which we will see later) and can invite users to collaborate on the project, defining their roles.
- DESIGNER: this is the most "significant" role since it represents the users who define the architectural part of the project, have the possibility to create, delete and modify the design decisions of the project (technologies to be used, distributed system, etc.).

- CODER: this role is related to programmers, those who write code related to every decision taken by the DESIGNERS.

This last role was included in the first version of the software, because initially it was not thought to use an existing git system (Github) but it was thought to create a new one, resident within the production server, giving the possibility to write code in the software, using an editor (Ace.js). Since it would have been a superfluous job, to create a git system from scratch, in the second version this role was eliminated, leaving space for ADMINS and DESIGNERS that will keep the developments of the CODERS under control directly within the screen of the individual decisions using, as we will see, the APIs of Github.

Creation of project



The image shows a modal window titled "Insert new record" with a close button (x) in the top right corner. The form contains three input fields: "Name", "Github endpoint" (with an information icon), and "Github access token" (with an information icon). Below the "Github endpoint" field, there is an example URL: "example: https://api.github.com/repos/:username/:repo/commits". At the bottom right of the modal, there are two buttons: "Close" and "Create".

Figure 6 - Project creation

Clicking on "Create new project" (fig. 5), a modal will open (fig. 6) where the project data is requested: name, Github repository endpoint where the project is present, and if this is private you must also enter the

GitHub access token, requested by the project administrator on the GitHub platform.

Project page

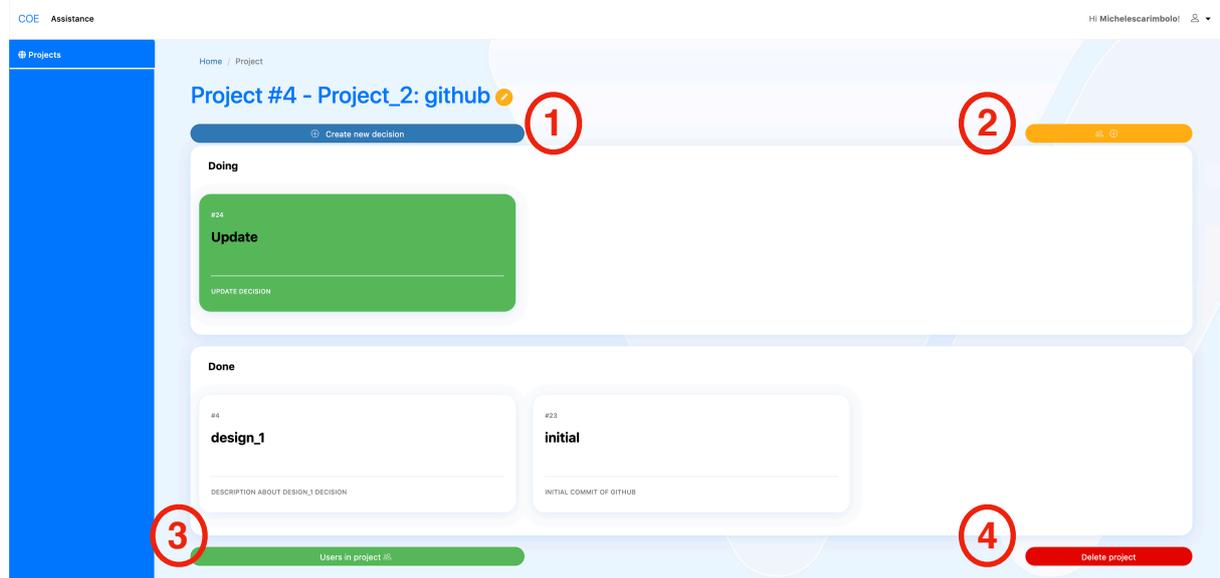


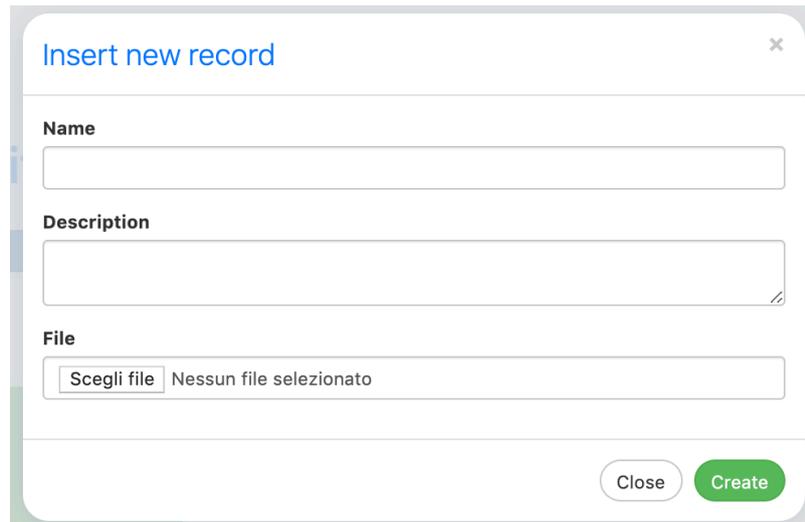
Figure 7 - Project page

Created the project, clicking on it will enter the project page. The two main sections are DOING and DONE, which reflect the project management board used in Scrum Development [10].

The actions that can be carried out by the buttons in fig. 7 are:

1. Create new decision
2. Invite user
3. Users in project
4. Delete project

CREATE NEW DECISION



The image shows a web form titled "Insert new record" with a close button (X) in the top right corner. The form contains three main sections: "Name" with a single-line text input field; "Description" with a multi-line text area; and "File" with a file selection button labeled "Scegli file" and the text "Nessun file selezionato". At the bottom right of the form, there are two buttons: "Close" and "Create".

Figure 8 - Insert new decision

In this section it is possible to add all the information necessary to best describe the architectural decisions taken by the DESIGNERS. After a long study on Software Sustainability, we chose to use only two textual fields. The third field is related to the possibility of adding a File to be attached to the decision (ex. UML file, PDF, etc.) that can help in sharing ideas.

The Measuring Sustainability of Architectural Knowledge is explained below:

“The sustainability of architectural knowledge can not only be estimated in terms of how much effort we need to maintain the knowledge, but also how stable the decisions are and their longevity as the system evolves. Software maintainers can use this table as a guide to evaluate the sustainability of their architecture. It is important to both assess the values of the metrics at a specific point in and also to track trends in their values over time. We use the metrics in this table to measure the sustainability of the architectural knowledge and provide sustainability indicators for an architecture.

Architectural Knowledge Practice Areas	Sustainability criteria for Architectural Knowledge	Quality attributes	Metrics
Architectural Knowledge maintenance Having a reduced set of design decisions and controlling the number of trace links between decisions and other software artifacts we ease the AK maintenance tasks when decisions change.	Granularity of the design decisions and trace links. This criterion limits the granularity of the decisions to be captured at the level of packages and classes, avoiding finer-grained decisions as a way to reduce the size of the decision model and make it more manageable. For example, decisions involving the creation of UML classes will be captured but not those concerning the creation of UML attributes or methods. Avoiding fine-grained decisions reduces the number of trace links to other software artifacts.	Complexity. The complexity of the decisions network can be reduced if we limit the granularity of the decisions captured and the number of trace links between them. Stability. Changes in the AK items do not affect to the decisions captured as no new nodes are needed in the decisions network when new AK items are added.	NodeCount EdgeCount Instability
	Size of the decision model. With this criterion we limit the number of design choices. Our experience in different projects suggests ranges of alternative decisions between [1:7/10].	Cost of the effort capturing less number of decisions.	Number of Children
	Number of AK attributes captured. Capturing less amount of AK items using configurable AK templates makes the AK more manageable. We observed that capturing a number between 3 and 6 AK items seems reasonable.	Cost of the effort in capturing a number of variable AK items.	Number of Fields
Architectural Knowledge evolution Estimating better the number of decisions that will be impacted by a change and limiting this impact to a certain level helps to reduce the number of decisions to be analyzed during evolution cycles. Also, if we know in advance which decisions must be revisited we can predict better the stability and longevity of the AK model.	Number of design decisions impacted. Decisions that change have an impact on other related decisions. Limiting this impact using a ripple effect algorithm, we can reduce the number of decisions that must be analyzed or revised. However, the designer must establish this limit based on past experience. We can also limit the exploration to prune those decisions in the network that are "end-nodes".	Changeability: We can reduce the amount of effort to change the decisions impacted by a change limiting the number of decisions analyzed. Stability: If a decision changes less number of times, it affects to the stability of the architecture, as good decisions endure over time.	Change Impact Analysis (ripple effect) Change proneness Instability
	Number of times a decision changes. We can store information about how many times a decision changes in specific periods of times and use this information to control the stability of the decisions and how often they change. Therefore, we can provide indicators about the longevity of more stable decisions.	Stability: Decisions that change less often play a key role in favor of the stability of the architecture and longevity of the decisions as well.	Decision Volatility
	Validity of decisions. We can set specific dates for decision review and remove obsolete decisions that seem no longer valid.	Stability, Timeliness	Not yet defined but the timestamp of the decisions can be used

Table 1 - Assessment criteria to measure architectural knowledge sustainability.

Table 1 suggests a set of metrics that software engineers can use to estimate the sustainability of architectural knowledge. We can combine several metrics to estimate a particular quality attribute. For instance, estimating the *complexity* of a decision network involves combining the "NodeCount", "EdgeCount", and "NumberOfChildren" metrics to estimate how complex, and hence how sustainable, the decision network is as it evolves. In those cases where we need to decrease the cost of the decisions captured and make the architectural knowledge capture more sustainable, the "NumberOfFields" indicator, combined with the number of decisions captured and the time spent in capturing them, serve as indicators to measure the ideal size of a decision model in different development contexts (e.g., agile versus RUP)." [2]

INVITE USER

In this section it is possible (for ADMINS) to invite users through their @username and assign them a role: ADMINS or DESIGNERS.

Once invited, the selected user will see in his Home page (fig. 5) the new project with the color relative to the role assigned by the ADMIN who invited him.

USERS IN PROJECT

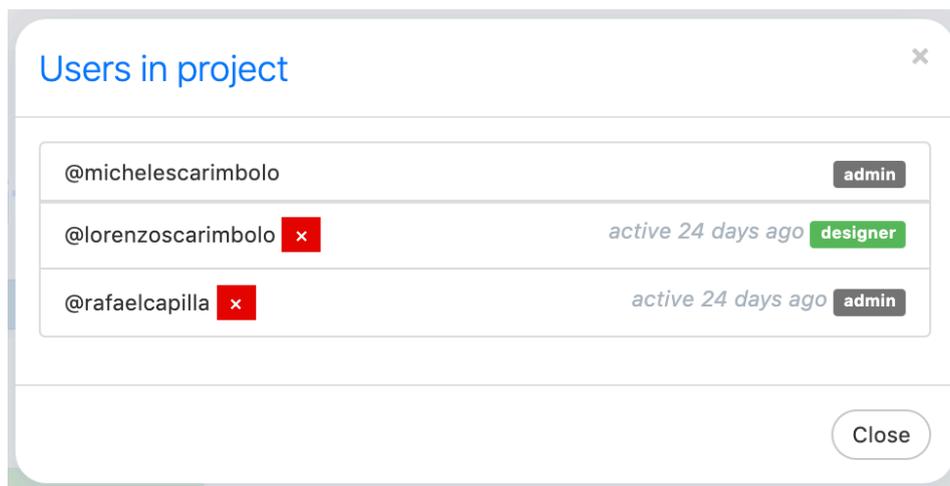


Figure 9 - Users in project view

Through this section it is possible to see the users who are collaborating in the project, their roles and their activity.

If you are an ADMIN of the project, the red X next to each name will be visible to have the possibility to exclude that user from the project.

DELETE PROJECT

The Delete functionality will be visible in all sections of the software, with the same interface, ie a modal that asks you if you are sure you want to delete that element.

Design decision page

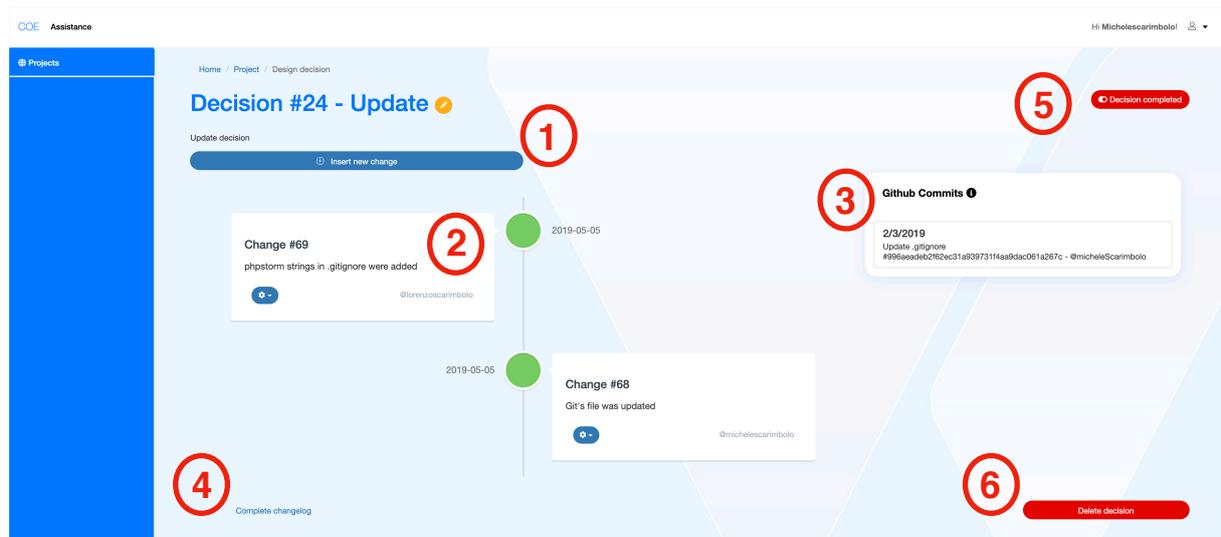


Figure 10 - Design decision page

After all the architectural decisions of the project have been created, it will be possible to analyze them one by one by clicking on them in the project page (fig. 7). This is the main section of all the software.

On this page you can check and view all the work done on a specific architectural decision. It is divided mainly into two sections: changes manually written by DESIGNERS and a list of COMMITS taken directly from the Github repository (initially linked when the project was created).

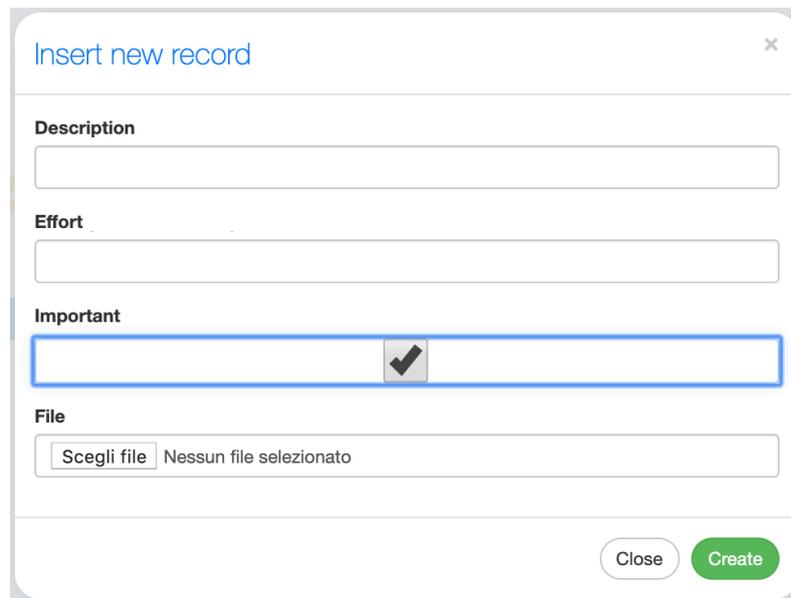
The following features can be highlighted:

1. Insert new change
2. Changes timeline view
3. Github commits view
4. Complete changelog

5. Decision completed

6. Delete decision

INSERT NEW CHANGE



The image shows a web form titled "Insert new record" with a close button in the top right corner. The form is divided into four sections: "Description" with a text input field; "Effort" with a text input field; "Important" with a checked checkbox; and "File" with a file selection button labeled "Scegli file" and the text "Nessun file selezionato". At the bottom right, there are "Close" and "Create" buttons.

Figure 11 - Insert new change in selected decision

This function has been inserted because in addition to the commits, it could become necessary to insert notes that you want to take as you develop a certain decision. The information that can be collected for each change is: description of the work done, effort (currently not in use since as explained in the section "Create new decision" you are not yet able to define an effort for the work done, you could trivially use time), important flags and files for any attachments.

The Important flag is used to decide whether the change entered should be highlighted when opening the decision page or it is something that could take second place: if checked, we will see the change in the timeline of the decision page (fig. 10, section (2)); otherwise we can review it in the Complete changelog section (fig.10, section (4)).

CHANGES TIMELINE VIEW

The timeline of the changes needed to bring out, just open the page, the important changes within the decision (ex. Update framework from Slim2 to Slim3).

It is characterized by a menu, in which you can choose to change or delete the change, from a tag that shows who created that change and the date of the operation.

GITHUB COMMITS VIEW

This view serves to show the progress of the CODERs work in relation to the decision you are browsing.

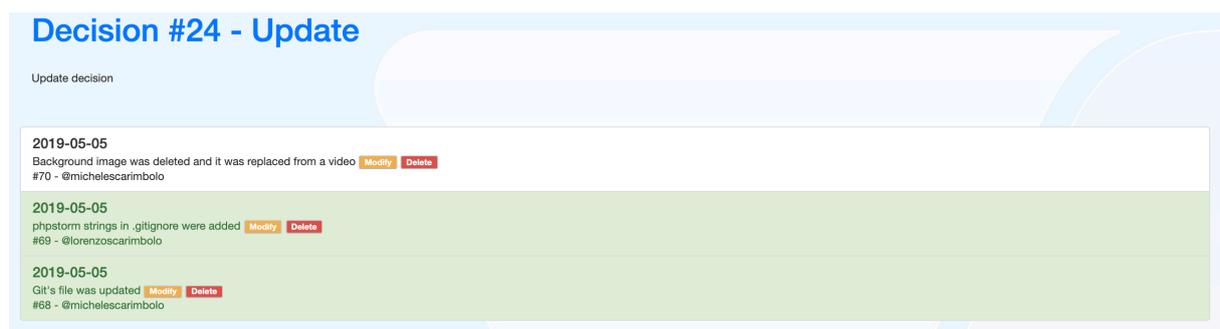
- How can you view certain commits from the same repository?

Thanks to the Github APIs [11] it is possible to return all the repository commits:

```
GET /repos/:owner/:repo/commits
```

Commits are filtered based on the name of the decision. When CODERs push into the project, based on the decision they are working on, **they will have to write the name of the decision as the first word/phrase** and then describe what has been done, so everything can be filtered and displayed correctly for each decision.

COMPLETE CHANGELOG



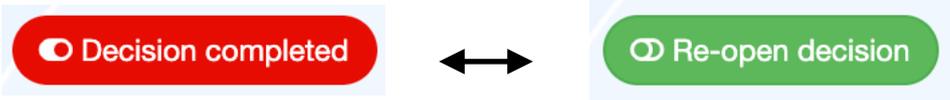
The screenshot shows a web interface for a decision titled "Decision #24 - Update". Below the title, there is a list of changes. Each change entry includes a date (2019-05-05), a description of the change, and a user identifier. The changes are:

- 2019-05-05: Background image was deleted and it was replaced from a video. #70 - @michelescarimbolo
- 2019-05-05: phpstorm strings in .gitignore were added. #69 - @lorenzocarimbolo
- 2019-05-05: Git's file was updated. #68 - @michelescarimbolo

This view is used to list all the changes that have not been marked as Important at the time of insertion, as they could be considered not so fundamental to be seen on the home page but it could be useful to keep track of them. The green ones are the important ones, the others not.

DECISION COMPLETED

When the decision is completed, simply click on "Decision completed" (fig. 10, section (5)) to move it to the DONE section on the project's home page. If you enter into a completed decision, the button changes to "Re-open decision" and if you click moves back into DOING them.



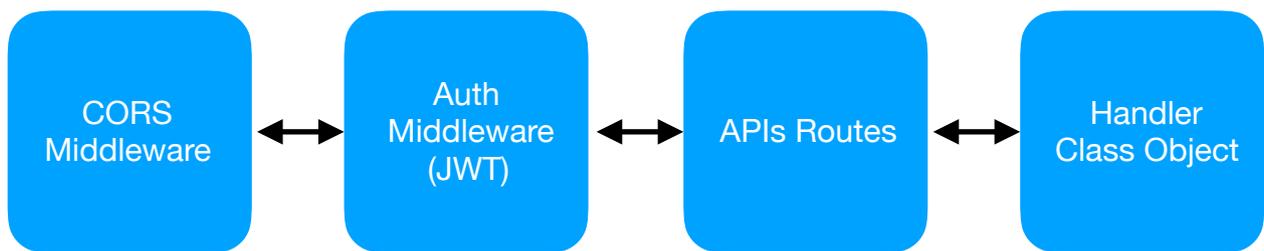
DELETE DECISION

Useful function to delete a decision and all related changes.

Code: main artifacts

The project is dived in two sides: client and server. These sides work in two separated virtual host, running on Apache server, installed in a Ubuntu machine.

SERVER



Server side is built with Slim3, PHP micro-framework. Thank to this framework it was possible realize a simple REST APIs structure for the software.

Example

```
/**
 * Route: /project
 * Methods: GET, POST, PUT, DELETE, OPTIONS
 */
$app->map(['GET', 'POST', 'PUT', 'DELETE', 'OPTIONS'], '/project/{id}', function (Request $request, Response $response, array $args) {
    $handler = new \handler\COE\HandlerProject($this);
    $method = $request->getMethod();
    if (isset($args['id'])) {
        $handler->setId($args['id']*1);
    }
    $handlerLogin = new \handler\COE\HandlerLogin($this);
    $user = $this->has('user') ? $this->get('user') : null;
    try {
        switch ($method) {
            case "GET": {
                return $response->withJson($handler->getResultInArray());
            }
            case "PUT": {
                $handler->setValuesInArray($request->getParsedBody());
                return $response->withJson($handler->getResultInArray());
            }
            case "DELETE": {
                if ($handlerLogin->verifyUserIsAdmin($user->id, $args['id']) * 1 == 1) {
                    return $response->withJson($handler->deleteValueProject());
                } else {
                    return $response->withJson("Unauthorized", 401);
                }
            }
        }
    } catch (Exception $e) {
        return $response->withJson("Unauthorized", 401);
    }
});
```

Before an example of API for Project entity. It was used PSR-7 for HTTP message interfaces.

API Route: `.../project[{id}]`

with GET, POST, PUT, DELETE, OPTION available methods.

The syntax of route is called Backus–Naur form or Backus normal form (BNF) that is a notation technique for context-free grammars. Analyzing in detail the route it's possible to see the `:id`, it's in `[]`, that mean "If it is present something after the `/`, it will be call `:id`". So with this syntax it's possible vary the function in one single route name.

When this API is called, based on the METHOD, it calls different function, in particular for the project entity:

- GET: get information for the project selected, because is called with the `:id` parameters, and retrieve the detail for project
- PUT: insert new project in the DB (without insert `:id` manually)
- DELETE: delete an selected project with `:id` parameter

Some method required specific authorization to be executed, in-fact before the main function call, it's present a function "verifyUserIsAdmin" that given the JWT, with some information about the user that have done the call, it check in the DB if his role (of the project selected) is an admin or not.

CLIENT

The UI was built in html5, css3 (bootstrap) and javascript (jQuery).

The design pattern used for the client was Model View Presenter (MVP).



Model View Presenter (MVP)

Model-View-Presenter (MVP) is an architecture pattern for the presentation layer of software applications. The pattern was originally developed at Taligent in 1990s and first was implemented in C++ and Java.

In MVP, the View and the Model are neatly separated and the View exposes a contract through which the Presenter access the portion of View that is dependent on the rest of the system.

The Model is the component which preserves data, state and business logic; it just exposes a group of service interfaces to Presenter and hides the internal details.

The View is the user interface, it receives user's action and contract to Presenter to achieve user's need, and then the View responds user by result information.

The Presenter sits in between the View and the Model; it receives input from the View and passes commands down to the Model. It then gets result and updates the View trough the contracted View interface.

“Fig. 12” illustrates the parts of MVP pattern and how they interact with each other.

Since the MVP pattern was put up in 1990s, it has been widely discussed in the area of software engineering; Martin Fowler reported some methods of implementing MVP at his papers and books. However, few wittier have considered how to implement it on concrete program; this process is extremely dependent on experience of developers.

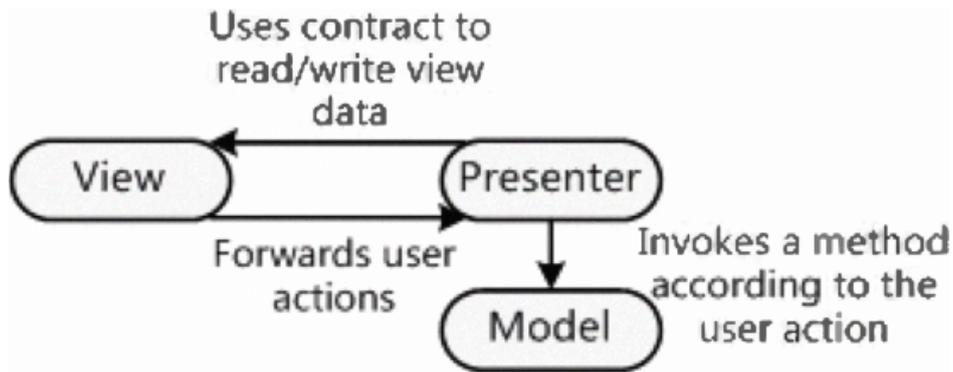


Figure 12 - MVP pattern

Contrast to traditional presentation layer, the advantage of presentation layer with MVP pattern is based on three facts:

- The View doesn't know the Model. Because of this, there is a low coupling between Model and View. It means that if Model or View was changed, another part not needs to modify as long as interfaces are stable. This also stands for the flexibility of architecture and the reusability of business logic in Model.
- The Presenter ignores any UI technology behind the View. According to this, the replacement of UI technology, such as transfer Windows Forms to WPF or to Web Forms, is not need any change of other parts. Even one application could have more than one UI technologies but one Model so that the C/S deployment and the B/S deployment are supported by it at the same time.
- The View is mockable for testing purposes. In tradition, it is impossible to test View or business logic component before another has completed because of the tight coupling between View and business logic. By the same token, the unit testing for View or business logic component is difficult. All of those problems are solved by MVP pattern. In MVP, there is no direct dependency between View and Model. For that reason, developer could use mock object to inject into View or Model so that they can be tested

on one's own.

THE ARCHITECTURE MODEL

As a pattern, MVP has various expressive forms and architectures when implementing in different platforms, the concrete implement is restrained by the features of platform, and consistency is another factor which should be considered when designing the concrete architecture model. Based on the above, an architecture model of implementing MVP pattern on .NET which is illustrated by “Fig. 13” is given here. This architecture model not only takes advantage of many specific characters of OOAD, practical experience also proves that this is workable and well-behaved.

This model is made up by five parts:

IView is the abstraction of View that is composed of a set of rules that declare what data and functions should be implemented in View. Generally every View component has its own IView component.

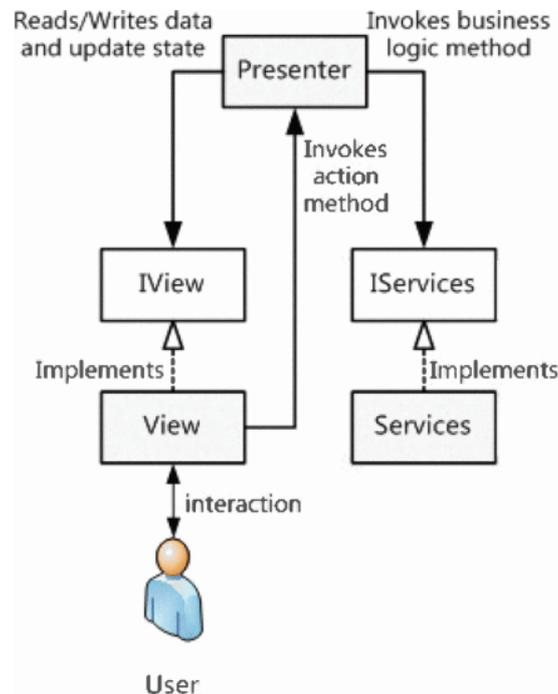


Figure 13 - The architecture model of MVP on .NET

View is the part which interacts with users. There are many technologies could be used to implement the View on. NET platform, such as Windows Forms, Web Forms, Silverlight, WPF and so on. Every View component should implement the homologous IView and one IView component could has many implements with different UI technologies, as a result the Views implemented from the same IView could take the place of each other.

IServices is a set of interfaces that define the functions should be implemented by business logic components. It is correspond to interface of The Model.

Services are the business logic components that implements IServices. View and Presenter need the help of Services to do application business on account of they do not have any business logic. Services having nothing to do UI technology commonly, the concrete implement of them are some general classes. Sometimes Services need a Repository to deal with the operation of database, this is out of this paper's range.

Presenter is the core of MVP pattern. On. NET platform presenter is a number of classes that dependent on IView and IServices. Just like the Services, Presenter is foreign to UI technology because it just dependent to IView. Presenter receives user actions and read input data from view, and then it invokes functions in Services to complete the business logic and modifies View's state. This entire works are called presentation logic.

“Fig. 14” is the sequence diagram that shows how MVP pattern works.

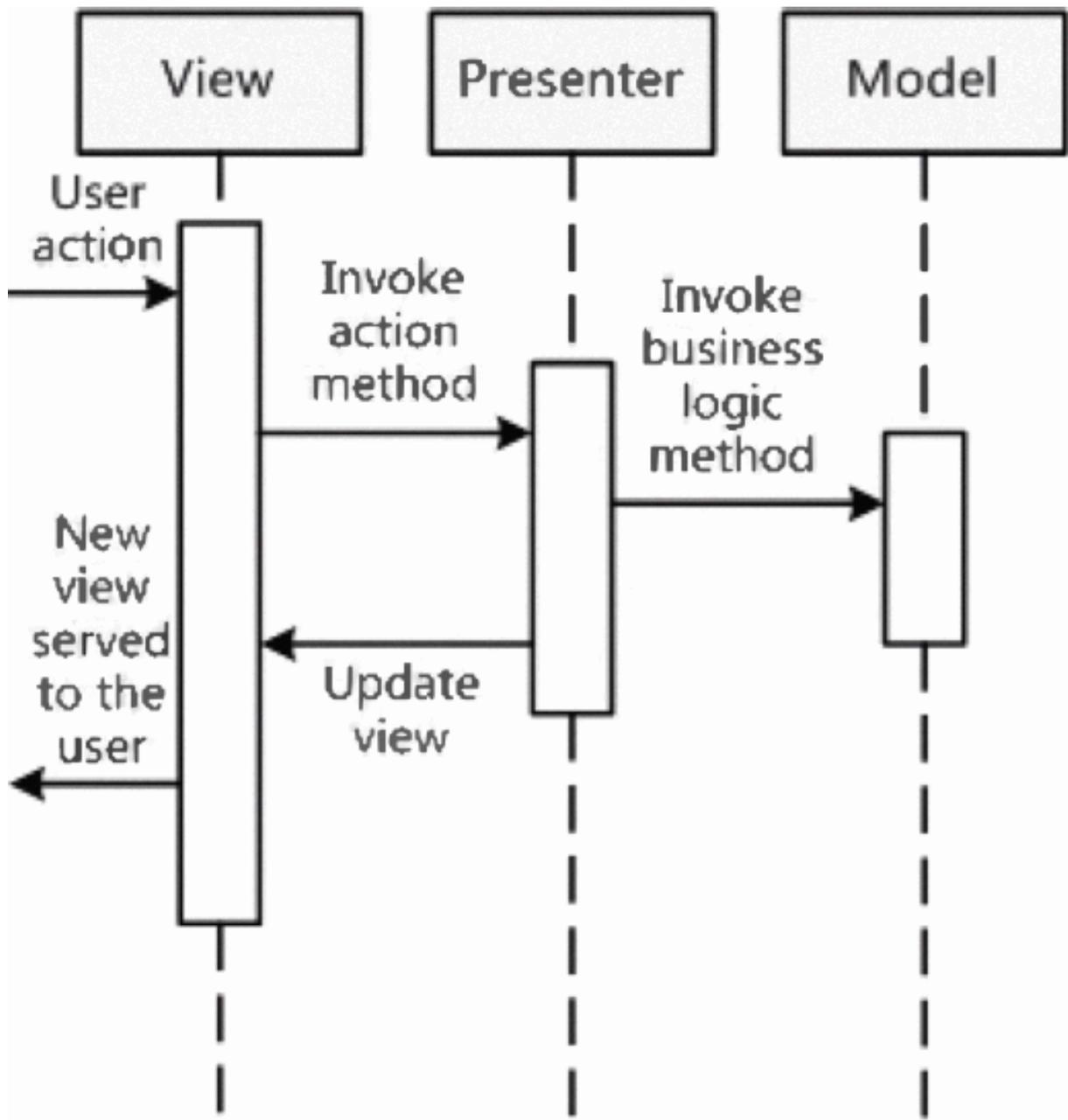


Figure 14. Sequence diagram of MVP pattern

Benchmarking

Trello

Supporting Agile Learning with Trello

An essential component of the process miniature is the integration of suitable technology to make the learning efficient and comprehensive. Kanban boards can be created on a wall or whiteboard with sticky post-its or other physical artifacts, but there are many benefits to structuring the agile learning process with a suitable software tool. To facilitate learning in our context, we needed a tool that supported the main features required for story cards and Kanban board columns, enabled online distribution and team collaboration, was easy to learn, did not require downloading or hosting, allowed private use and boards to be copied, and was free to use. Although there are many tools that can support the use of Kanban boards, and could therefore have equally well supported this learning activity, many require a minimum subscription to use (e.g. LeanKit), are free but require additional payments for some key features (e.g. creating private boards in Taiga or copying boards in KanbanFlow). Open source options are free but require web hosting (e.g. Agile Apex). Given these constraints, the tool we chose, which met all of our requirements, was Trello. Although Trello is a generic, web-based visual tool it can be used for any kind of planning or organization. Once a Trello board is created, users are able to add movable lists.

These lists are well-suited to the creation of the columns on a Kanban board. Editable cards can then be added to the lists. These cards have a range of features to assist the user such as assigning team members, and adding comments, descriptions, checklists and attachments, and are able to be categorized by color. In the Kanban context, they make ideal story cards.

Using Trello as a Kanban board to support Scrum processes has a number of advantages, including making progress visible to the whole team and allowing details of every task (such as comments, checklists, due dates, and attachments) to be added to cards. A key feature of Trello is that it supports and tracks collaboration. It enables all team members to participate in discussions, view the workflow, share files and notes and comment on the various tasks in the workflow. Users can also print, copy, and share their lists and the calendar feature helps users to meet deadlines. This digital collaboration can be asynchronous and distributed across locations. [13]

Comparison

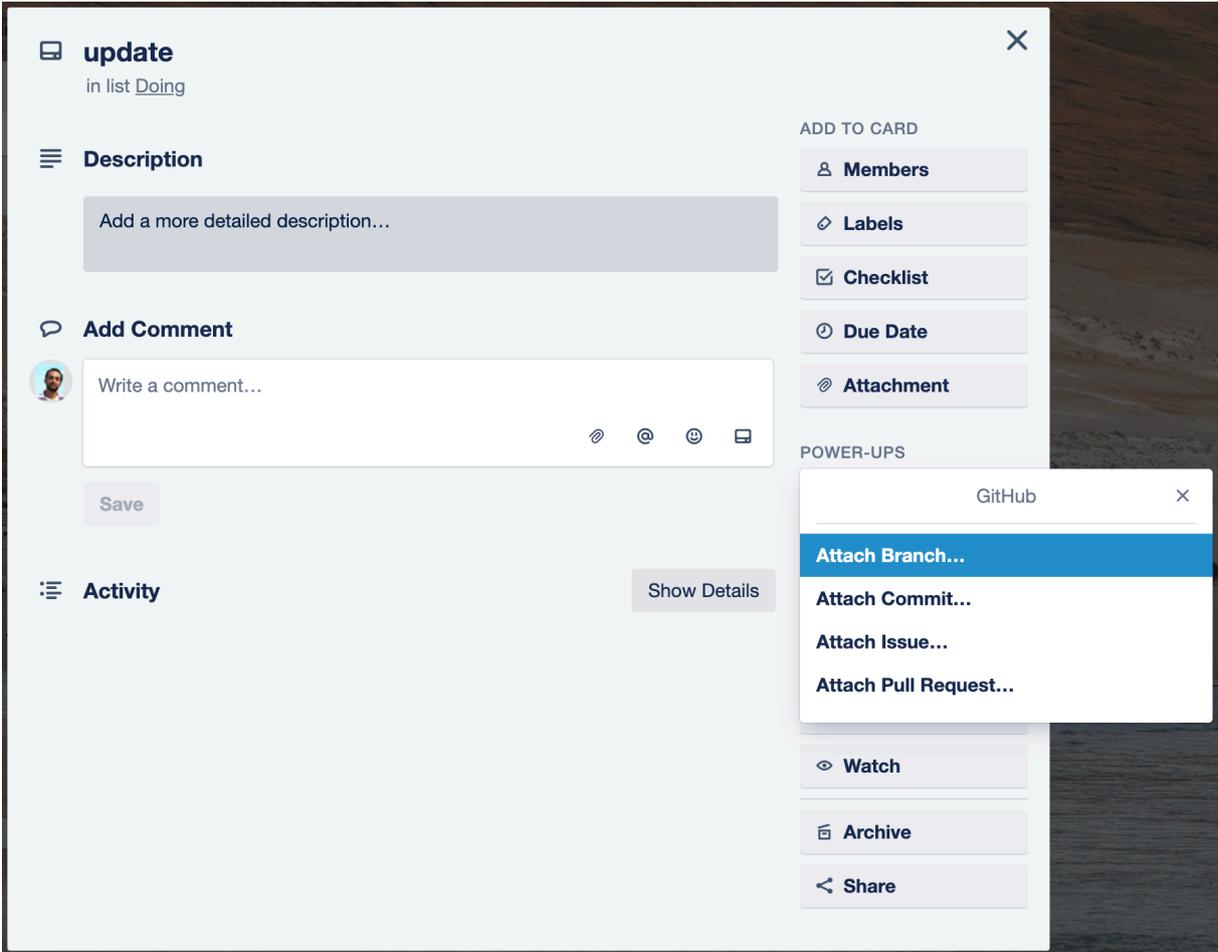


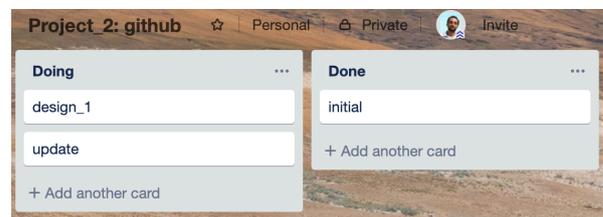
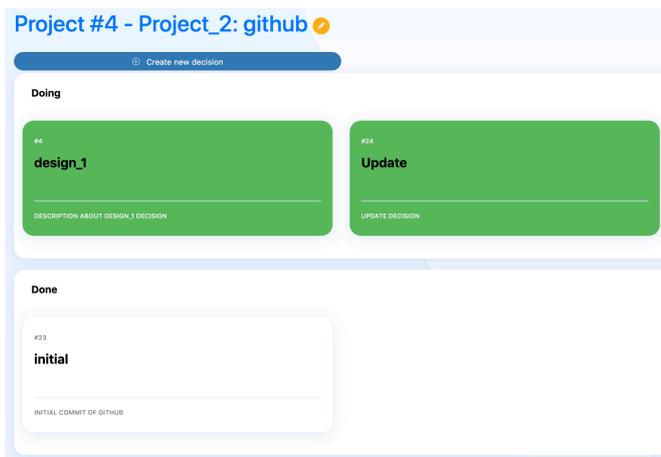
Figure 15 - Decision management in Trello

Trello includes the ability to add GitHub as a third-party application. This function allows you to link each card to a branch, commit, issue or pull request. The developed software filters, card by card, commits of a

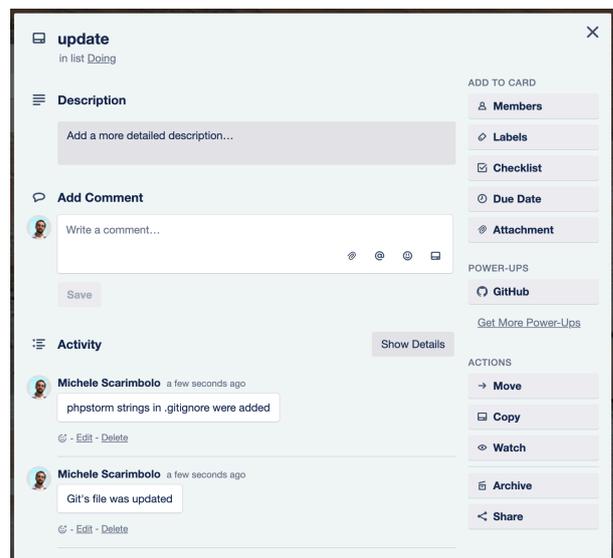
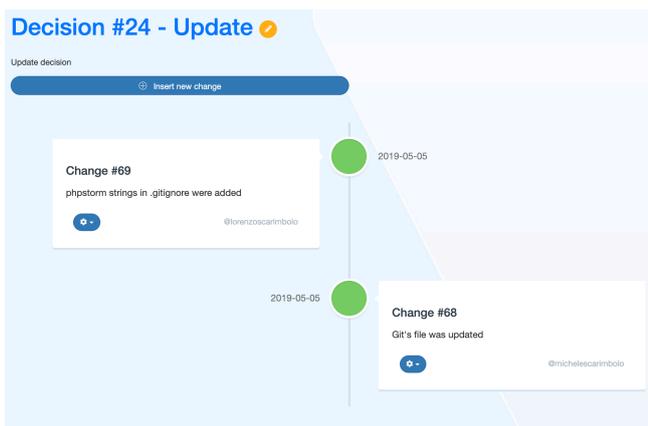
single branch, while Trello gives the possibility to connect a commit for each card or a branch for each card. This is not useful if more functionality is developed on a single branch, since each feature corresponds to a Kanban card.

The features in common between Trello and the software developed are the following:

- Create a card with name, description and membership list



- Leave comments (in developed software are called Changes)



- Collaborate with multiple users in the same project
- Attach File

Asana

Asana is a software-as-a-service designed to improve team collaboration and work management. It helps teams manage projects and tasks in one tool. Teams can create projects, assign work to teammates, specify deadlines, and communicate about tasks directly in Asana. It also includes reporting tools, file attachments, calendars, and more.

In May 2013, Asana launched Organizations, a way for companies of all sizes to use Asana, reporting tools to help teams monitor project progress, and IT admin tools.

In 2014, Asana launched Calendar View for projects and tasks, its native iOS app, and Dashboards.

In January 2015, Asana released its native Android app. Later that year, the company added team conversations. In September 2015, Asana unveiled a completely redesigned application and brand.

In 2016, Asana added administrator features including member management, team management, and password and security controls. Then, status updates were added so teams could communicate the state of a project to stakeholders, and task dependencies followed in July 2016. In September 2016, the company launched custom fields, “an interface and architecture that will let you tailor Asana’s information management to cover a variety of structured data points”. A few months later, Asana launched Boards so teams could organize and visualize their projects in columns. The Verge reported that, “By integrating lists and boards into a single product, Asana may have just vaulted ahead of its rivals.” The company also released pre-made project templates.

In March 2017, Asana announced its integration with Microsoft Teams, followed by the launch of custom project templates in June. In fall 2017, start dates, a new integration with Gmail, and comment-only projects were released. Also in November, Asana launched its app in French and German.

At the beginning of 2018, Asana launched a new CSV importer so teams could upload their data into the app. In February 2018, the app was released in Spanish and Portuguese. In March 2018, Asana announced a new interactive feature called Timeline, which businesses can use to visualize and map out their projects. [14]

Comparison

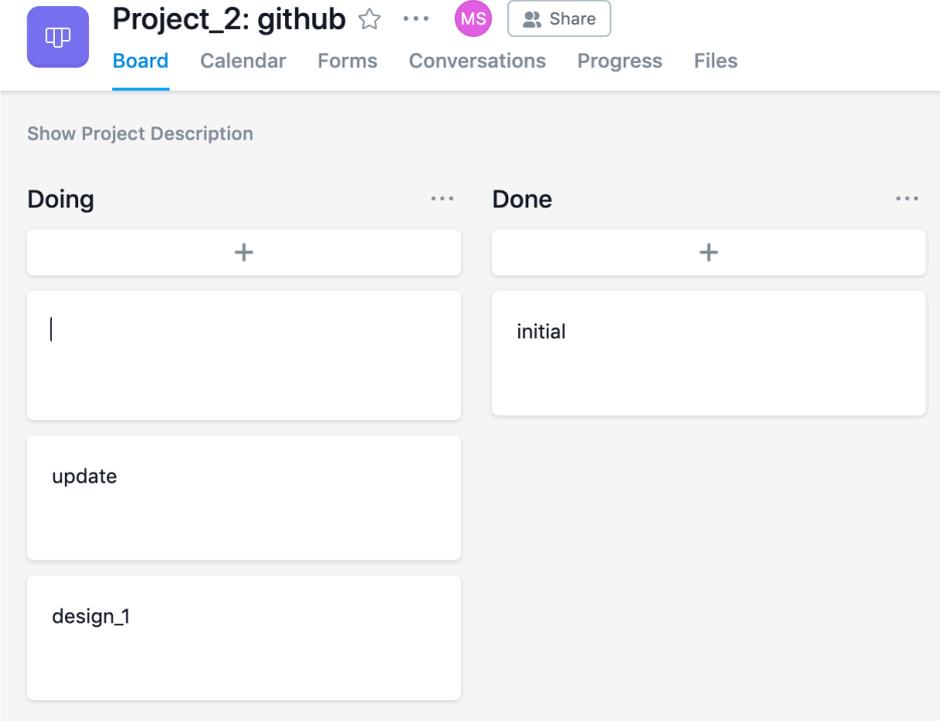


Figure 16 - Kanban board in Asana

Asana provides the ability, like the previous ones, to administer card decisions as a board in Kanban development.

For each decision it is possible to add a name, a description and attach any files. Here too there is the possibility of enabling a collaboration in the whole project to assign tasks to different users with different roles.

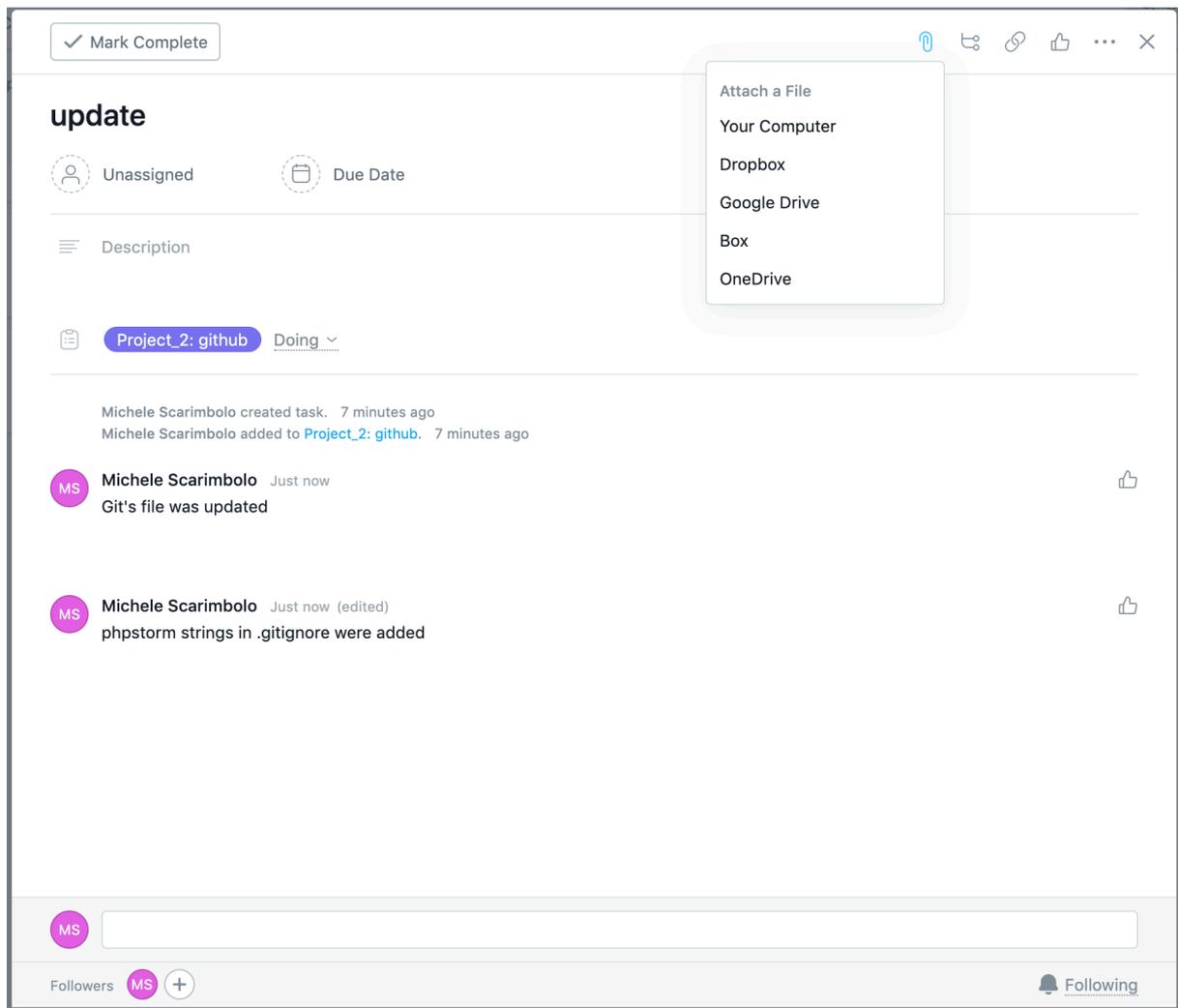


Figure 17 - Decision management in Asana

As in the previous ones, we find a similar administration of decisions, with the possibility of adding attachments, assigning tasks and deadlines and above all to keep track of the activity of what is being done (such as Changes or comments for Trello).

In relation to the Git system, Asana offers the possibility of adding the Github extension, but unlike the developed software that gives a single repository, with only one branch, it filters the commits based on the decisions you have in the project, Asana offers you the views of all the commits in the added Github repository, as in figure 18.

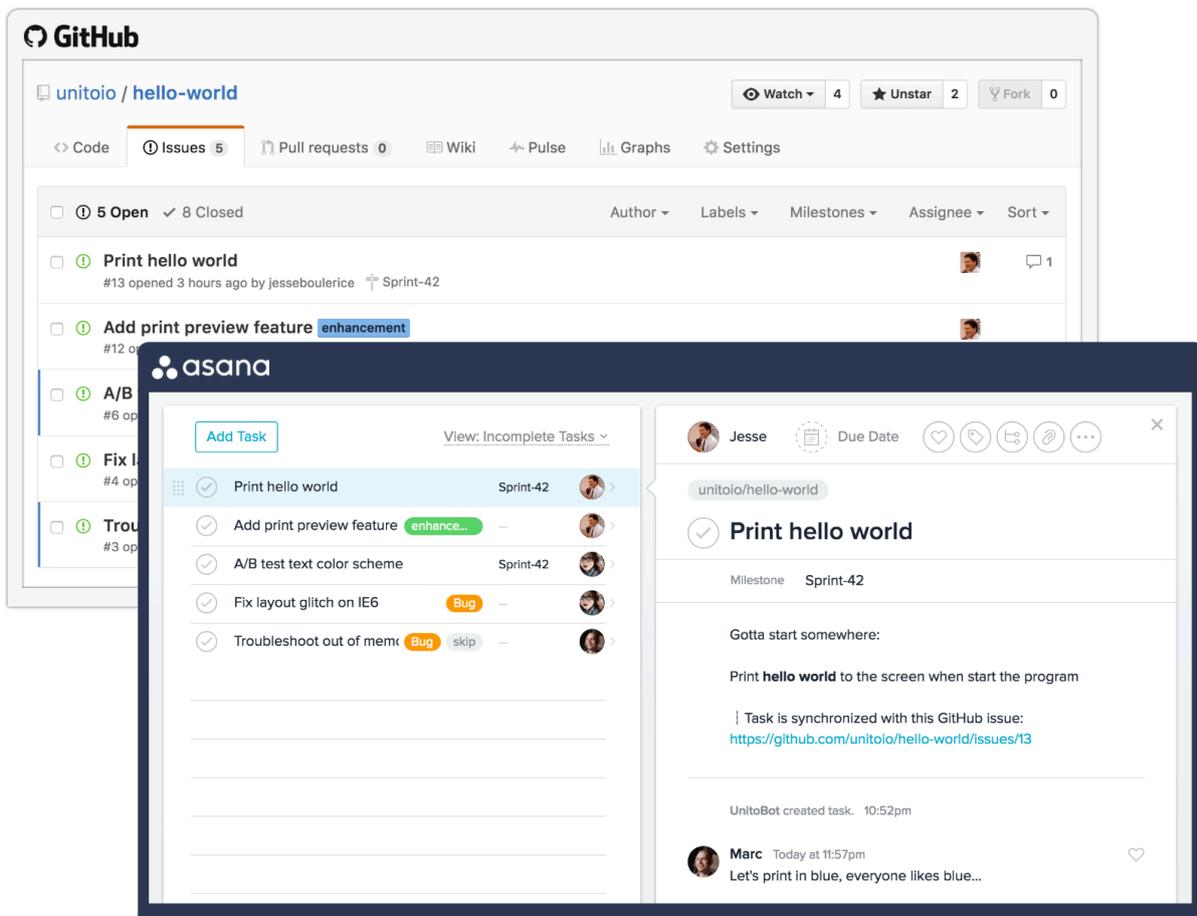


Figure 18 - Asana + Github [16]

So it's just a listing of what happens in the Github repository, without splitting commits based on what happens decision by decision.

Asana vs Trello

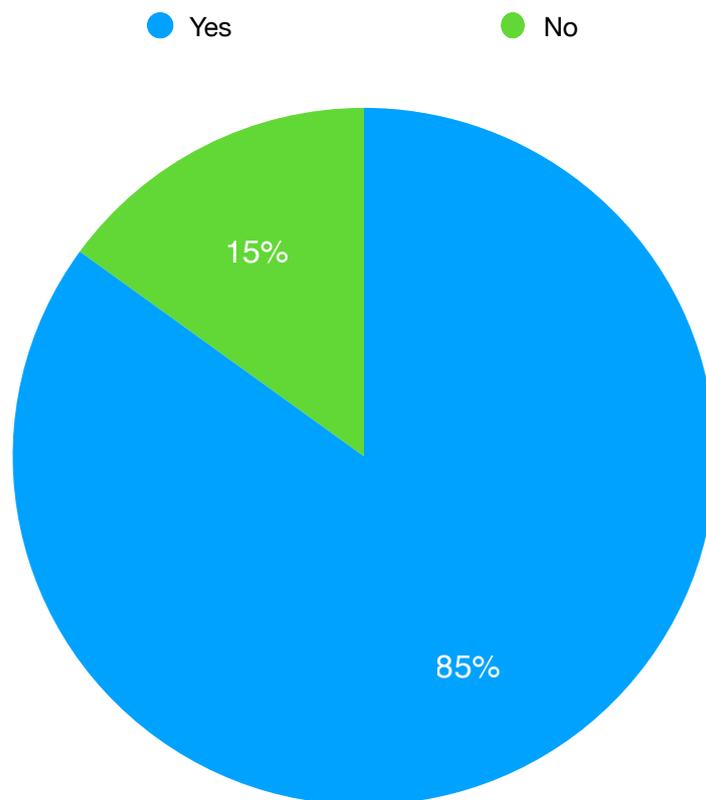
Asana	Trello
Free or zero pricing for the basic service	Activity feed
Quick overview on front and back of cards	Add assignees, attachments, and hearts to tasks
Easy organization with tags, labels and Categories	Automatic updates to email/inbox
Drag and drop functionality	Create custom calendars and views
In-line editing	Email bridge
Checklists, with progress meter	My Tasks list and Focus Mode
Easy uploading of files and attachments	Track tasks and add followers
Data filtering	Get notifications and reminders
Archiving of card records (e.g. comments and changes)	iPhone support, HTML5 mobile site
Deadline reminders	Multiple workspaces
Email notifications	Project Sections and Search Views
Activity log	Real-time updates
Assign tasks	See team members' tasks and priorities
Voting feature	Set goals, priorities, and due dates
Information retrieval and back-up	Set project permissions
SSL encryption of data	Project and task creation
Texts and visuals fit any screen size	Comment on tasks
Search function	Task dependencies
Mobile functionality to access boards on the go	Gantt Charts (Asana has Timeline)
developer API	Kanban support (Asana has boards)

Table 2 - benchmarking between Asana and Trello software [15]

III. Results and future developments

The developed application has made it possible to create a system that perfectly integrates the utility of a project management tool with the possibility of following, step by step, what the developers are doing for each single function designed.

It has been tried out in some work groups, university colleagues and friends and 85% of whom have developed a great desire to use it every day, while the remaining part has preferred to continue using existing tools (understandable for large projects, now already initiated).



Continuous engineering in our software

As a first step, after the release of the application, we will integrate the Continuous Engineering system.

To integrate this system it will be necessary to have a cloud computing system, such as AWS, Azure or the one that provides GitHub (which we already use). The idea is as follows: while the developers build the project, the CI / CD system acts independently at each commit or every n-day, creating a new version of the application that can be used by the end user.

This will avoid having to make a huge merge every time you want to release the application, especially if an application is very large with several branches or with many design decisions.

Below is an example of how the Github system for CI / CD works, called GitLab:

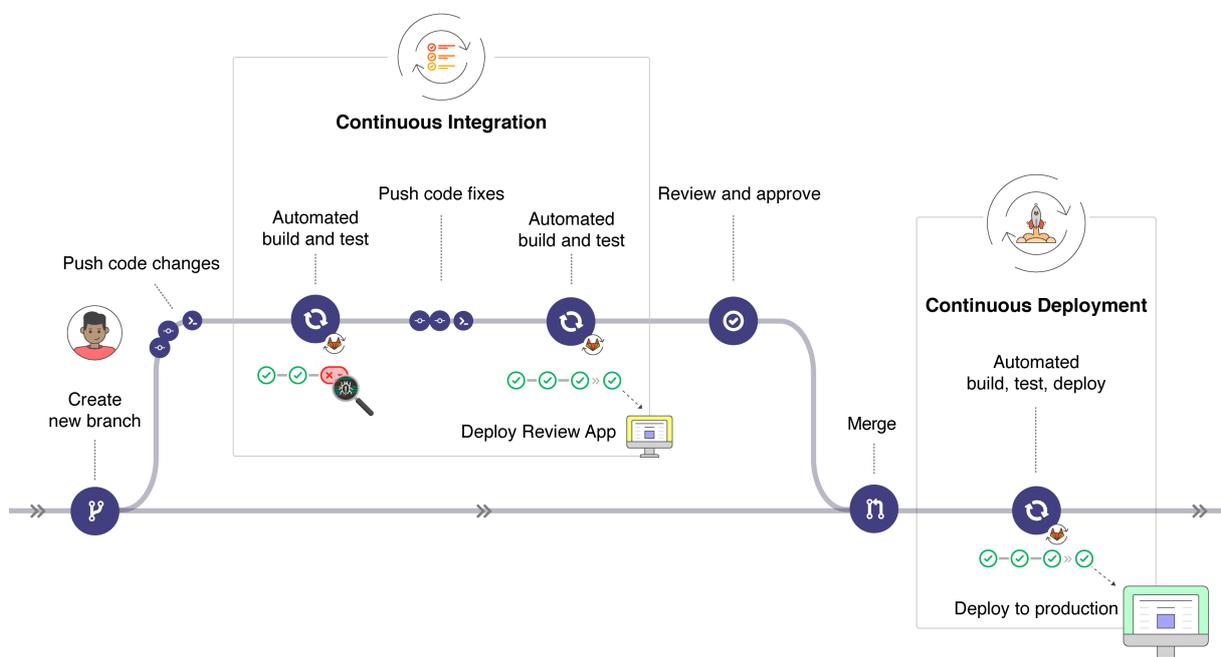


Figure 19 - How does continuous engineering of Github work

Using GitLab for CI/CD

To use GitLab CI/CD, all you need is an application codebase hosted in a Git repository, and for your build, test, and deployment scripts to be specified in a file called `.gitlab-ci.yml`, located in the root path of your repository.

In this file, you can define the scripts you want to run, define include and cache dependencies, choose commands you want to run in sequence and those you want to run in parallel, define where you want to deploy your app, and specify whether you will want to run the scripts automatically or trigger any of them manually. Once you're familiar with GitLab CI/CD you can add more advanced steps into the configuration file.

To add scripts to that file, you'll need to organize them in a sequence that suits your application and are in accordance with the tests you wish to perform. To visualize the process, imagine that all the scripts you add to the configuration file are the same as the commands you run on a terminal in your computer.

Once you've added your `.gitlab-ci.yml` configuration file to your repository, GitLab will detect it and run your scripts with the tool called GitLab Runner, which works similarly to your terminal.

The scripts are grouped into **jobs**, and together they compose a **pipeline**. A minimalist example of `.gitlab-ci.yml` file could contain:

```
before_script:
  - apt-get install rubygems ruby-dev -y
run-test:
  script:
    - ruby --version
```

The `before_script` attribute would install the dependencies for your app before running anything, and a **job** called `run-test` would print the Ruby version of the current system. Both of them compose a **pipeline** triggered at every push to any branch of the repository.

GitLab CI/CD not only executes the jobs you've set, but also shows you what's happening during execution, as you would see in your terminal:

The screenshot shows the GitLab CI/CD interface. On the left is a sidebar with navigation options like Project, Repository, Issues, Merge Requests, CI/CD, Pipelines, Jobs, Schedules, Charts, Operations, Registry, Snippets, and Settings. The main area displays a terminal window for a job named 'review-build-cng'. The terminal output shows the runner version (11.8.0), the image used (ruby:2.5-alpine), and the execution of various system packages like nhttpd, libssh2, libcurl, curl, oniguruma, jq, and openssl. It also shows the installation of gitlab and the execution of a script 'review_apps/review-apps.sh'. On the right, a job status panel shows the job name, duration (20 minutes 30 seconds), timeout (1h 40m), runner information, and a list of jobs in the pipeline, including 'danger-review', 'db:check-schema-pg', 'db:migrate:reset-mysql', 'db:migrate:reset-pg', 'db:rollback-mysql', and 'db:rollback-pg'.

You create the strategy for your app and GitLab runs the pipeline for you according to what you've defined. Your pipeline status is also displayed by GitLab:

The screenshot shows the pipeline status view in GitLab. It lists three jobs:

- failed**: Job #49651920 by user 'use-fallbac...' with commit '3c50e19e'. It failed at 00:14:45, 24 minutes ago.
- running**: Job #49649365 by user 'feature/add...' with commit 'fef3c0a8'. It is currently running, labeled 'latest', and is fixing a 'RuboCop offense'.
- passed**: Job #49647437 by user 'master' with commit '40ec172f'. It passed at 00:54:04, 18 minutes ago, and is a 'Merge branch 'web-ide-defa...'.

At the end, if anything goes wrong, you can easily roll back all the changes: [17]

The screenshot shows the deployment view for the 'production' environment. It includes buttons for 'View deployment', 'Monitoring', 'Edit', and 'Stop'. Below is a table of deployments:

ID	Commit	Job	Created
#20270	Y master -> c187a1a Merge branch 'untag-anchor-job' into 'master'	pages (#169250636) by [user]	1 hour ago
#20269	Y master -> c187a1a Merge branch 'untag-anchor-job' into 'master'	pages (#169215222) by [user]	1 hour ago

At the bottom right, there is a 'Rollback environment' button with a dropdown menu.

Git and social developments

Git: a feature we want to develop is the ability to provide the user who creates a project, to choose the Git server that he prefers (SourceTree, BitBucket, etc.).

When you insert a new project, you will choose a server, among those present on the platform (already pre-installed), entering all the specifications that that type of server requires to return the data (ex commits); otherwise you could have a new git server inserted with all the required address specifications. Obviously this possibility could influence the use of Github's CI / CD, so also this functionality will undergo changes giving the opportunity, also in the Continuous Engineering settings, to choose which server to use and to give, in input to this server, addresses and properties of the git repositories chosen during project creation.

Social: This last future development is related to a section of Social communication between users:
chat (single chat or group chat).

create development groups with custom tags

The general idea is to create features that are used by the Slack software, already exploited in many large software development situations.

With all these developments, we will try to combine most of the features provided by so many software, in a single solution, making the development experience of medium / large design and software development companies easier, faster and more interactive.

IV. Bibliografy

[1] Stapleton J., "DSDM Dynamic Systems Development Method: The Method in Practice", Addison Wesley, 1997

[2] Toward Architectural Knowledge Sustainability: New Opportunities to Extend the Longevity of Systems, Rafael Capilla, Elisa Yumi Nakagawa, Uwe Zdun, Carlos Carrillo

[3] <https://git-scm.com/about>

[4] R. Tim, S. Tanachutiwat, M. Vukadinovic, H. Schlebusch and H. Lichter, "Continuous integration processes for modern client-side web applications," *2017 International Electrical Engineering Congress (iEECON)*, Pattaya, 2017, pp. 1-4.

[5] M. Shahin, M. Ali Babar and L. Zhu, "Continuous Integration, Delivery and Deployment: A Systematic Review on Approaches, Tools, Challenges and Practices," in *IEEE Access*, vol. 5, pp. 3909-3943, 2017.

[6] <https://getbootstrap.com/>

[7] <https://jquery.com/>

[8] <https://httpd.apache.org/>

[9] <http://www.slimframework.com/>

[10] <https://www.mountaingoatsoftware.com/agile/scrum>

[11] <https://developer.github.com/v3/repos/commits/>

[12] M. Ikonen, E. Pirinen, F. Fagerholm, P. Kettunen and P. Abrahamsson, "On the Impact of Kanban on Software Project Work: An Empirical Case Study Investigation," *2011 16th IEEE International Conference on Engineering of Complex Computer Systems*, Las Vegas, NV, 2011, pp. 305-314.

[13] D. Parsons, R. Thorn, M. Inkila and K. MacCallum, "Using Trello to Support Agile and Lean Learning with Scrum and Kanban in Teacher Professional Development," *2018 IEEE International Conference on Teaching, Assessment, and Learning for Engineering (TALE)*, Wollongong, NSW, 2018, pp. 720-724.

[14] [https://en.wikipedia.org/wiki/Asana_\(software\)](https://en.wikipedia.org/wiki/Asana_(software))

[15] <https://comparisons.financesonline.com/asana-vs-trello>

[16] <https://asana.com/apps/github>

[17] <https://docs.gitlab.com/ee/ci/introduction/index.html#how-gitlab-cicd-works>