CVDA : Cycle de Vie de Développement d'Applications

Catarina FERREIRA DA SILVA
Catarina.ferreira@univ-lyon1.fr

Objectifs de ce module

- Distinguer les différents types d'Ateliers de Génie Logiciel (AGL) et savoir manipuler quelques-uns (PowerDesigner, Netbeans,…)
- Apprendre à choisir un AGL pour répondre à des besoins spécifiques
- Savoir effectuer de la veille technologique, c'est-à-dire chercher des outils, les comparer, et les choisir en fonction des besoins
- Comprendre la notion de cycle de vie d'un logiciel
- Connaître les différents modèles de cycle de développement de logiciels : traditionnel en cascade, cycle en V, cycle en spirale, *quick and dirty*, semi-itératif, *Rapid Application Development* (RAD), *eXtreme Programming* (XP) et Agile
- Connaître et utiliser des outils de gestion de versions
- Connaître et utiliser des outils de test de logiciels
- Connaître et utiliser des outils de debugging

Modalités d'évaluation

- Un exposé de veille technologique en groupe : 30%
- Un DS contenant environ la 1er moitié de la matière : 30%
- Un DS contenant environ la 2ème moitié de la matière : 30%
- La participation dans les séances de cours et des TP à retourner à la fin des séances : 10%

Atelier de Génie Logiciel

- Un AGL est un ensemble d'outils intégrés couvrant une partie significative du cycle de vie d'un logiciel, et qui permettent de mettre en œuvre les principes du Génie Logiciel
- Un ensemble d'outils de Génie Logiciel regroupés sous un « chapeau » commun, ce qui assure une certaine homogénéité d'utilisation entre les différents outils
- Un AGL est un logiciel qui aide à fabriquer d’autres logiciels

AGL orientés conception

- Spécification des besoins (support de communication avec le client et/ou le chef de projet)
- Aide à la conception (réalisation de diagrammes UML par exemple). C'est un gain de temps par rapport au papier crayon, et cela permet également la réutilisation d'éléments dans d'autres phases du cycle de vie (génération de code à partir du modèle UML par exemple)
- Aide à la documentation de conception (production de rapports de conception, comme avec PowerDesigner). Dans ce cas, les AGL sont de véritables outils de communication, avant tout chose
- Outils de gestion de projet (liste des tâches, temps de travail, diagrammes de **Gantt**, etc.)
- Générateurs de code (production du script de création d'une BD à partir de PowerDesigner)
- Outils de prototypage rapide (réalisation de maquettes et de code quick-and-dirty)

AGL orientés développement

- Aide à la documentation de code (génération de Javadoc)
- Editeur de code (indentation automatique, coloration syntaxique, auto-complétion)
- Outils de **profiling de code** (place mémoire, temps d'exécution, code non utilisé, etc.)
- Debugger, optimiseur de code
- Outil de **gestion de versions** (cet aspect fera l'objet d'un mini-cours dédié)
- Outil de configuration et de suivi des tests, génération de jeux de tests
- Gestionnaire de bugs
- Suivi et maintenance (demande d'ajouts de nouvelles fonctionnalités, **tests de non régression**)

Les AGL en résumé

- Le terme AGL recouvre une gamme vaste et imprécise d'outils visant à aider les concepteurs et les développeurs à réaliser des logiciels de qualité, dans de bonnes conditions
  - Un AGL doit permettre d'améliorer la qualité du produit fini, la productivité de ses réalisateurs, et le confort de travail
- Certains AGL permettent de donner plusieurs points de vue sur la réalisation du logiciel : cycle de vie, état d'avancement, connexion avec d'autres composants, etc.
  - D'autres se contentent de donner un point de vue unique : la modélisation par exemple
- Le choix de l'AGL à utiliser doit être fait en fonction des objectifs du projet, et non l'inverse
  - Il est donc très important de savoir identifier les caractéristiques pertinentes d'un projet qui serviront de critères de choix de l'AGL.
- Les objectifs de la partie suivante (travail pratique de veille technologique) sont d'une part d'apprendre à mieux connaître les AGL existants et à en repérer les fonctionnalités, et d'autre part, d'apprendre à choisir un AGL pour un objectif donné

Les questions à se poser lorsque l'on doit choisir un AGL

- Temps de prise en main admissible ?
- A-t-on déjà des connaissances sur l'outil ?
- Cherche-t-on un outil de conception ou de développement ?
- Quelles fonctionnalités sont disponibles ?
- Un AGL est-il imposé par la structure dans laquelle se déroule le projet ?
- Doit-on choisir une solution partagée par plusieurs utilisateurs ?
- La solution retenue est-elle difficile à déployer à l'échelle de la structure ?

Les critères de choix

- Orienté conception ou orienté développement ?
- L'outil est-il simple à prendre en main ou bien riche en fonctionnalités ?
- Léger / efficace / rapide ?
- Quels langages de programmations sont supportés ?
- Sur quelles plateformes tourne la solution ?
- Gratuit ou payant ? Libre ? Version d'évaluation ?
- Supporte le travail collaboratif ?
- Réputation, qualité des mises à jour ?
- Documentation ? Communauté d'utilisateurs ?

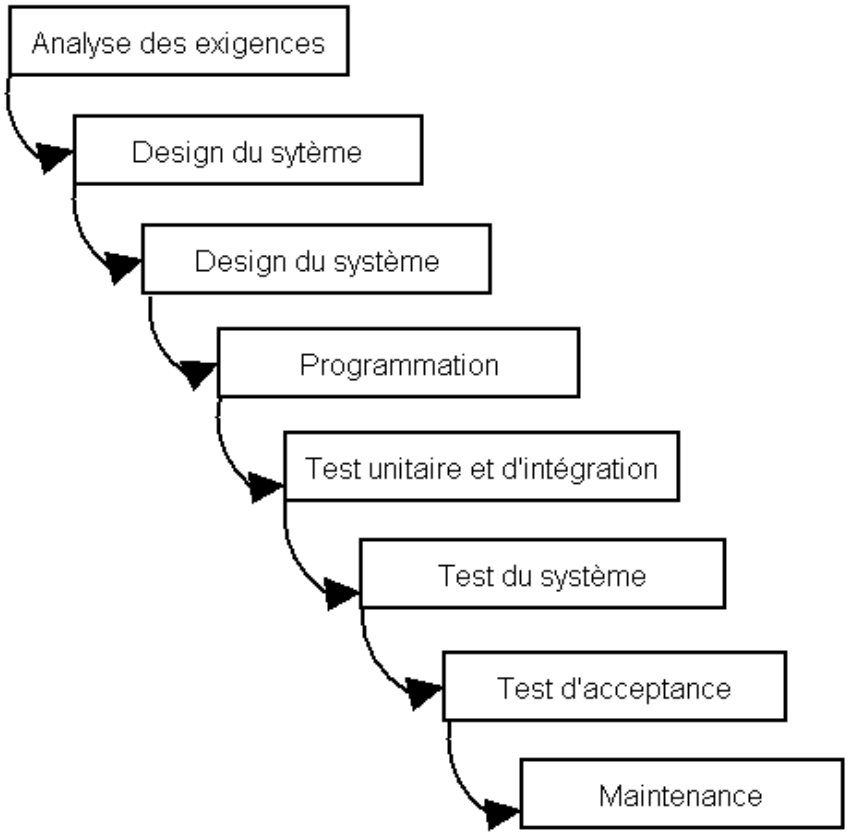
Etapes du Cycle de Vie de Logiciel

- Définition des objectifs
- Analyse des besoins et faisabilité
- Spécifications ou conception générale
- Conception détaillée
- Développement
- Tests (unitaires, fonctionnels)
- Intégration
- Qualification
- Documentation
- Mise en production
- Maintenance

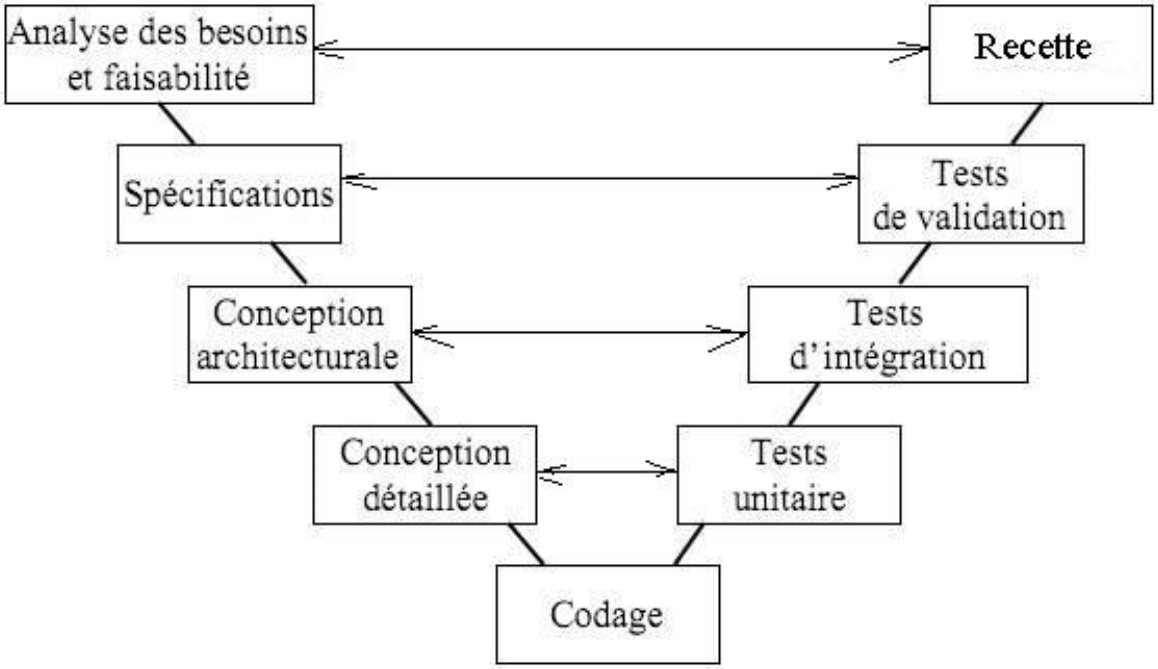
Les modèles de conception et de développement

- Le modèle en cascade
- Le cycle en V
- Le cycle en spirale
- Le développement itératif et incrémental
- L'approche *Quick-and-dirty*
- Le cycle semi-itératif
- La méthode RAD
- La méthode de l'eXtreme Programming
- Les méthodes agiles

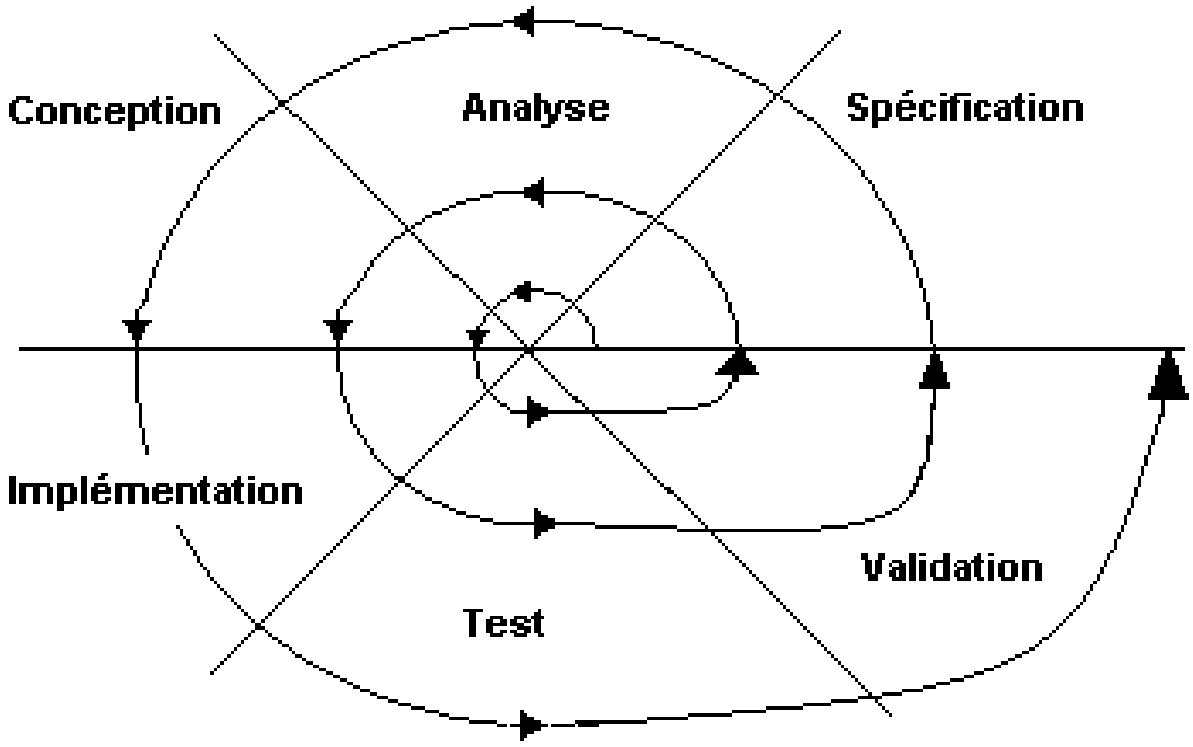
Le modèle en cascade



Le cycle en V


Analyse des besoins et faisabilité
Recette
Spécifications
Tests de validation
Conception architecturale
Tests d'intégration
Conception détaillée
Tests unitaire
Codage

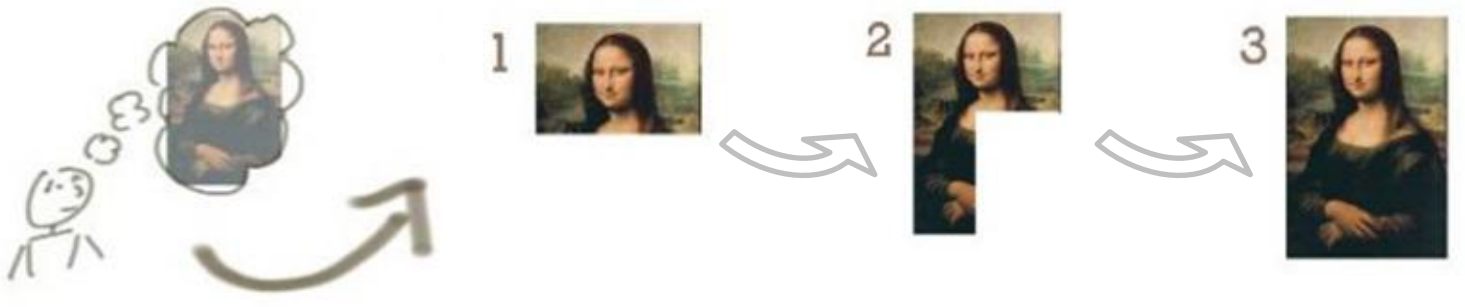
Le cycle en spirale version simple



Le développement itératif et incrémental

- Spécification pilotée par les cas d'utilisation (use-case driven)
  - Qu'est-ce que le logiciel doit faire pour chaque utilisateur ?
  - Les tests doivent assurer les cas d'utilisations
- Développement orientée architecture
  - Spécification détaillé des cas d'utilisation en termes de sous-systèmes, classes, composants
- Développement itératif et incrémental
  - Projet divisé en mini-projets, dont chacun est une itération qui se traduit par une augmentation (ou incrément)

Modèle INCREMENTALE



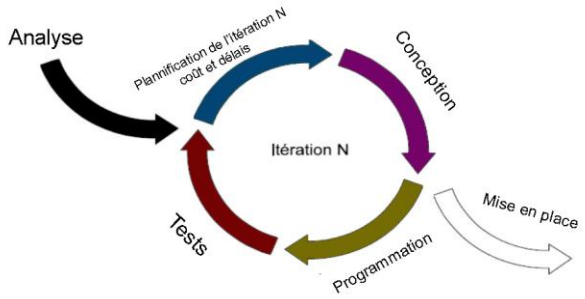
Idée initiale | Livrable 1 | Livrable 2 | Résultat final

http://romy.tetue.net/mona-lisa-agile

Modèle INCREMENTALE

Aucune cohérence d’ensemble

Modèle ITERATIF



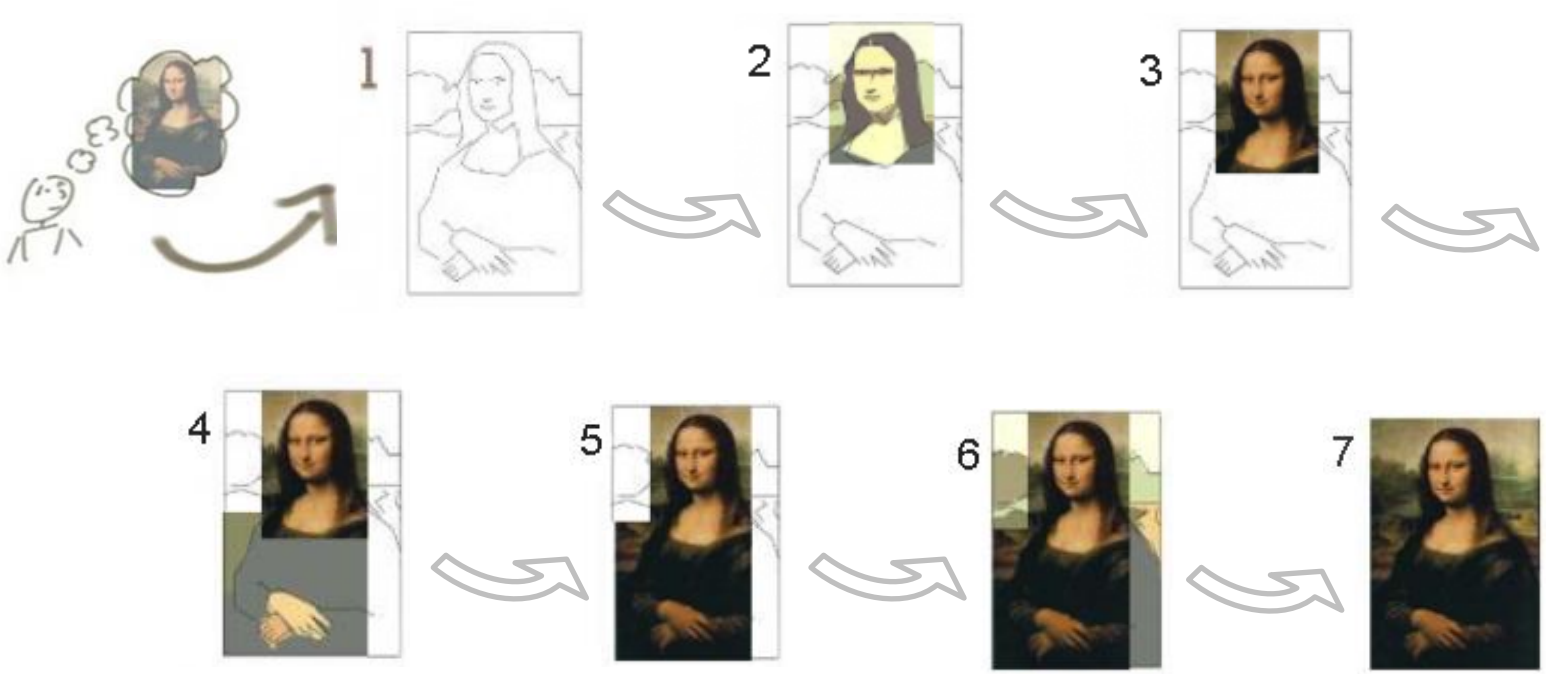
Idée floue

Amélioration à chaque itération

Résultat final

http://romy.tetue.net/mona-lisa-agile

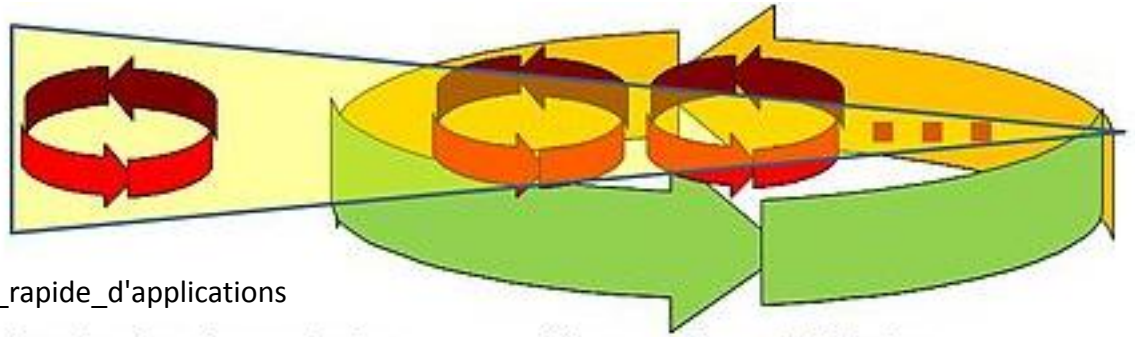
Modèle ITERATIF et INCREMENTALE



http://romy.tetue.net/mona-lisa-agile

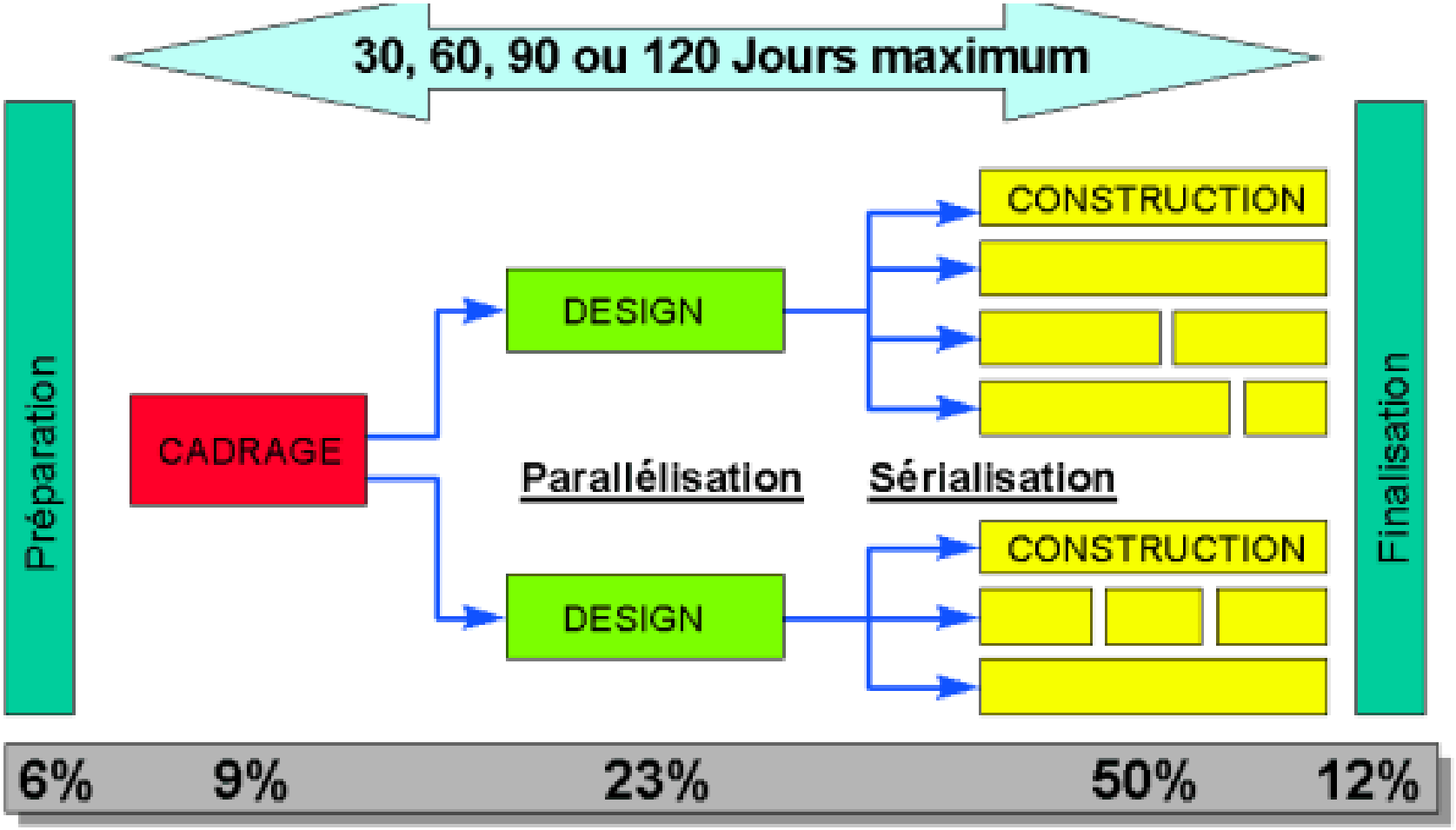
La méthode RAD

- Les deux premières phases sont classiques
  - Expression des besoins
  - Conception de la solution
- Mais la troisième phase change
  - Itérations courtes pour le développement
    - permet de faire avancer le code rapidement, et par morceaux
    - en assurant la possibilité de faire de nombreux tests



https://fr.wikipedia.org/wiki/Développement_rapide_d'applications

Un schéma RAD



https://fr.wikipedia.org/wiki/Développement_rapide_d'applications

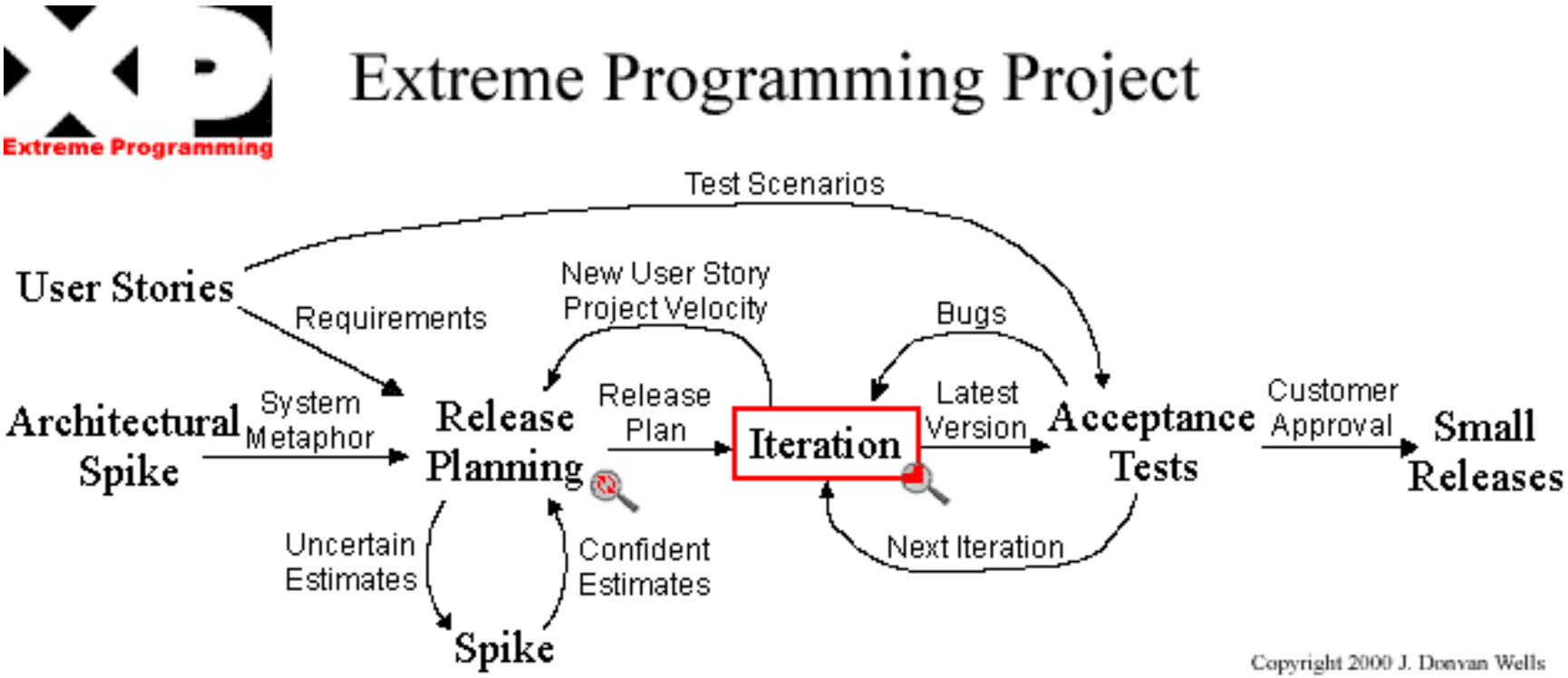
RAD : cycle itératif, incrémental et adaptatif

- Genèse des méthodes agiles (flexibles)
- Implication le plus possible de l'utilisateur final dans le développement
  - Lui faisant valider les avancées de l'application le plus souvent possible
- Cycles de développement très courts
  - Facilitent également la mise en place de tests simples mais efficaces

La méthode XP (eXtreme Programming)

- Les activités d'analyse, de programmation, de test et de validation sont effectuées continuellement et parallèlement
  - Intégration continue
  - Forte coopération de l'utilisateur, parfois considéré comme co-auteur du logiciel
- Selon un cycle qui comporte de très nombreuses et fréquentes itérations avec à chaque fois un jeu restreint de fonctionnalités

Un schéma XP



http://www.extremeprogramming.org

Principe de méthodes **agiles**

- En rupture avec les procédés d'ingénierie classiques
- Regroupe les méthodes RAD, XP, Scrum…
- Une structure commune : **itérative**, **incrémentale** et **adaptative**
- Mettent l'accent sur
  - Les changements constants du cahier des charges et du code source des logiciels,
  - Une collaboration étroite et une forte implication de l'utilisateur final
  - Un cycle de développement en **spirale** avec de **nombreuses et courtes itérations**

Méthodes Agiles : définitions de lots de fonctionnalités

- Fonctionnent selon un schéma itératif pour arriver à la solution finale
- L'analyse des besoins débouche sur une liste de spécifications à classer **par lot de réalisation**
  - A l'intérieur de ces lots **par priorité d'implémentations** (haute, moyenne, faible)
  - Certaines fonctionnalités sont dépendantes d'autres
    - Ceci conditionne de la définition des **priorités**
- La définition du nombre d'itérations permet de garder la maîtrise du déroulement du projet tant en terme de charge que de planning

Méthodes Agiles : itérations et lots

- On lance un certain nombre d'**itérations** basées sur les lots
- Une itération est constituée par l'implémentation d'une **liste de spécifications plutôt courte**
- L'application est alors développée, testée et livrée aux utilisateurs **pour qu'ils puissent faire leur retour**
- **Les demandes de modifications sont prises en compte** et implémentées pour donner lieu à une autre livraison

Avantages des Méthodes Agiles

- Bien adaptées aux clients qui veulent rapidement disposer de l'application
  - La livraison même partielle est rassurante car montre l'avancement du projet
- Permettent de prendre en compte une expression de besoin floue ou incomplète
  - C'est devant l'application que l'utilisateur prendra conscience des oublis ou des erreurs dans le cahier des charges qu'il a pu commettre
- L'interaction permanente entre les utilisateurs et l'équipe de développement
  - Evite l'effet tunnel, où les utilisateurs doivent attendre de longues semaines la livraison de l'application
  - Favorise une meilleure compréhension des besoins des utilisateurs par les développeurs

Manifeste pour le développement Agile / Flexible de logiciels

- Nous découvrons comment mieux développer des logiciels par la pratique et en aidant les autres à le faire
- Valorisation
  - **Les individus et leurs interactions** plus que les processus et les outils
  - **Des logiciels opérationnels** plus qu'une documentation exhaustive
  - **La collaboration avec les clients** plus que la négociation contractuelle
  - **L'adaptation au changement** plus que le suivi d'un plan
- Nous reconnaissons la valeur des seconds éléments, mais privilégions les premiers

http://agilemanifesto.org/iso/fr/manifesto.html

12 Principes sous-jacents à ce manifeste (1/3)

1. Notre plus haute priorité est de satisfaire le client en livrant rapidement et régulièrement des fonctionnalités à grande valeur ajoutée

2. Accueillez positivement les changements de besoins, même tard dans le projet. Les processus Agiles exploitent le changement pour donner un avantage compétitif au client

3. Livrez fréquemment un logiciel opérationnel avec des cycles de quelques semaines à quelques mois et une préférence pour les plus courts

http://agilemanifesto.org/iso/fr/principles.html

12 Principes sous-jacents à ce manifeste (2/3)

4. Les utilisateurs ou leurs représentants et les développeurs doivent travailler ensemble quotidiennement tout au long du projet

5. Réalisez les projets avec des personnes motivées. Fournissez-leur l'environnement et le soutien dont ils ont besoin et faites-leur confiance pour atteindre les objectifs fixés

6. La méthode la plus simple et la plus efficace pour transmettre de l'information à l'équipe de développement et à l'intérieur de celle-ci est le dialogue en face à face

7. Un logiciel opérationnel est la principale mesure d'avancement

12 Principes sous-jacents à ce manifeste (3/3)

8. Les processus Agiles encouragent un rythme de développement soutenable. Ensemble, les commanditaires, les développeurs et les utilisateurs devraient être capables de maintenir indéfiniment un rythme constant

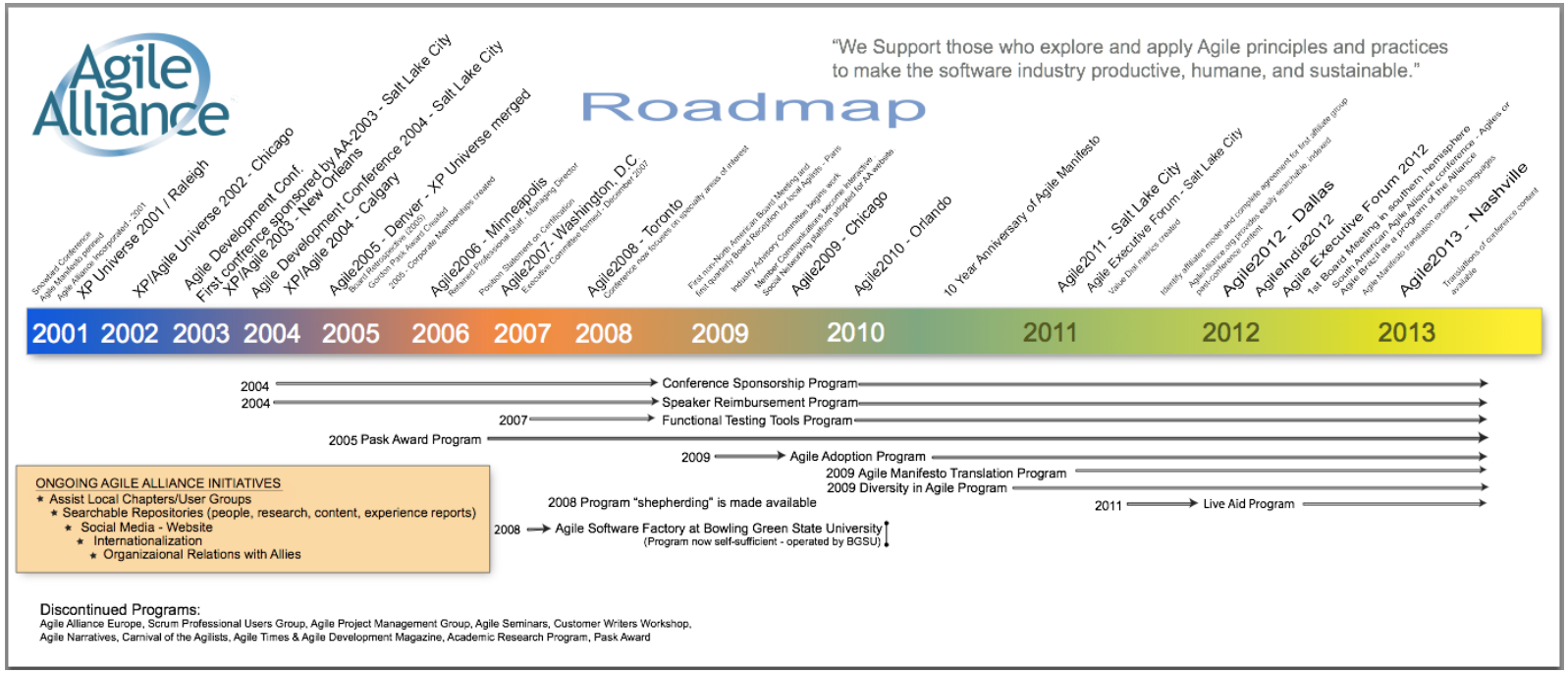
9. Une attention continue à l'excellence technique et à une bonne conception renforce l'Agilité

10. La simplicité – c'est-à-dire l'art de minimiser la quantité de travail inutile – est essentielle

11. Les meilleures architectures, spécifications et conceptions émergent d'équipes auto-organisées

12. À intervalles réguliers, l'équipe réfléchit aux moyens de devenir plus efficace, puis règle et modifie son comportement en conséquence

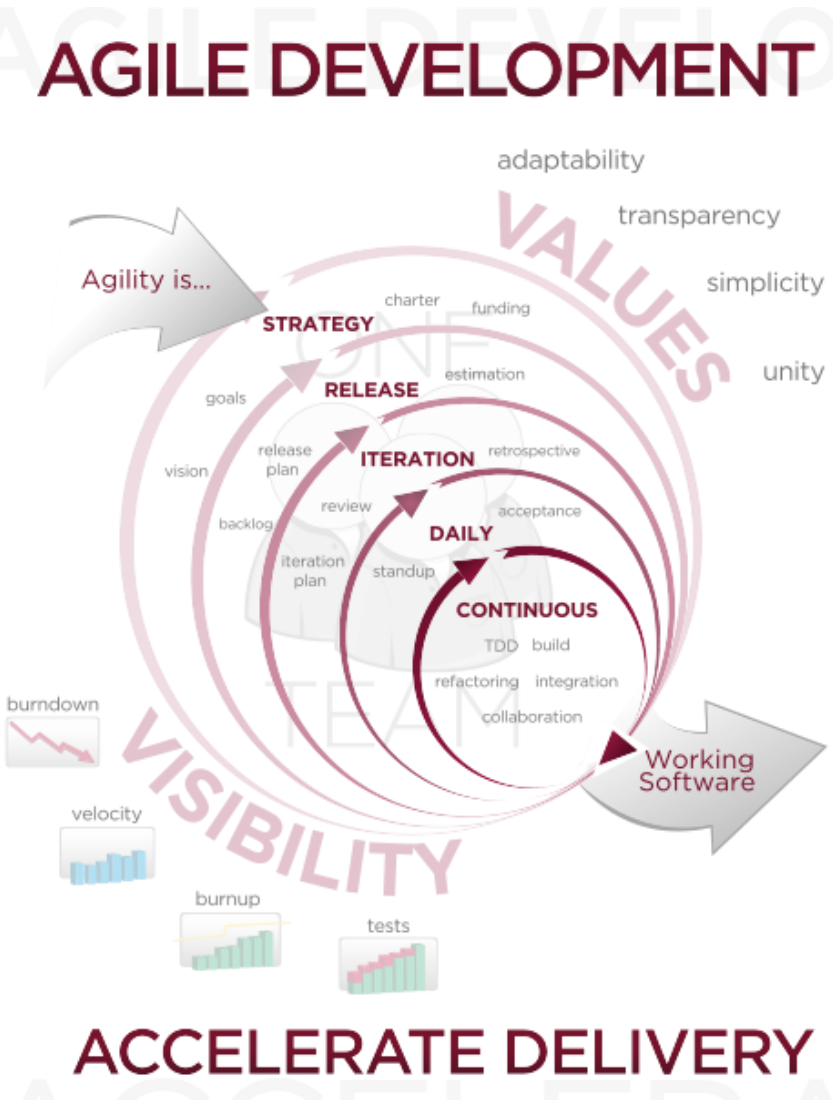
Feuille de route de l'Alliance pour l'Agilité



« Agile Alliance supports those who explore and apply Agile principles and practices in order to make the software industry more productive, humane and sustainable. We share our passion to deliver software better everyday »
http://www.agilealliance.org

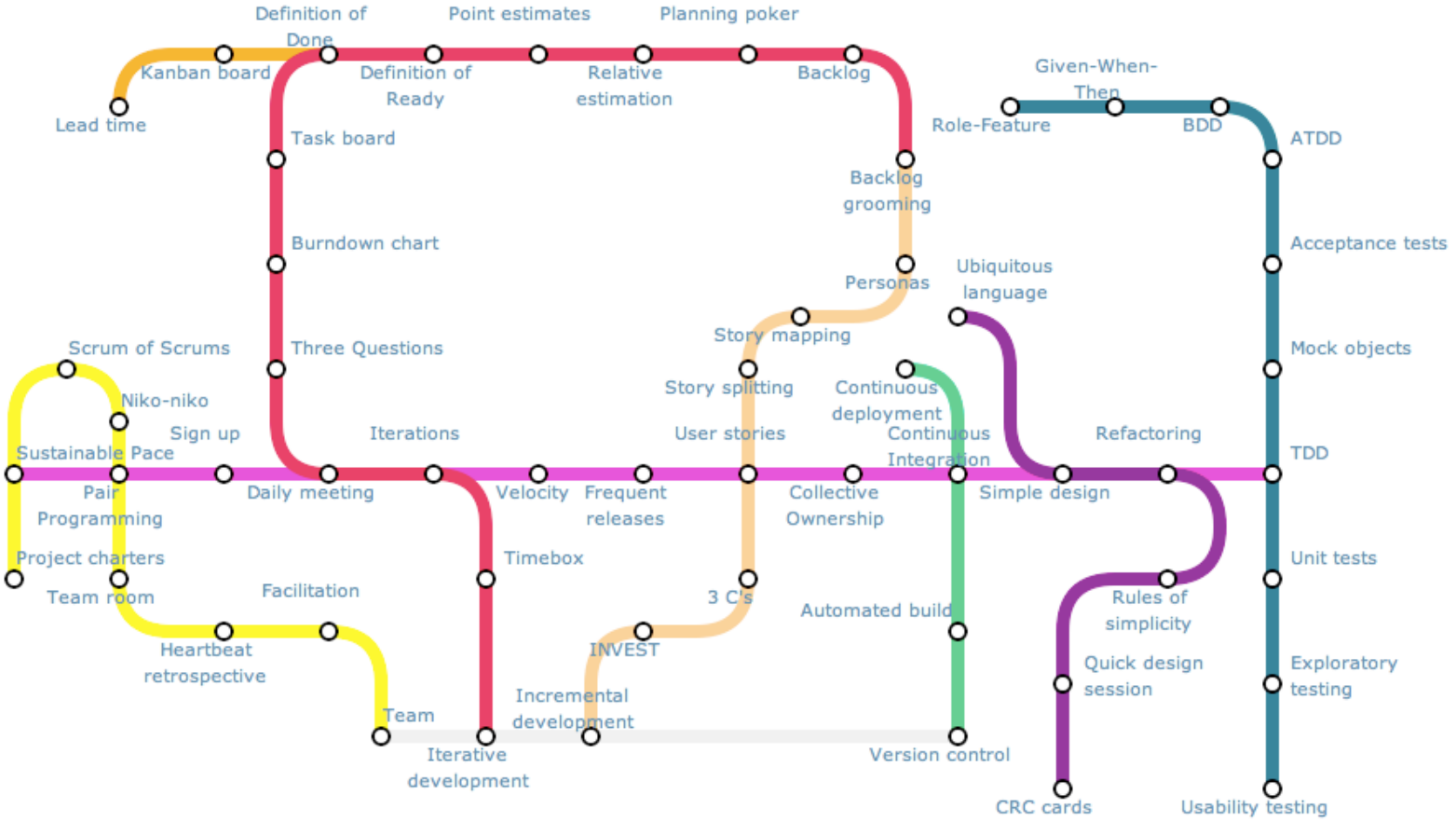
Le développement agile

- Nombreuses et courtes itérations
- Structure itérative, incrémentale et adaptative



https://commons.wikimedia.org/wiki/File:Agile_Software_Development_methodology.svg

Carte de l'agilité / flexibilité



http://guide.agilealliance.org

Pour en savoir plus

- http://www.agiledata.org/
- Video Learning Center
  - http://www.agilealliance.org/resources/learning-center/
- Livre *Test-Driven Development By Example, Kent Beck, Three Rivers Institute, http://www.cse.yorku.ca/course_archive/2003-04/W/3311/sectionM/case_studies/money/KentBeck_TDD_byexample.pdf*

Qualité d'un logiciel

- Aspect très important et trop souvent négligé
- **La qualité du logiciel est définie par son aptitude à satisfaire les besoins des utilisateurs**
- Dans le domaine du logiciel, satisfaire les besoins de l'utilisateur suppose une démarche qualité qui prenne en compte :
  - la qualité du processus de développement (coûts, délais, méthodes, organisation, personnel, techniques, outils)
  - la qualité intrinsèque du produit (modularité, simplicité, ...)
  - la qualité du service fourni par le logiciel en exploitation

La qualité du processus de développement

- Basée sur l'utilisation de méthodes de développement et de gestion de projet
- Généralement définies dans le Manuel Qualité de l'entreprise rédigé au cours de la mise en place d'une politique d'assurance qualité
- L'évaluation de la qualité intrinsèque du logiciel est effectuée sur le produit en développement en fonction des facteurs de qualité attendus, définis lors de la commande (spécifications)
- La qualité du service porte sur le logiciel en exploitation chez l'utilisateur (ou client) et consiste notamment à vérifier son adéquation aux exigences spécifiées
- La qualité dépend entièrement de la conception et de la réalisation du logiciel

Critères de qualité

- **Validité** : répond aux besoins exprimés dans le cahier des charges
- **Fiabilité** : capacité à fonctionner dans des conditions anormales
- **Extensibilité** : facilité avec laquelle il est possible d'ajouter des fonctionnalités au logiciel
- **Réutilisabilité** : aptitude à être réutilisé, même partiellement, dans d'autres applications
- **Compatibilité** : capacité à être combiné avec d'autres
- **Efficacité** : utilisation optimale des ressources matérielles et du temps
- **Portabilité** : facilité avec laquelle le logiciel peut être utilisé sur d'autres plateformes
- **Vérifiabilité** : facilité avec laquelle on peut vérifier et valider la qualité du logiciel, tests
- **Intégrité** : protection du code et des données
- **Facilité d'emploi** :aspect utilisateur, ergonomie. L'ergonomie est une exigence grandissante chez les utilisateurs

Les tests de logiciel

- Pour assurer la qualité d'un logiciel, il est nécessaire de faire des **tests** fréquents et complets
- Au niveau de la conception, on ne parlera pas de « test » mais de « validation » des choix de conception effectués
- Au niveau développement, les tests visent essentiellement à chasser les **bugs**, quels qu'ils soient (comportement incorrect, inattendu ou manquant)
- Les bugs peuvent provenir des erreurs de conception ou de programmation
- La gravité du dysfonctionnement peut aller de
  - très mineure (apparence légèrement incorrecte d'un élément d'interface graphique)
  - à des évènements en passant par des pertes plus ou moins grandes de données, et, plus rarement, par une détérioration du matériel