TDD
Agilité et Kanban
Planning Poker

Philippe Collet

Licence 3 Informatique – S6

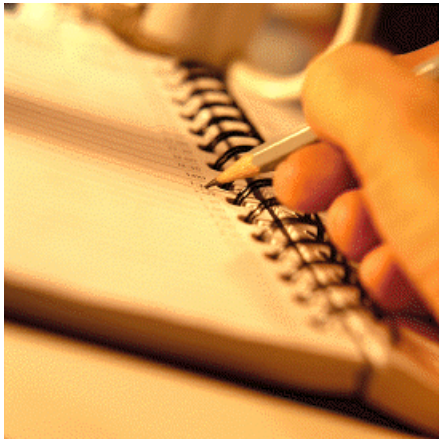
2013-2014

http://deptinfo.unice.fr/twiki/bin/view/Linfo/ProjetDeLicence201314

Université Nice Sophia Antipolis

Plan

- TDD
- XP
- Scrum
- Kanban
- Planning Poker



Tests : Aspects méthodologiques (rappel)

- **Coder/tester, coder/tester…**
- **lancer les tests aussi souvent que possible**
  - aussi souvent que le compilateur !
- **Commencer par écrire les tests sur les parties les plus critiques**
  - Ecrire les tests qui ont le meilleur retour sur investissement !
  - Approche *Extreme Programming*
- **Quand on ajoute des fonctionnalités, on écrit d'abord les tests**
  - *Test-Driven Development*…
- **Si on se retrouve à déboguer à coup de System.out.println(), il vaut mieux écrire un test à la place**
- **Quand on trouve un bug, écrire un test qui le caractérise**

Test Driven Development

- Méthode traditionnelle (incrémentale)
  - Ajouter un peu de code
  - Ajouter un test sur ce bout de code
- Méthode TDD
  - Ajouter un code de test
  - Ajouter du code qui respecte le test
- Mise en pratique (code de couleur Junit)
  - R (Red): écrire un code de test et les faire échouer
  - G (Green) : écrire le code métier qui valide le test
  - R (Refactor) : remanier le code afin d'en améliorer la qualité

Cycle TDD : 5 étapes

1. **Ecriture d'un premier test**

2. **Exécuter le test et vérifier qu'il échoue**
   - car le code qu'il teste n'a pas encore été implémenté

3. **Ecriture de l'implémentation pour faire passer le test**
   - il existe différentes manières de corriger ce code

4. **Exécution des tests afin de contrôler que les tests passent**
   - l'implémentation va respecter les règles fonctionnelles des tests unitaires

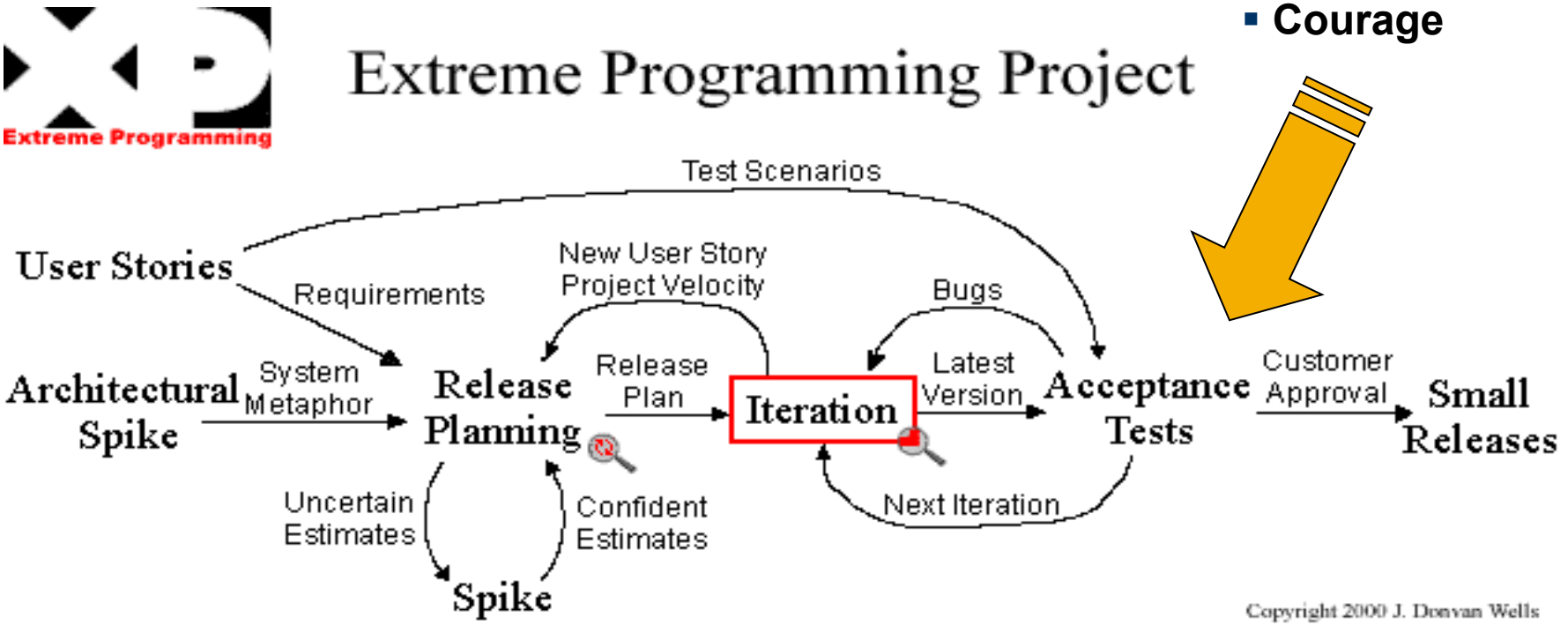
5. **Remaniement (Refactor) du code afin d'en améliorer la qualité**
   - mais en conservant les mêmes fonctionnalités
   - Les tests sont **repassés** après refactoring !!!

eXtreme Programming (XP…) : mise en oeuvre…

- Ensemble de « Bests Practices » de développement
  (travail en équipes, transfert de compétences…)
- plutôt pour des projets de moins de 10 personnes

4 Valeurs

- **Communication**
- **Simplicité**
- **Feedback**
- **Courage**



- **Collaboration étroite entre équipe(s) de programmation et experts métier**
  - **Communication orale, pas écrite**
  - **Livraison fréquente de fonctionnalités déployables et utilisables (= qui apportent une valeur ajoutée)**
  - **Equipe auto-organisée et soudée**

Manifeste Agile : 12 principes

1. **Our highest priority is to satisfy the costumer through early and continuous delivery of valuable software.**

2. **Welcome changing requirements, even late in development. Agile process harness change for the customer´s competitive advantage.**

3. **Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale.**

4. **Business people and developers must work together daily throughout the project.**

5. **Build projects around motivated individuals. Give them the environment and support they need, and trust them to get the job done.**

6. **The most efficient and effective method of conveying information to and within a development team is face to face conversation.**

Manifeste Agile : 12 principes

7. **Working software is the primary measure of progress**

8. **Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely**

9. **Continuous attention to technical excellence and good design enhances agility**

10. **Simplicity – the art of maximizing the amount of work not done – is essential**

11. **The best architectures, requirements, and designs emerge from self-organizing teams**

12. **At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly**

Scrum : principes

- **Isolement de l'équipe de développement**
  - **l'équipe est isolée de toute influence extérieure qui pourrait lui nuire. Seules l'information et les tâches reliées au projet lui parviennent : pas d'évolution des besoins dans chaque sprint.**

- **Développement progressif**
  - **afin de forcer l'équipe à progresser, elle doit livrer une solution tous les 30 jours. Durant cette période de développement l'équipe se doit de livrer une série de fonctionnalités qui devront être opérationnelles à la fin des 30 jours.**

- **Pouvoir à l'équipe**
  - **l'équipe reçoit les pleins pouvoirs pour réaliser les fonctionnalités. C'est elle qui détient la responsabilité de décider comment atteindre ses objectifs. Sa seule contrainte est de livrer une solution qui convienne au client dans un délai de 30 jours.**

- **Contrôle du travail**
  - **le travail est contrôlé quotidiennement pour savoir si tout va bien pour les membres de l'équipe et à la fin des 30 jours de développement pour savoir si la solution répond au besoin du client.**

Scrum : rôles et pratiques

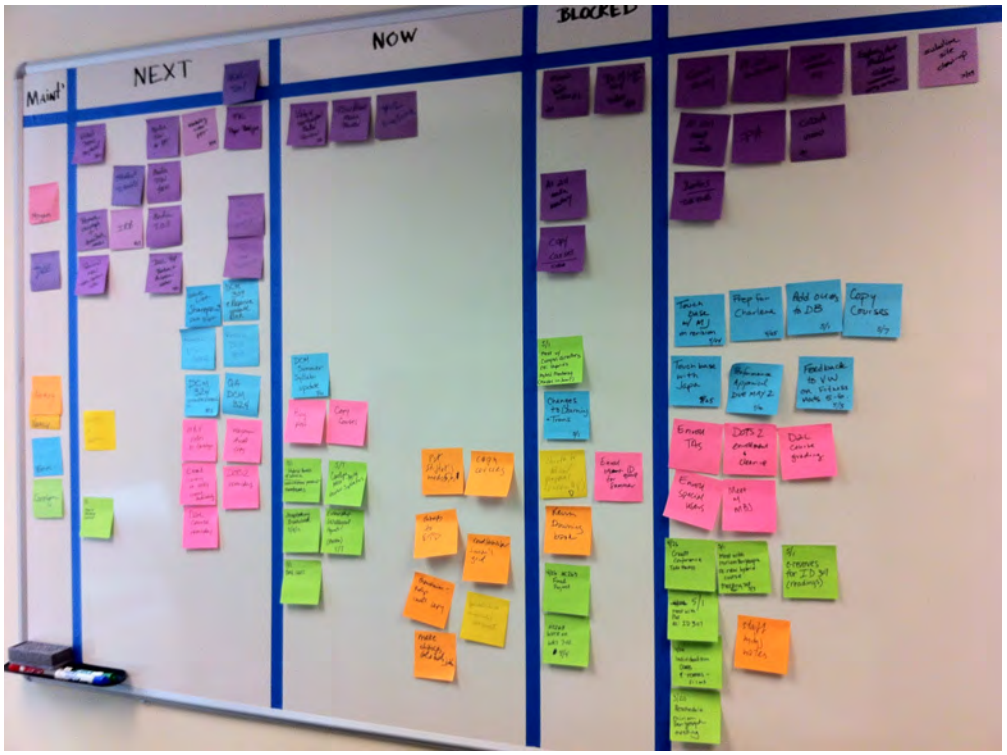
- **Scrum Master**
  - expert de l'application de Scrum
- **Product owner**
  - responsable officiel du projet
- **Scrum Team**
  - équipe projet.
- **Customer**
  - participe aux réunions liées aux fonctionnalités
- **Management**
  - prend les décisions
- **Product Backlog**
  - état courant des tâches à accomplir
- **Effort Estimation**
  - permanente, sur les entrées du backlog
- **Sprint**
  - itération de 30 jours
- **Sprint Planning Meeting**
  - réunion de décision des objectifs du prochain sprint et de la manière de les implémenter
- **Sprint Backlog**
  - Product Backlog limité au sprint en cours
- **Daily Scrum meeting**
  - ce qui a été fait, ce qui reste à faire, les problèmes
- **Sprint Review Meeting**
  - présentation des résultats du sprint

Scrum dans vos projets

- 1 sprint = 3 semaines / 1 mois
  - Trop long
- Le client doit toujours être à disposition
  - Ce n'est pas le cas
- L'équipe travaille tout le temps ensemble et à plein temps
  - Ce n'est pas le cas non plus
- So what ?

Cherchons juste à :

- Maximiser le minimalisme…
- Kanban
  - Mot japonais signifiant étiquette (ou petite fiche)
  - Pratique basée sur l'utilisation d'étiquette (post-it) pour matérialiser les informations sur le processus



Kanban : historique

- **Méthode inventée à la fin des années 1950 dans les usines Toyota**
  - Mise en place entre deux postes de travail
  - Une simple fiche cartonnée fixée sur les bacs de pièces dans une ligne d'assemblage ou une zone de stockage

- **Intérêts chez Toyota**
  - Limite la production du poste amont aux besoins exacts du poste aval
  - Le nombre de kanban en circulation doit être limité pour éviter la constitution d'en-cours trop importants

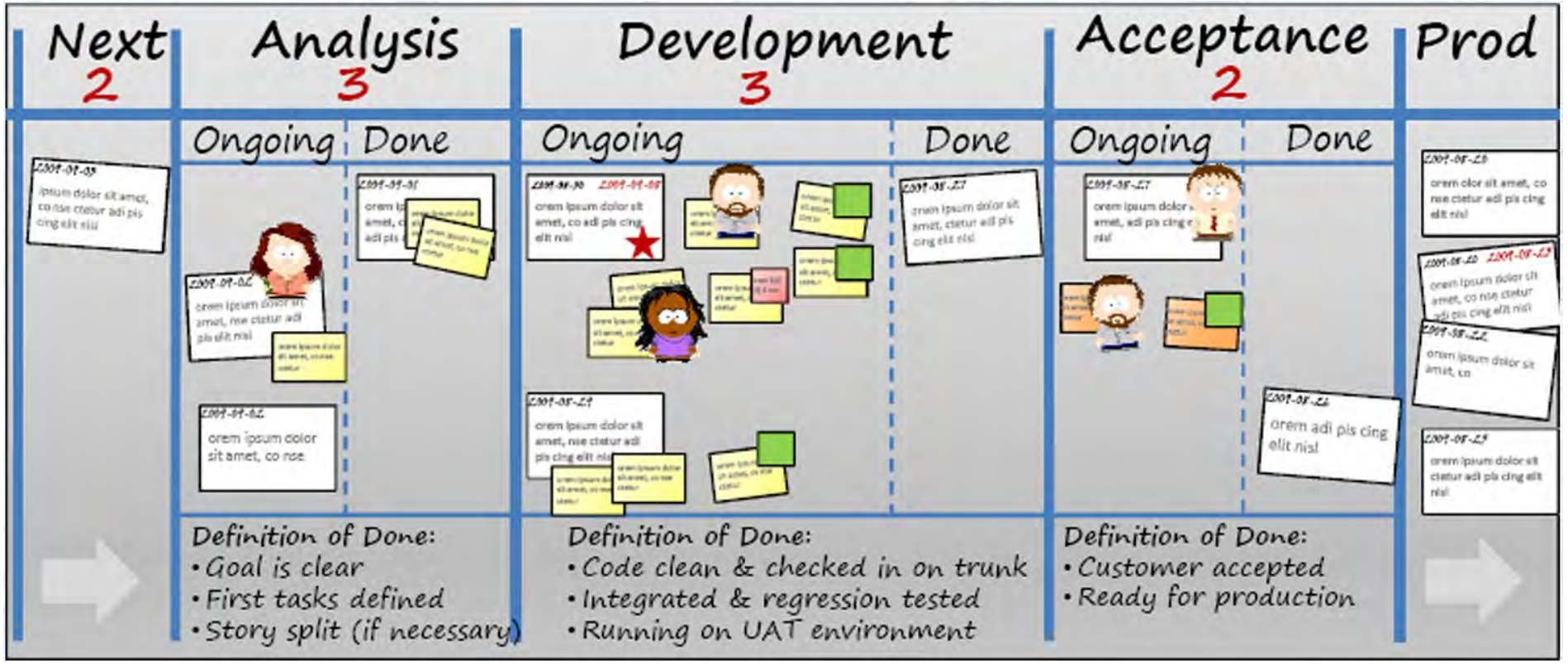
- **Cette méthode est au départ adaptée aux entreprises ayant une production répétitive et relativement régulière**

Kanban : premiers principes

- Le problème à résoudre est un workflow
  - La solution consiste à le visualiser
- Diviser le travail (comme dans Scrum…)
  - Décrire chaque élément sur une fiche et la mettre au mur (tableau, board)
  - Tracer des colonnes, donnez leur le nom des étapes du workflow
  - Placer les éléments du travail
- En tant que développeur
  - Choisir ce qui est à faire en fonction de ses compétences ou missions
  - Faire ce qui est « en cours »
  - Mettre à jour le tableau Kanban

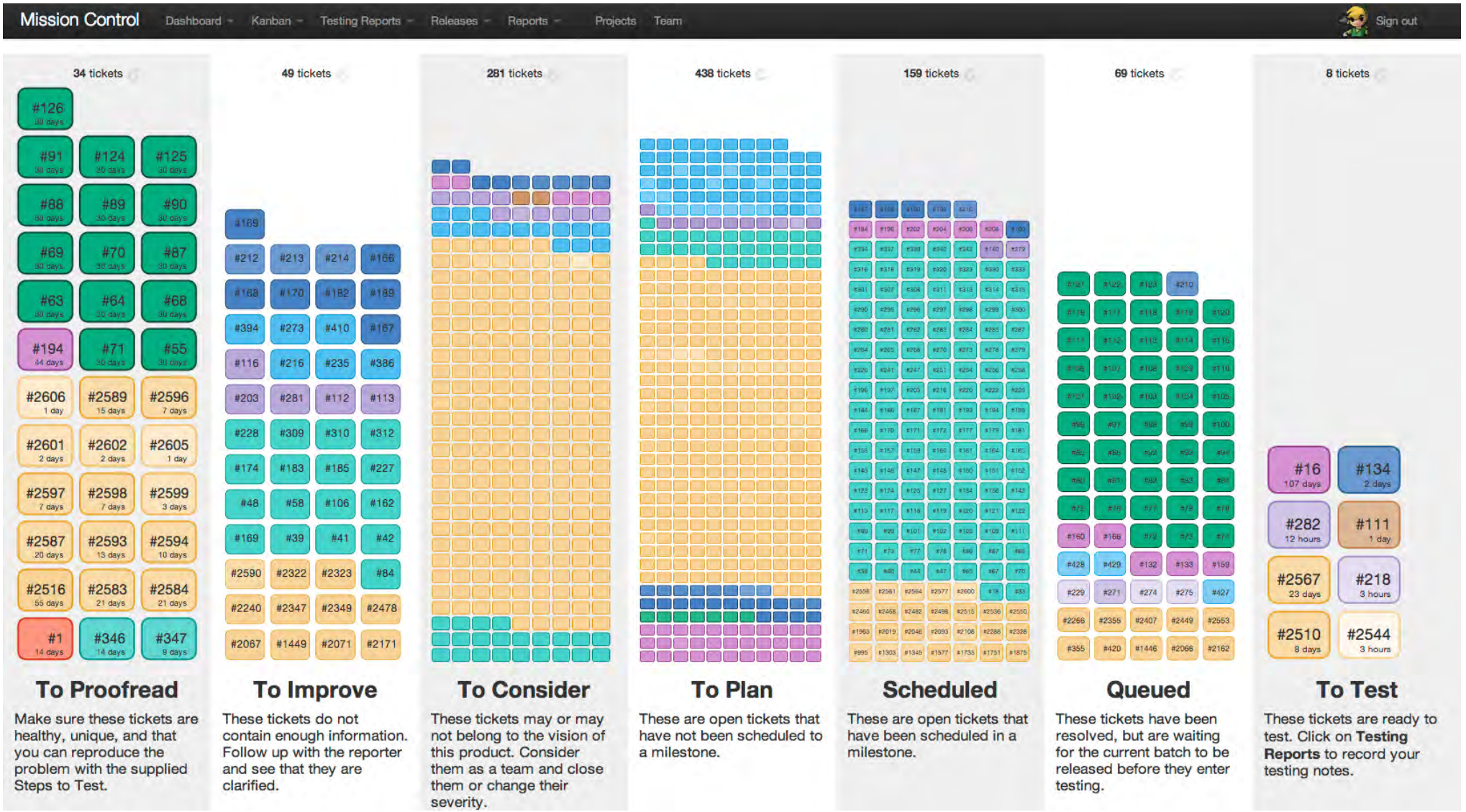
Kanban: exemples d'étapes

- Au minimum
  - ToDo → In Progress → Done
- Plus Fin
  - ToDo → Chosen → Development →
  - Tests → Delivery → Done



http://www.modalisa-technology.com/

Kanban sur très gros projets : github.com



Kanban : autres principes

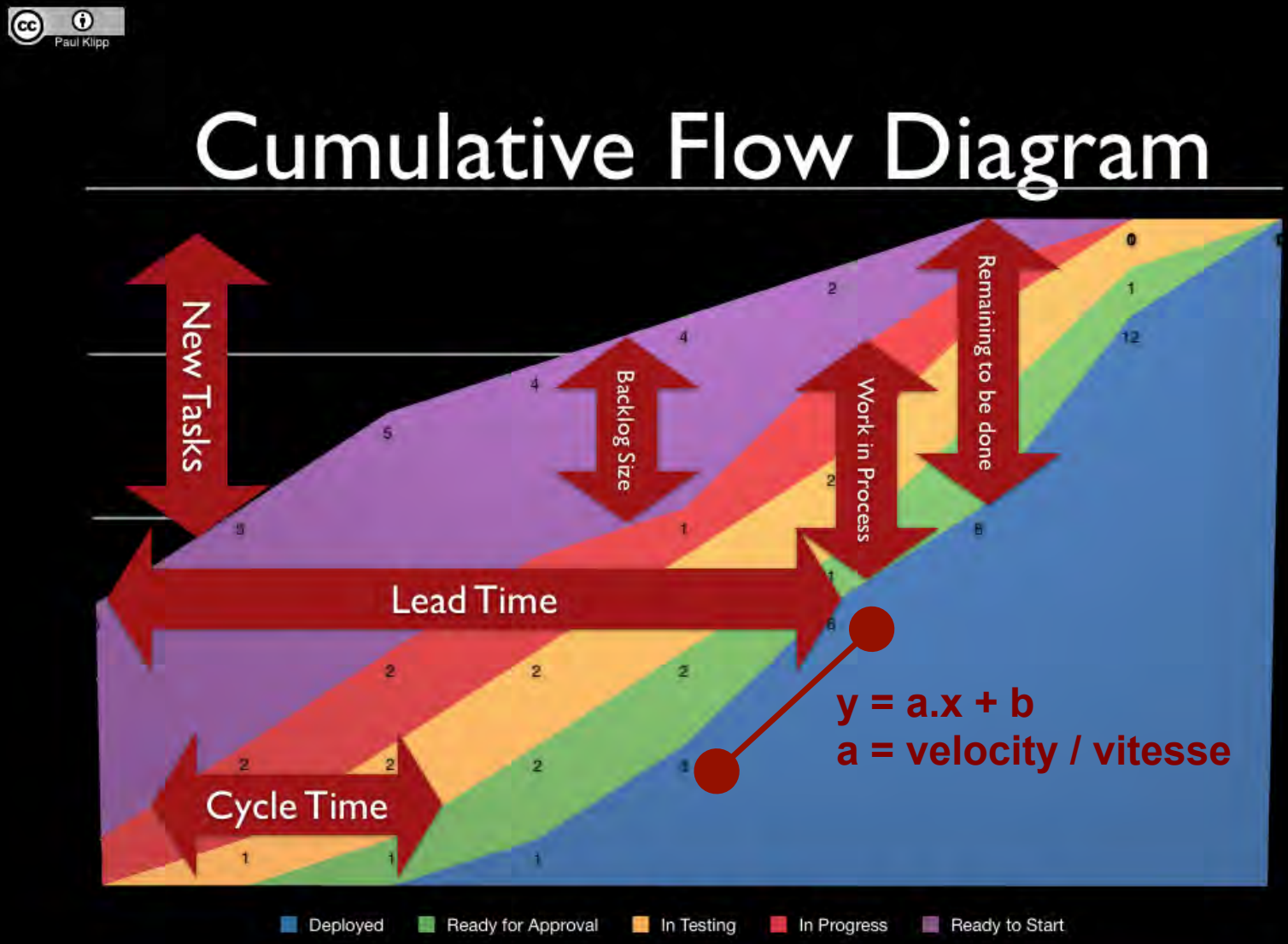
- Et si j'ai 12000 fiches dans « In Progress », ou « TODO », je fais quoi ?

- Limiter le TAF (Travail A Faire / WIP : Work in Progress)
  - Fixer des bornes au nombre d'éléments dans chaque étape

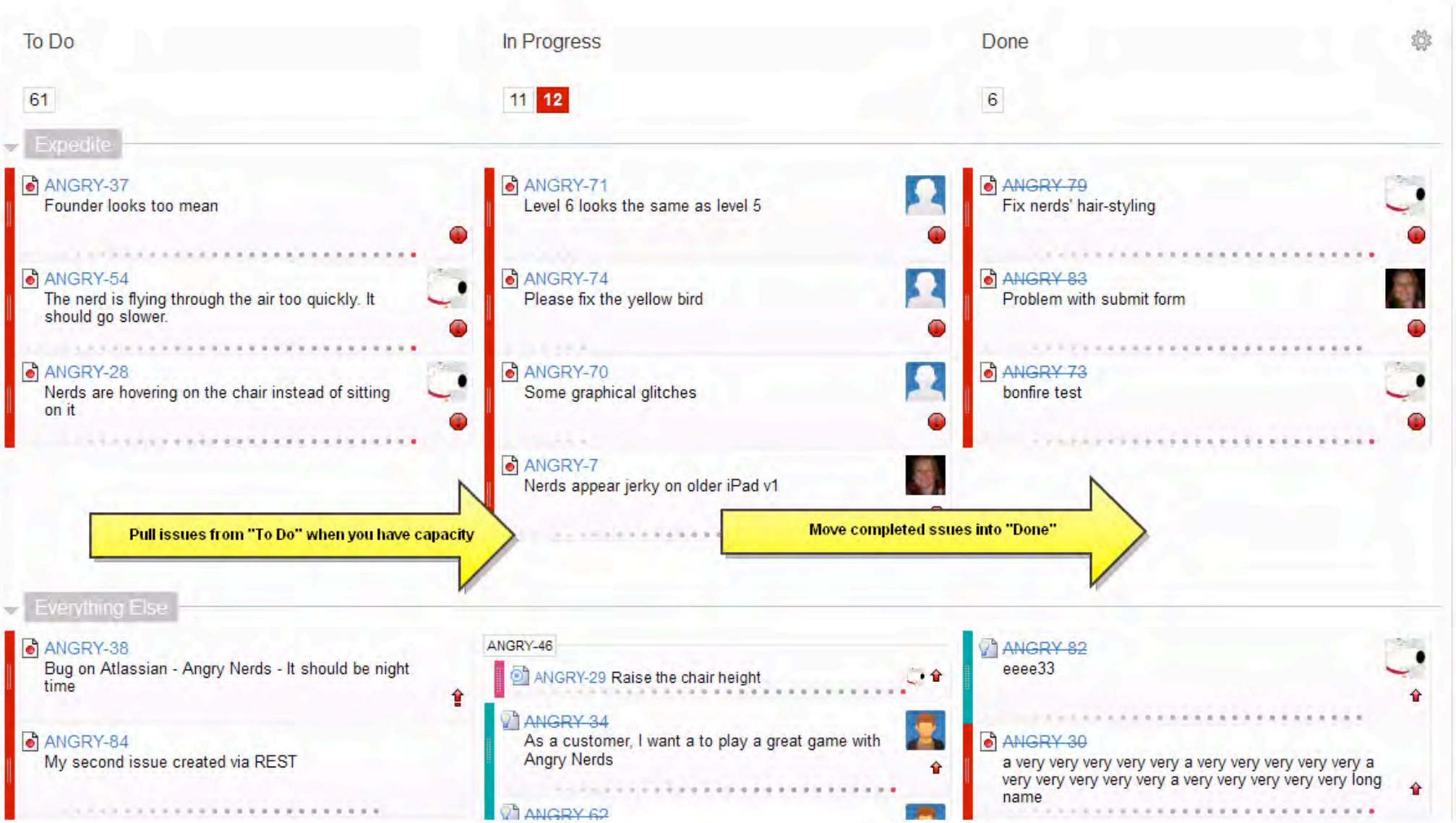
- Mesurer le temps de cycle (lead time)
  - C'est le temps moyen pour traiter complètement un élément, c'est à dire le faire passer par toutes les étapes du workflow
  - Optimiser le processus en réduisant le temps de cycle et en le rendant prévisible

No silver bullet : pas de borne fixe, ca dépend…

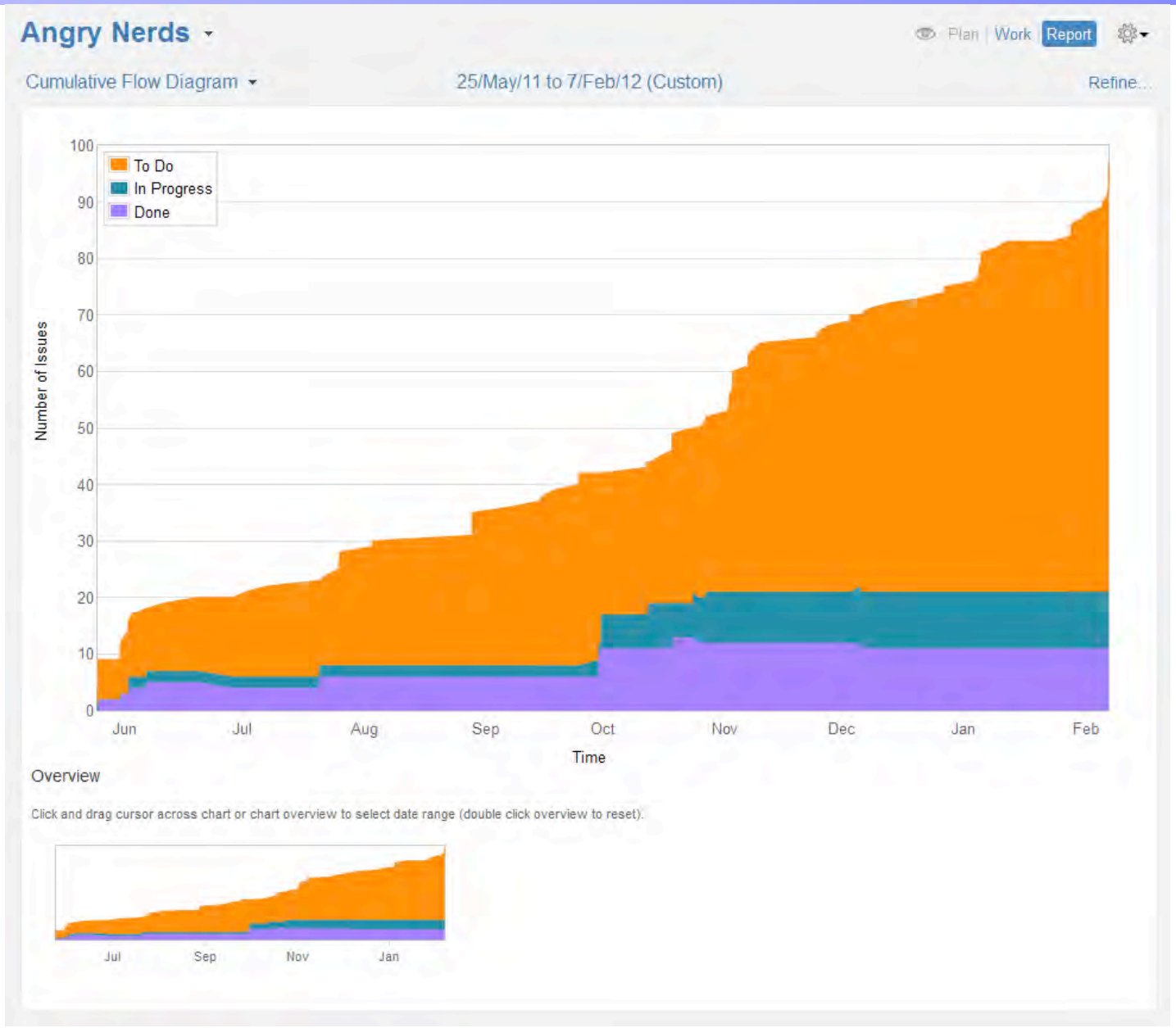
Mesurer...


Paul Klipp
Cumulative Flow Diagram
New Tasks
Backlog Size
Work in Process
Remaining to be done
Lead Time
Cycle Time
y = a.x + b
a = velocity / vitesse
Deployed
Ready for Approval
In Testing
In Progress
Ready to Start

Kanban dans JIRA



WIP (Work-In-Progress) cumulatif dans JIRA



Estimer la durée des tâches

- Une activité extrêmement complexe
  - Seule l'expérience permet de réaliser des estimations avec une marge d'erreur acceptable

- Méthodes « classique » d'estimation
  - Plein de formules mathématiques
  - Utilisation de plusieurs « experts »
  - Plus ou moins coûteuses et très peu précises

Planning Poker

- Utilisé dans les méthodes *agiles*
  - Livraison incrémentale
  - Correction de « trajectoire » fréquente
- Auteur : J. Grenning (2002)
  - Popularisé par M. Cohn (*Agile Estimating and Planning*)
- Avantage : expression libre de tous sur l'estimation

Planning Poker : déroulement

- Tous les développeurs sont impliqués
  - Ils estiment l'ensemble de la tâche, pas uniquement leur partie
  - Un des développeurs est le modérateur
- Le product owner peut être la, mais ne participe pas
  - Il obtiendra des estimations sur l'ensemble des « stories »
- Chaque développeur reçoit un paquet de cartes

Les cartes

Plus c'est gros, plus l'estimation est grossière

Planning Poker : déroulement

- Pour chaque « user story » ou activité à évaluer
  - Le modérateur lit la description
  - Le *product owner* répond aux éventuelles questions
  - Chaque développeur choisit ensuite une carte pour cette estimation, la carte reste cachée
- Ensuite, les cartes sont retournées
  - Les estimations vont différer, la plus grande et la plus petite explique leur point de vue
  - On discute
  - On repart

Planning Poker : déroulement

- Le *product owner* utilise les résultats pour fixer les priorités

- A la fin de l'itération, on compare l'estimation aux nombre de jours réel :
  - On obtient la vélocité de l'équipe

- Au début, il faut bien choisir la première *story* qui va servir à l'estimation
  - Trop grande : on va se retrouver avec des fractions
  - Trop petite : l'estimation sera trop facile

Pourquoi ça marche ?

- Plusieurs « experts » donnent leur opinion sans s'influencer
  - On ne parle pas
  - On n'influence pas par le langage du corps
- Cela améliore la qualité de l'estimation
  - On doit justifier ses estimations
  - Moyenner les estimations des personnes donne de meilleurs résultats
- L'utilisation des nombres de Fibonacci ?
  - On ne sait pas trop l'expliquer…

Questions