

Conduite de projet : Outils et processus pour le développement logiciel collaboratif

Aldric Degorre
(IRIF, U-Paris) – adegorre@irif.fr
(d'après transparents originaux de Yann Régis-Gianas)

29 septembre 2023

Plan

Retour sur les deux semaines précédentes

Travailler sur l'historique avec git

Développement collaboratif avec GitLab

Méthodologie de collaboration “git-flow”

Processus de développement

- ▶ La différence entre **programmer** et **développer**.
- ▶ La spécificité de la conduite d'un projet logiciel.
- ▶ Les méthodes Agile.
- ▶ Les outils pour le développement collaboratif, notamment git.

En travaux dirigés

Après sondage :

- ▶ seules 17 équipes (\simeq 100 étudiants) m'ont ajouté à leur projet gitlab → la moitié d'entre vous n'a pas créé de projet ou ne sait pas suivre de consigne?
- ▶ 1 équipes a créé son projet mais n'a pas travaillé depuis¹
- ▶ 12 équipes ont laissé une trace visible mais triviale (1 issue ou moins, peu de commits)
- ▶ Seulement 4 équipes ont travaillé!
Parmi elles, pas une seule n'a fait de merge request.
1 seule a assigné des issues à tous ses membres!!!²

1. Pas de façon visible et, comme dit le proverbe, « Si ce n'est pas sur GitLab, ça ne s'est pas produit! »

2. Dans un projet en bonne santé : il faut un backlog, des tâches assignées, des tâches en cours de résolution... et même des merge requests!

En travaux dirigés



En travaux dirigés

- ▶ Certains ne sont pas venus ou alors en retard :
 - 2 absences injustifiées = cours non validés
 - Trop de retards (>5 minutes) = 3 points de moins
- ▶ Tout membre de l'équipe doit travailler sur une tâche, et une "bonne" tâche.
- ▶ Une "bonne" tâche :
 - ▶ est clairement formulée;
 - ▶ produit un résultat (document, code, ...) évaluable;
 - ▶ est une activité dont on peut estimer la durée;
 - ▶ est affectée à une personne mais est la responsabilité de toute l'équipe.
- ▶ Pour l'analyse de l'existant, vous aviez 90 minutes.³

3. Plus ou moins, en fonction de la vitesse de résolution des problèmes de mise en place...

En travaux dirigés

- ▶ Certains ne sont pas venus ou alors en retard :
 - 2 absences injustifiées = cours non validés
 - Trop de retards (>5 minutes) = 3 points de moins
- ▶ Tout membre de l'équipe doit travailler sur une tâche, et une "bonne" tâche.
- ▶ Une "bonne" tâche :
 - ▶ est clairement formulée;
 - ▶ produit un résultat (document, code, ...) évaluable;
 - ▶ est une activité dont on peut estimer la durée;
 - ▶ est affectée à une personne mais est la responsabilité de toute l'équipe.
- ▶ Pour l'analyse de l'existant, vous aviez 90 minutes.³
- ▶ Certains ont dit que c'était trop court mais :

3. Plus ou moins, en fonction de la vitesse de résolution des problèmes de mise en place...

En travaux dirigés

- ▶ Certains ne sont pas venus ou alors en retard :
 - 2 absences injustifiées = cours non validés
 - Trop de retards (>5 minutes) = 3 points de moins
- ▶ Tout membre de l'équipe doit travailler sur une tâche, et une "bonne" tâche.
- ▶ Une "bonne" tâche :
 - ▶ est clairement formulée;
 - ▶ produit un résultat (document, code, ...) évaluable;
 - ▶ est une activité dont on peut estimer la durée;
 - ▶ est affectée à une personne mais est la responsabilité de toute l'équipe.
- ▶ Pour l'analyse de l'existant, vous aviez 90 minutes.³
- ▶ Certains ont dit que c'était trop court mais :

$$6 \text{ membres} * 1.5 \text{ heures} = 9 \text{ heures} \cdot \text{membres}$$

3. Plus ou moins, en fonction de la vitesse de résolution des problèmes de mise en place...

En travaux dirigés

- ▶ Certains ne sont pas venus ou alors en retard :
 - 2 absences injustifiées = cours non validés
 - Trop de retards (>5 minutes) = 3 points de moins
- ▶ Tout membre de l'équipe doit travailler sur une tâche, et une "bonne" tâche.
- ▶ Une "bonne" tâche :
 - ▶ est clairement formulée;
 - ▶ produit un résultat (document, code, ...) évaluable;
 - ▶ est une activité dont on peut estimer la durée;
 - ▶ est affectée à une personne mais est la responsabilité de toute l'équipe.
- ▶ Pour l'analyse de l'existant, vous aviez 90 minutes.³
- ▶ Certains ont dit que c'était trop court mais :

$$6 \text{ membres} * 1.5 \text{ heures} = 9 \text{ heures} \cdot \text{membres}$$

- ▶ Etait-ce si déraisonnable ?

3. Plus ou moins, en fonction de la vitesse de résolution des problèmes de mise en place...

Pour la suite

- ▶ N'oubliez pas de rajouter vos chargés de TD (et moi-même) dans les membres du projet Gitlab.
- ▶ Chaque séquence de 2 semaines sera l'objet d'un **sprint**.
- ▶ La séance de TD type :
 - ▶ 1/4 d'heure : bilan du dernier sprint, comment s'améliorer ?
 - ▶ 1h45 : préparation du sprint suivant.
- ▶ Vous devrez produire une vidéo de présentation de votre projet.
- ▶ Rappel : l'évaluation est continue.

Séance de cours d'aujourd'hui

1. Outils collaboratifs :
 - ▶ Git (suite)
 - ▶ et GitLab.
2. Présentation de Scrum.

Plan

Retour sur les deux semaines précédentes

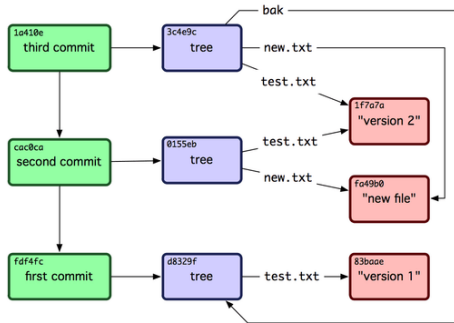
Travailler sur l'historique avec git

Développement collaboratif avec GitLab

Méthodologie de collaboration “git-flow”

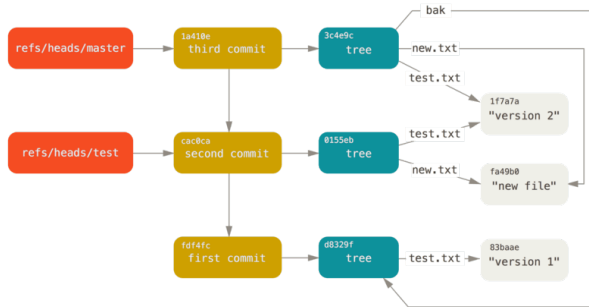
Processus de développement

Quelques rappels



- ▶ **blob** : Contenus de fichiers sauvegardés par Git.
- ▶ **tree** : Arborescence de fichiers.
- ▶ **commit** : Description d'un changement.
- ▶ **sha1** : Identificateur unique pour tout objet stocké par Git.
- ▶ **branch** : Séquence de commits caractérisée par un nom.

Notion de références



- Mémoriser des **sha1**, c'est difficile!
- **référence** : nom associé à une arborescence.
- Exemples : **master**, **HEAD**, ...

Index, HEAD et arborescence de travail

- ▶ Quand on se déplace dans un dépôt Git, en dehors du répertoire **.git**, on se trouve dans **l'arborescence de travail** (ou "working tree", appelée parfois "arborescence courante").
- ▶ Cette arborescence peut contenir des fichiers modifiés dont les changements n'ont pas encore été pris en compte par Git.
- ▶ Le rôle de l'**index** est de sauvegarder les changements à inclure dans le prochain commit : c'est une sorte de zone de préparation ("staging").
- ▶ **HEAD** est une référence qui pointe vers la branche courante, qui elle-même pointe vers son commit le plus récent.

Lire : https://git-scm.com/book/fr/v2/Utilitaires-Git-Reset-d%C3%A9mystifi%C3%A9s_git_reset

git reset vs git checkout

git reset [<unCommit>]

Fait pointer la branche pointée par HEAD vers **unCommit**.

- ▶ **git reset --soft** : Ne fait rien de plus que cela.
(Notamment : ni l'arborescence de travail ni l'index ne sont modifiés.)
- ▶ **git reset --mixed** (action par défaut) : remplace en plus l'index par le contenu de HEAD.
- ▶ **git reset --hard** : remplace en plus l'arborescence de travail par le contenu de l'index.

Remarques :

- ▶ l'argument [<unCommit>] est facultatif⁴. Par défaut c'est la branche pointée par HEAD (→ **git reset --soft** tout seul ne fait rien).
- ▶ ces commandes font juste ce qui a été dit. Ainsi un fichier qui serait devenu différent entre HEAD et l'arborescence de travail n'est pas automatiquement validé dans l'index (**git status** lui donnera le statut M).

4. C'est la signification des crochets [] dans la méta-syntaxe.

git reset vs git checkout

`git reset [<unCommit>]`

Fait pointer **la branche pointée par HEAD** vers `unCommit`.

`git checkout [<unCommitOuBranche>]`

Déplace HEAD vers le commit ou la branche passée en paramètre.

`git checkout` **ne déplace en aucun cas la branche** anciennement pointée par HEAD!

- ▶ Si [<unCommitOuBranche>] n'est pas spécifié, HEAD reste sur la branche courante.
- ▶ Le contenu de l'arborescence de travail (sauf modifications locales) et de l'index deviennent identiques à celui de [<unCommitOuBranche>].
- ▶ Si [<unCommitOuBranche>] prend la forme d'un sha1 (donc un commit quelconque), HEAD n'est alors plus associé à une branche. On parle de HEAD '**détachée**'.

`git reset -- file` vs `git checkout -- file`

`git reset -- file`

Pour annuler un `git add`...

- ▶ `git reset -- file` copie `file` de HEAD vers index.
- ▶ `git reset sha1 -- file` copie `file` de `sha1` vers index.

Contrairement à la forme sans "`-- file`", cette commande ne déplace pas la branche pointée par HEAD.

git reset -- file vs git checkout -- file

git reset -- file

Pour annuler un git add...

- ▶ `git reset -- file` copie `file` de HEAD vers index.
- ▶ `git reset sha1 -- file` copie `file` de sha1 vers index.

Contrairement à la forme sans "`-- file`", cette commande ne déplace pas la branche pointée par HEAD.

git checkout -- file

Pour restaurer un fichier dans l'arborescence de travail...

- ▶ `git checkout -- file` copie `file` de HEAD vers index et l'arborescence de travail.
- ▶ `git checkout sha1 -- file` copie `file` de sha1 vers index et l'arborescence de travail.

Contrairement à la forme sans "`-- file`", cette commande ne déplace pas HEAD.

git revert

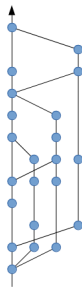
```
1 $ git revert [commit-sha1]
```

- ▶ Tout ce que nous venons de voir fonctionne **sur votre copie locale**.
- ▶ Si vous avez déplacé une branche...
... `git push -f` permettrait de pousser ces changements d'historique locaux sur un dépôt partagé, **mais...**
- ▶ **... c'est une mauvaise pratique** car cela pourrait **casser les historiques de vos collaborateurs!**
(On peut le faire dans des cas très spécifiques, mais c'est une opération très délicate!)
- ▶ La seule situation où un `git push -f` a du sens est la mise à jour de la branche d'une MR.
- ▶ Quand on souhaite annuler un commit déjà publié, il ne faut pas réécrire l'histoire mais simplement créer un commit qui annule ce commit. C'est ce que fait `git revert`.

git rebase

```
1 $ git rebase [ref]
```

Non-linear history



Linear history



Voir <https://git-scm.com/book/fr/v2/Les-branches-avec-Git-Rebaser-Rebasing>

git rebase

```
1 $ git rebase [options]
```

Cette commande permet de réécrire le sommet de l'historique d'une branche (source) sur une autre (cible).

- ▶ les commits de la source inconnus de la cible⁴ sont mis dans une zone d'attente
- ▶ la branche courante est déplacée vers la cible
- ▶ les commits en attente sont réappliqués par dessus, un à un, dans l'ordre (cela est fait automatiquement si possible, sinon, stratégies de fusion. Par exemple : `git rebase -i` demande **intéreactivement** à l'utilisateur).

Bien lire l'aide pour comprendre les options!

4. C'est-à-dire ultérieurs à leur point de divergence.

git pull --rebase

```
1 $ git pull --rebase
```

- ▶ Par défaut, `git pull = git fetch; git merge`
Résultat : historique = 2 séquences parallèles (une avec les commits du **f**etch, l'autre avec nos commits) qui divergent puis convergent (lors du commit de **m**erge).
- ▶ Il faut plutôt lui préférer `git pull --rebase` pour éviter des historiques non linéaires.
En effet : `git pull --rebase = git fetch; git rebase`.
Résultat : historique linéaire contenant les commits distants (de **f**etch), **suivis de** nos commits (remaniés).

git, autres situations, cas pratiques, etc.

- ▶ Pour l'instant nous n'avons qu'effleuré les commandes existant dans git.
- ▶ Il serait indigeste de tout présenter dans ces transparents.
- ▶ Cependant, n'hésitez pas à demander de l'aide à vos enseignants pour telle ou telle situation. Un wiki⁵ est déjà disponible sur le gitlab du cours, proposant des fiches pratiques.

5. <https://gaufre.informatique.univ-paris-diderot.fr/cproj2022/cours/wikis/home>

Plan

Retour sur les deux semaines précédentes

Travailler sur l'historique avec git

Développement collaboratif avec GitLab

Méthodologie de collaboration “git-flow”

Processus de développement



GitLab

GitLab est une application web pour la collaboration autour d'un dépôt git.

Notion de projet Gitlab

L'aspect décentralisé de Git a de nombreux avantages en termes de flexibilité et de résilience. Cependant, si plusieurs développeurs souhaitent collaborer en travaillant sur un même dépôt, il faut leur donner accès à un dépôt de référence. C'est ce problème que résout Gitlab.

En plus de cela, Gitlab fournit des outils de **gestion de projet** :

1. Contrôle d'accès sur le dépôt.
2. Système de tickets, pour documenter les bugs, les évolutions,
3. Système de contrôle et de revue de "merge requests" (voir transparent suivant).
4. Système d'intégration continue.
5. Système de notifications pour être alerté des événements d'un projet.

“Demandes de fusion” ou “Merge requests” (ou “Pull requests”⁶)

- ▶ Tout changement fait sur le code source d'un projet doit être documenté et chaque développeur doit en avoir conscience et le comprendre.
- ▶ À éviter : faire une multitude de petits commits dans son coin et les pousser directement sur le dépôt du projet (sans que personne ne contrôle la qualité ni ne comprenne l'objectif de ces changements).
- ▶ → **bonne pratique : ne jamais pousser directement des changements** sur un dépôt de référence d'un projet.
 - étape intermédiaire : soumission d'une “merge request” (MR).
- ▶ Soumettre une MR permet aux autres membres de l'équipe de **relire** les changements pour les comprendre et faire des suggestions d'améliorations.
 - C'est seulement lorsque la MR a été validée par suffisamment de membres de l'équipe qu'elle est effectivement intégrée au dépôt.

6. “Pull request” est la terminologie du site GitHub, concurrent de GitLab. Elle est très couramment utilisée. Ainsi on dira souvent PR au lieu de MR sans y faire attention...

MR dans GitLab

The screenshot displays the GitLab web interface for creating a new merge request. The top navigation bar includes the GitLab logo and links to Projects, Groups, Activity, Milestones, Snippets, and a search bar. The left sidebar shows the project navigation menu with options like Project, Repository, Issues, Merge Requests, CI / CD, Operations, Wiki, Snippets, and Settings. The main content area is titled 'New Merge Request' and is divided into two panels: 'Source branch' and 'Target branch'.

Source branch

Source branch: yrg/majae, document-usage

Populate README
Yann Regis-Gianas authored 2 days ago

df4d609a

Target branch



Target branch: yrg/majae, master


Initial commit
Yann Regis-Gianas authored 2 days ago


0b1dbc2f

Compare branches and continue

MR dans GitLab


 Request to merge `document-usag` into `develop`Open in Web IDECheck out branch


 Merge


 Delete source branch

> 1 commit and 1 merge commit will be added to `develop`. [Modify merge commit](#)

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

 0

 0




Discussion 5


Commits 1


Changes 1

Show all activity ▾





0/4 discussions resolved






 Yann Regis-Gianas @yrg · 2 days ago

Maintainer


@adegorre @klimann Does that PR match our discussion?

 KLIMANN INES @klimann started a discussion on the diff 1 day ago





Toggle discussion

README.md

```
21 + 2. The assignment: After the deadline, the students choices are
22 +   processed by an algorithm which decides the task assignment.
23 +
24 + 3. The reporting: The execution trace of the assignment algorithm
25 +   is described in a report, so that each participant can check that
26 +   its execution has been flawless and conform to its specification.
27 +
28 + ## What is the assignment algorithm used by maljae?
29 +
30 + This part of the document is still work-in-progress.
31 +
32 + ## How maljae is supposed to be used?
33 +
34 + maljae is made of two main components: a webserver to collect the team
35 + information and a command-line tool to implement the task assignment
36 + algorithm.
```

 KLIMANN INES @klimann · 1 day ago

Maintainer

What about the third component to show the subjects attribution?

Fork de projet Gitlab

Gitlab favorise le développement collaboratif en autorisant n'importe qui à créer une copie⁷ d'un projet. Cette opération s'appelle un fork dans le jargon de Gitlab ("divergence", dans l'interface en français).

Un fork permet à toute personne extérieure aux membres d'un projet Gitlab de proposer des contributions. Pour cela, il lui suffit, après avoir créé de fork et y avoir ajouté sa contribution, de créer une MR pour la proposer aux membres du projet d'origine ("upstream project")⁸.



7. Je n'utilise volontairement pas le mot "clone", que l'on réservera à l'opération consistant à copier un dépôt git. Quand on fait un fork, le dépôt du nouveau projet est, certes, cloné depuis le projet d'origine, mais on crée aussi un projet complet, pas seulement un dépôt.

8. Accepter la fusion consiste alors, pour le projet upstream à faire un "pull" du dépôt de son fork, d'où la terminologie "pull request" utilisée sur GitHub.

Gestion de tickets avec Gitlab

Gitlab permet de créer des **tickets**. Un ticket documente un problème à résoudre sur le projet : cela peut être une erreur à corriger dans le code, une fonctionnalité à rajouter, ...

Les tickets peuvent **être référencés** par les MR, dans les discussions ou dans les outils de gestion de projet. Cela améliore la traçabilité du projet.

Les tickets peuvent être **ouverts** (à résoudre) ou **fermés** (résolus).

Les tickets peuvent aussi être catégorisés par des étiquettes.



The screenshot shows the 'Issues' page for the 'maljae' project by Yann Regis-Gianas. The page has a header with the project name and a breadcrumb trail. Below the header, there are filters for 'Open' (4), 'Closed' (0), and 'All' (4). To the right of the filters are icons for RSS, a calendar, and a refresh button, along with buttons for 'Edit issues' and 'New issue'. A search bar with a magnifying glass icon and the text 'Search or filter results...' is present. To the right of the search bar is a dropdown menu for 'Created date' and a button with a list icon. The main content area displays a list of four issues, each with a title, a number, a status, and a date. The issues are: 'Design the slot assignment algorithm' (#4, 'Doing', updated 1 day ago), 'Design the project data model' (#3, 'To Do', updated 2 days ago), 'Implement a project skeleton' (#2, 'Doing', updated 2 days ago), and 'Document project usage' (#1, 'Doing', updated 2 days ago). Each issue has a status label ('Doing' or 'To Do') and a date ('opened 2 days ago by Yann Regis-Gianas').

Yann Regis-Gianas > maljae > Issues

Open 4 Closed 0 All 4

Search or filter results...

Created date

- Design the slot assignment algorithm
#4 · opened 2 days ago by Yann Regis-Gianas (Doing) updated 1 day ago
- Design the project data model
#3 · opened 2 days ago by Yann Regis-Gianas (To Do) updated 2 days ago
- Implement a project skeleton
#2 · opened 2 days ago by Yann Regis-Gianas (Doing) updated 2 days ago
- Document project usage
#1 · opened 2 days ago by Yann Regis-Gianas (Doing) updated 2 days ago

Le module tableau/board ou « kanban »

Gitlab Org > Issue Boards

Development ▾ Search or filter results... Show labels ☒ Group by None ▾ Edit board Create list

> Open 56 10 +

Milestones swimlanes
Doing dev plan
gitlab-org/gitlab-test#80 Jun 17 4

Assign issue to epic
To Do backend
gitlab-org/gitlab-test#45

Create group
To Do
gitlab-org/gitlab-test#78 3

Remove issue from board
To Do
gitlab-org/gitlab-shell#38

Add lists for assignees and milestones
To Do
gitlab-org/gitlab-shell#2

> Deliverable 32 8 +

Update issue due date from sidebar
To Do frontend
gitlab-org/gitlab-test#54

Update issue's labels from sidebar
To Do frontend
gitlab-org/gitlab-test#55

Update issue labels
Doing
gitlab-org/gitlab-test#75

Drag and drop issue between epics
To Do frontend
gitlab-org/gitlab-test#33

Paginate issues in Swimlanes
To Do
gitlab-org/gitlab-test#39

> Closed 59 2

Persist collapsed state of Swimlanes
Deliverable test test
Dec 30, 2020 1w 2
gitlab-org/gitlab-test#32

Remove list from board
Caliber Colorado Doing feature proposal
gitlab-org/gitlab-test#44

Remove issue from Swimlane
To Do frontend
gitlab-org/gitlab-test#36

Expand diff to entire file
Premium To Do dev manage frontend
gitlab-org/gitlab-test#25

Laboriosam commodi ab in eum qui suscipit necessitatibus modi fuga.
Deliverable frontend

Le module tableau/board ou « kanban »

Kanban = panneau d'affichage

Méthode Kanban :

- ▶ élaborée par David J. Anderson au début des années 2000
- ▶ inspirée du Kanban utilisé dans les usines Toyota depuis la fin des années 1940
- ▶ méthode d'organisation du travail basée sur un certain nombre de principes
- ▶ s'applique essentiellement au développement logiciel, elle fait partie des méthodes agiles

... mais finalement, concrètement, on en garde surtout le système visuel : le tableau « Kanban » qu'on utilisera aussi dans Scrum⁹.

9. En toute rigueur il faut alors parler de tableau Scrum.

Le module tableau/board ou « kanban »

- ▶ On y répartit des fiches Kanban, des post-its représentant des **unités de travail**.
- ▶ Divisé en au moins 3 colonnes : « en attente », « en cours » et « réalisé ».
Mais on peut détailler plus !
- ▶ L'ensemble des colonnes représentent ce qu'on appelle le **workflow**. Celui-ci dépend de la méthode d'organisation utilisée par l'équipe.
- ▶ Workflow possible pour Scrum :
 - ▶ « Stories » : tickets bruts, « user stories »
 - ▶ « ToDo » : items suffisamment détaillé/raffiné, sélectionnés pour le sprint en cours et assignés
 - ▶ « InProgress » : travail en cours de réalisation
 - ▶ « Testing » : travail réalisé mais pas complètement testé
 - ▶ « Done » : travail fini, testé (et intégré à la version de développement)

Autres points importants à venir

D'autres fonctionnalités de GitLab seront détaillées dans les cours suivants.

Plan

Retour sur les deux semaines précédentes

Travailler sur l'historique avec git

Développement collaboratif avec GitLab

Méthodologie de collaboration “git-flow”

Processus de développement

Comment organiser les branches ?



Nous aurons une discipline pour organiser les branches des dépôts :

- ▶ La branche “master” ou “main” pour des commits de “releases”.
- ▶ La branche “develop” pour des commits d’évolutions bien identifiables.
- ▶ Chaque développeur travaillant sur une évolution est dans une branche locale dédiée.

Cette structuration s’appelle le “git-flow” et est bien documentée ici :

[https:](https://www.atlassian.com/git/tutorials/comparing-workflows/gitflow-workflow)

[//www.atlassian.com/git/tutorials/comparing-workflows/gitflow-workflow](https://www.atlassian.com/git/tutorials/comparing-workflows/gitflow-workflow)

Plan

Retour sur les deux semaines précédentes

Travailler sur l'historique avec git

Développement collaboratif avec GitLab

Méthodologie de collaboration “git-flow”

Processus de développement

Quelques citations à discuter

"First do it, then do it right, then do it better."

- Addy Osmani

Quelques citations à discuter

"There's nothing more permanent than a temporary hack."
- Kyle Simpson

Quelques citations à discuter

"Weeks of coding can save you hours of planning."

- Unknown

Quelques citations à discuter

“Any fool can write code that a computer can understand. Good programmers write code that humans can understand.”

– Martin Fowler

Quelques citations à discuter

"In programming, the hard part isn't solving problems, but deciding what problems to solve."

- Paul Graham

Les méthodes Agile

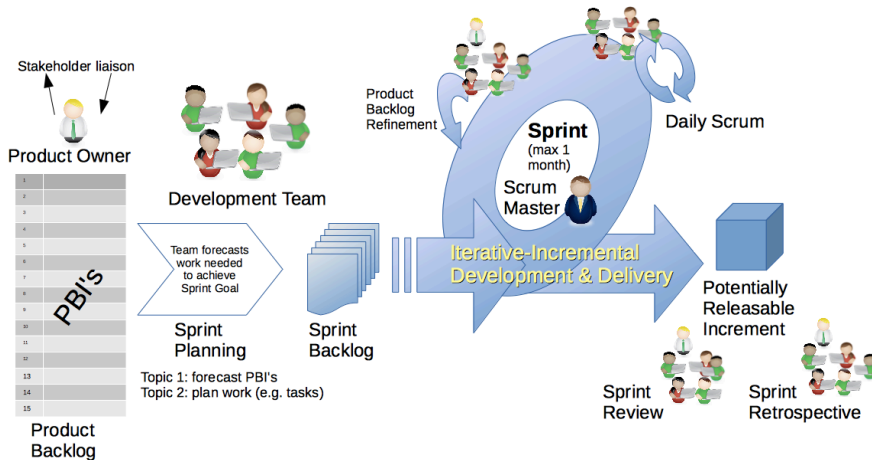
- ▶ Méthodes Agile = processus de développement de logiciel.
- ▶ Retour sur les principes des méthodes de cette famille.
- ▶ Scrum est l'une de ces méthodes.
- ▶ Autres méthodes : (selon Wikipedia)
 - ▶ RAD (Rapid Application Development)
 - ▶ XP (eXtreme Programming)
 - ▶ ASD (Adaptative Software Development)
 - ▶ FDD (Feature-Driven Development)
 - ▶ BDD (Behavior-Driven Development)
 - ▶ Crystal clear

Remarque : ces méthodes ne se consacrent pas forcément aux mêmes aspects du cycle de développement et peuvent parfois se compléter.

Scrum

- ▶ Processus incrémental et itératif.
- ▶ Organisés autour de **sprints**.
- ▶ Chaque sprint est :
 - ▶ limité dans le temps (de 2 à 4 semaines);
 - ▶ décomposé en une phase de **planification**, une phase de **développement**, une **revue** et une **rétrospective**;
- ▶ focalisé sur un **ensemble fixé d'objectifs** mais **intègre le raffinement** de ces objectifs (pour un sprint ultérieur).
- ▶ mené par une équipe **pluridisciplinaire** qui va **chercher le travail où il est**;
- ▶ engendre un **logiciel fonctionnel** (intégré, testé, documenté et déployable).

Scrum



Les rôles dans Scrum

- ▶ **Product owner** : schématiquement, c'est le client. Contrairement aux approches traditionnelles, le client co-développe le logiciel. Le client sait ce qui est important, ce qui a de la valeur.
- ▶ **Equipe** : elle est formée de ses membres ($7 + / - 2$), capables de prendre en charge tous les aspects du projet.
- ▶ **Scrum Master** : il aide l'équipe à bien fonctionner en lui rappelant les principes de Scrum. Le Scrum Master partage son expertise et son expérience en développement logiciel et a un point de vue extérieur sur le projet.

“Product backlog”

- ▶ **Product backlog** : liste **ordonnée** d'éléments (évolutions, développements interne, explorations, corrections de bugs).
- ▶ Il définit ce qu'est le projet et ce qui ne l'est pas.
- ▶ Il caractérise ce qui est important et ce qui l'est moins.

Comment bien définir le backlog?

- ▶ Le niveau de détails doit dépendre de la proximité de la date de développement : plus on se rapproche du moment où l'élément devra être réalisé et plus on l'a décomposé en tâches élémentaires bien définies.
- ▶ Le produit d'une tâche doit être objectivement vérifiable.
- ▶ La durée d'une tâche doit être estimable.
- ▶ Les tâches doivent être priorisées : les plus importantes et les plus risquées viennent en premier, puis les plus importants et les moins risquées, et les autres.

Comment bien définir le backlog?

- ▶ Le niveau de détails doit dépendre de la proximité de la date de développement : plus on se rapproche du moment où l'élément devra être réalisé et plus on l'a décomposé en tâches élémentaires bien définies.
- ▶ Le produit d'une tâche doit être objectivement vérifiable.
- ▶ La durée d'une tâche doit être estimable.
- ▶ Les tâches doivent être priorisées : les plus importantes et les plus risquées viennent en premier, puis les plus importants et les moins risquées, et les autres.

**Plus l'équipe se connaît, plus le projet se précise
et meilleur est le backlog.**

Comment estimer le temps de réalisation d'une tâche ?

- ▶ Maintenir un historique sur l'efficacité de l'équipe.
- ▶ Observer les Sprint burndown chart : graphique indiquant pour chaque jour d'un sprint, le nombre de tâches restant à effectuer.
- ▶ Poker planning scrum pour chaque tâche :
 1. Les membres de l'équipe discutent de la nature et de la difficulté de la tâche.
 2. Chaque membre de l'équipe écrit sur un papier (caché) une valeur prise dans l'ensemble : 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, « ∞ » et « ? »
 3. On retourne le papier : tant que l'on n'a pas atteint l'unanimité, on reprend en 1.
- ▶ Le rythme doit rester **soutenable**.

Utilisation d'un kanban



- C'est le moment de s'aider du « kanban »... notamment du module “board” de Gitlab.

Daily scrum

- ▶ **Daily scrum** : Réunion courte de 15 minutes maximum.
- ▶ Objectif : un point d'information par chaque membre, pour l'équipe.
- ▶ Chaque membre répond à :
 1. Qu'ai-je fait depuis la dernière réunion ?
 2. Que vais-je faire jusqu'à la prochaine ?
 3. Quels sont mes difficultés ?
- ▶ C'est un point d'information, pas de discussion.
- ▶ On peut organiser des réunions en petit groupe pour résoudre un problème mais sans bloquer l'équipe entière.

Méthode de développement incrémental

- ▶ Nous avons déjà parlé du git-flow.
- ▶ Nous verrons deux méthodes complémentaires pour structurer le développement :
 - ▶ La programmation dirigée par des tests.
 - ▶ L'utilisation de l'intégration continue.

Raffinement du “product backlog”

- ▶ Pendant le sprint, on organise une réunion pour raffiner les éléments du backlog pour les sprints à venir (surtout pas le sprint courant).

Revue de Sprint

- ▶ Réunion de l'équipe pour faire le tour de ce qui a été intégré lors du sprint. (Par exemple, en regardant les MRs intégrées.)
- ▶ Lors de cette réunion, les membres de l'équipe :
 - ▶ prennent connaissance du travail effectué;
 - ▶ doivent évaluer la qualité de ce travail.
- ▶ On peut alors décider une mise en production / release du logiciel.

Rétrospective de Sprint

- ▶ Prise de recul de l'équipe sur elle-même :
 - ▶ Qu'est-ce qui a posé problème pour son bon fonctionnement ?
 - ▶ Qu'est-ce qui a bien fonctionné ?

Planification du prochain Sprint

- ▶ L'équipe met à jour le "product backlog".
- ▶ Le sommet de ce dernier fournit en général le "sprint backlog".

Suggestions pour de thèmes pour les sprints à venir

Sprint 1 Trouver les bugs et les corriger (un ticket par bug). Améliorer la qualité du code (corriger les FIXME).

Sprint 2 Trouver les fonctionnalités qui manquent le plus pour un Pac-Man minimal (chercher les TODOS). Les implémenter. Faire du refactoring.¹⁰

Sprint 3 Envisager des fonctionnalités additionnelles. Les implémenter et faire le refactoring.

...

Sprint 6 Préparer livrable d'évaluation : démo exécutable et vidéo.

10. Changer la façon dont est structuré et écrite le code, sans en changer le comportement.

Retour au « kanban », au Scrum Board, plus exactement

Quel workflow pour Scrum ?

- ▶ « Stories » : tickets bruts, « user stories » = product backlog
- ▶ « ToDo » : items suffisamment détaillé/raffiné, sélectionnés pour le sprint en cours et assignés = sprint backlog
- ▶ « InProgress » : travail en cours de réalisation
- ▶ « Testing » : travail réalisé mais pas complètement testé
- ▶ « Done » : travail fini, testé (et intégré à la version de développement)

L'idée est de remplir ToDo lors de la planification... et de tout vider sauf Stories et Done pendant le sprint.

Retour au « kanban », au Scrum Board, plus exactement

Quel workflow pour Scrum ?

- ▶ « Stories » : tickets bruts, « user stories » = product backlog
- ▶ « ToDo » : items suffisamment détaillé/raffiné, sélectionnés pour le sprint en cours et assignés = sprint backlog
- ▶ « InProgress » : travail en cours de réalisation
- ▶ « Testing » : travail réalisé mais pas complètement testé
- ▶ « Done » : travail fini, testé (et intégré à la version de développement)

L'idée est de remplir ToDo lors de la planification... et de tout vider sauf Stories et Done pendant le sprint.

Et vous, qu'en pensez-vous ? (Peut-on envisager d'autres colonnes, quel serait leur cycle de vie ?)

Conclusion

- ▶ Vous devez avoir lu le livre “The Scrum Primer”.
- ▶ Bon sprints à vous!
- ▶ Si vous souhaitez approfondir :
 - ▶ The Pragmatic Programmer : From Journeyman to Master
– Andrew Hunt
 - ▶ Agile Project Management with Scrum
– Ken Schwaber
 - ▶ Les pratiques de l'équipe agile
– Thomas Thiry