IFT3912 – Développement et maintenance de logiciels

Développement agile

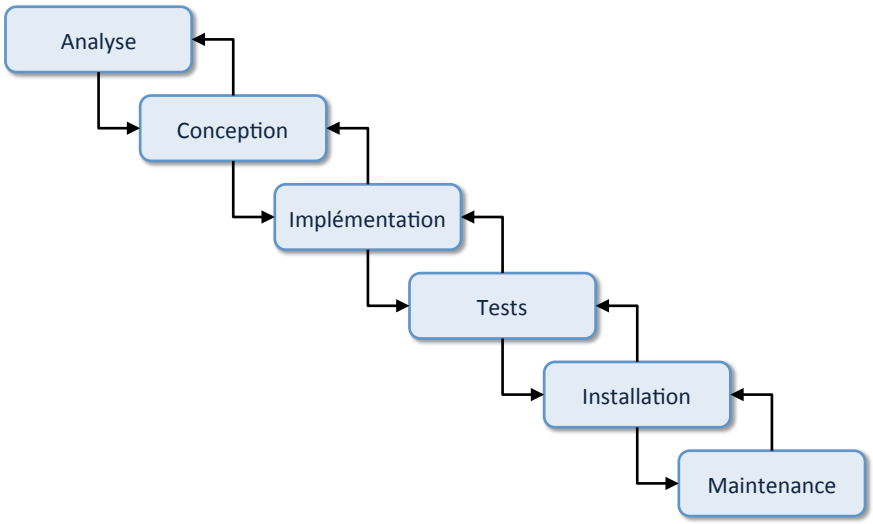
Bruno Dufour
Université de Montréal
dufour@iro.umontreal.ca

Modèles de développement

- Développement en cascade: modèle séquentiel
- Développement itératif: modèle cyclique, incrémental
- Développement par réutilisation: basé sur la réutilisation massive de composants existants
  - Peut être combiné aux autres modèles de développement
  - Ex: Composants COTS (*commercial off-the-shelf*), J2EE, intergiciels, services web, etc.

2

Processus de développement traditionnel



3

Développement en cascade

- Modèle de développement séquentiel basé sur les documents (*document-driven*)
  - Analyse: clarifier et établir un ensemble de besoins
  - Conception: concevoir un système basé sur ces besoins
  - Implémentation: programmer conformément à la spécification résultant de l'analyse et de la conception
  - Tests: évaluer la qualité du logiciel produit
  - Installation: livraison / déploiment du logiciel chez le client

4

Problèmes du modèle en cascade

- Rigidité du processus
  - Analyse: le processus fait l'hypothèse que les besoins peuvent être complètement précisés et capturés au début du projet
    - Les changements aux besoins sont très communs
    - Le client peut changer d'avis après avoir utilisé le logiciel
  - Conception: le processus fait l'hypothèse que la conception peut être complétée avant de débuter l'implémentation
    - Irréaliste dans les cas où la solution n'est pas bien comprise
    - Un changement des besoins peu invalider la conception

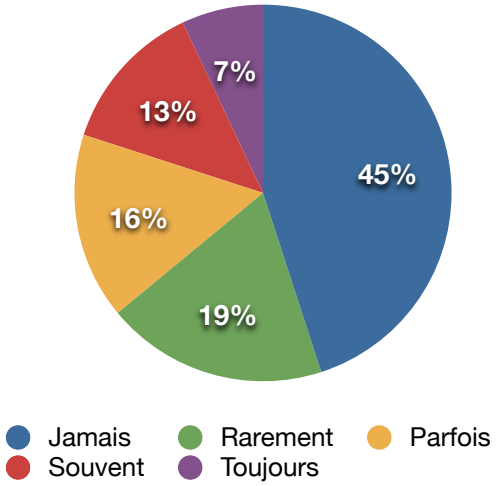
5

Problèmes du modèle en cascade

- Difficulté à s'adapter aux changements
  - Les changements imprévus peuvent invalider beaucoup de travail des phases antérieures
- Le client ne voit que le système final
  - La quasi-totalité du développement s'effectue sans commentaires du client
    - Le développement peut facilement dévier des besoins réels du client
    - Le client modifie souvent sa perception du problème et par conséquent ses besoins après avoir utilisé le système
- Les problèmes difficiles ne sont pas attaqués en premier
  - Peut mener à l'abandon de certains projets

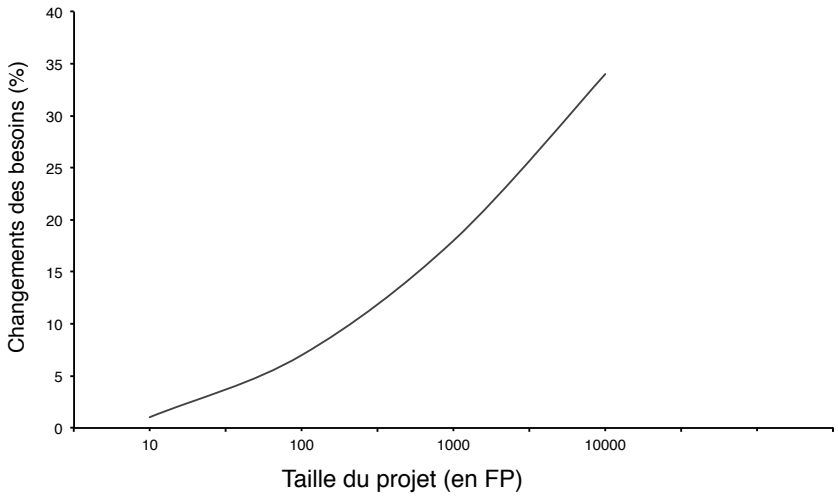
6

Utilisation des fonctionnalités demandées



7

Le changement dans le développement



Source: Craig Larman, "Applying UML and Patterns", Prentice Hall 2004.

8

Coût des changements

- Les changements sont coûteux puisque qu'ils nécessitent de refaire une partie du travail
- 2 stratégies principales pour réduire ce coût
  - **Éviter les changements**: on inclut des activités au projet qui permettent d'anticiper les changements avant que beaucoup de travail ne soit à refaire
    - Ex: utilisation d'un prototype
  - **Tolérer les changements**: on utilise un processus de développement qui permet de s'adapter facilement aux changements qui surviennent
    - Ex: développement itératif

9

Faire face aux changements

- Prototype: une version du système partiel ou complet développé rapidement
  - Permet d'obtenir les commentaires du client très rapidement, et de réduire le risque de changements des besoins (évite les changements)
  - Permet d'explorer des solutions spécifiques et d'évaluer leur faisabilité
  - Facilite la conception des interfaces graphiques
- Le prototype est généralement composé de code jetable qui ne sera pas intégré au système final

10

Faire face aux changements

- Livraison incrémentale: des versions successives et partielles du système sont livrées au client pour permettre l'expérimentation et recueillir ses commentaires
  - Le processus facilite l'apport de changements au système
  - Les fonctionnalités du noyau sont implémentées rapidement et donc testées de manière plus approfondie
  - Les utilisateurs peuvent se familiariser avec le système plus rapidement (contrairement au prototype, le système partiel sera intégré au système final)
  - Le client peut bénéficier rapidement d'un système partiel

11

Développement itératif & méthodologies agiles

12

Développement itératif

- Le cycle de vie du logiciel est constitué de mini-projets appelés **itérations**
  - Chaque itération inclut des activités d'analyse, de conception, d'implémentation et de test
  - Le but d'une itération est de produire une version stable, testée et **partielle** d'un logiciel
  - Le résultat d'une itération n'est pas un prototype, mais une version incomplète du système final
  - Le résultat de la dernière itération est le logiciel complété.
  - Les versions intermédiaires peuvent être internes ou externes
  - Chaque itération ajoute des fonctionnalités au logiciel
  - Occasionnellement, une itération peut être consacrée à l'amélioration (ex: performance), sans ajout de fonctionnalité

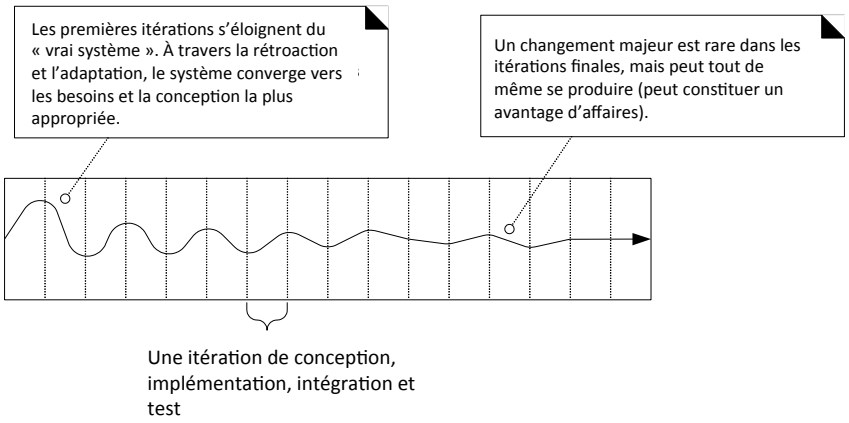
13

Développement itératif et évolution

- Le système s'agrandit progressivement, itération par itération
  - Le système peut ne devenir convenable pour un environnement de production qu'après plusieurs itérations
- Le cycle de vie du logiciel est basé sur le raffinement successif et l'évolution du système à travers une série d'itérations avec rétroaction et adaptation
  - Les besoins doivent être stables à l'intérieur d'une itération pour éviter le chaos
  - Aucune activité ne devrait être ajoutée ou modifiée au cours d'une itération
  - Des activités peuvent être remises à d'autres itérations si le travail ne peut être complété à temps

14

Évolution et convergence



Source: Craig Larman, "Applying UML and Patterns", Prentice Hall 2004.

15

Durée des itérations

- Chaque projet doit fixer une durée pour les itérations
  - La durée typique d'une itération est 2-6 semaines
  - Une durée de 2-3 semaine est très répandue
- Des itérations courtes permettent la rétroaction et l'adaptation rapide aux changements
  - Des itérations longues augmentent les risques
- Les itérations ont toutes la même durée (*timeboxing*)
  - Si certaines tâches ne peuvent être effectuées à temps, elle sont remises à une itération ultérieure
  - Une itération ne doit **jamais** se terminer en retard

16

Avantages du développement itératif

- Permet les changements
  - Le développement peut s'adapter aux changements au cours des itérations subséquentes.
- Permet l'évolution des besoins
  - Les utilisateurs peuvent donner leurs commentaires suite à l'utilisation du système opérationnel
  - Les nouveaux besoins peuvent être pris en compte de façon incrémentale
- Réduction des risques
  - Les risques sont identifiés rapidement
- Architecture robuste
  - L'architecture peut être évaluée et améliorée très tôt

17

Méthodologies agiles

18

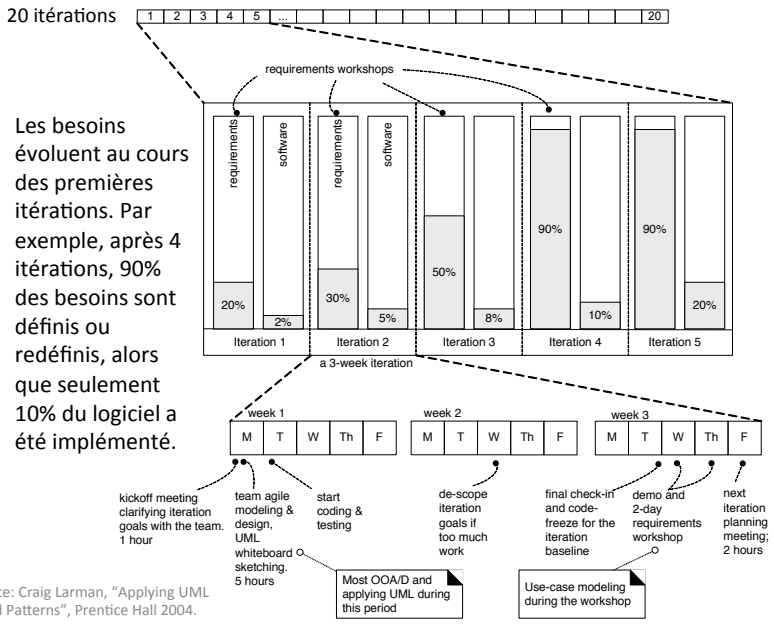
19

Caractéristiques des méthodologies agiles

- Agilité : réponse rapide et flexible aux changements
- Plusieurs méthodologies existent, mais ont des caractéristiques communes
  - Itérations à durée fixe
  - Développement évolutif
    - Raffinement progressif des besoins et de la conception
  - Planification adaptive
  - Livraisons incrémentales

20

Analyse et conception évolutives



Source: Craig Larman, "Applying UML and Patterns", Prentice Hall 2004.

21

Le manifeste Agile

« Nous découvrons de meilleurs façons de développer des logiciels en faisant et en aidant d'autres à le faire. Par ce travail, nous en sommes venus à préférer: »

- Les individus et interactions — aux processus et outils
- Les logiciels fonctionnels — à la documentation complète
- La collaboration avec le client — à la négociation de contrat
- Répondre aux changements — à suivre un plan

« Bien que nous reconnaissons la valeur des items de droite, vous valorisons les items de gauche. »

22

Les principes agiles

1. Notre priorité la plus importante est de satisfaire le client grâce à la livraison rapide et continue de logiciels utiles
2. Accueillez l'évolution des besoins, même tard dans le développement. Les processus agiles exploitent les changements pour l'avantage concurrentiel du client.
3. Livrez des logiciels fonctionnels fréquemment, de quelques semaines à quelques mois, avec une préférence pour les échelles de temps courtes.
4. Les gens d'affaires et les développeurs doivent travailler ensemble à chaque jour au cours du projet.
5. Construisez des projets autour d'individus motivés. Donnez-leur l'environnement et le soutien dont ils ont besoin, et faites-leur confiance pour effectuer le travail.

23

Les principes agiles

6. La méthode la plus efficace de transmettre des informations vers et au sein d'une équipe de développement est la conversation face-à-face.
7. Un logiciel fonctionnel est la principale mesure de progrès.
8. Les processus agiles font la promotion du développement durable.
9. Les promoteurs, les développeurs et les utilisateurs devraient être en mesure de maintenir un rythme constant indéfiniment.
10. Une attention continue à l'excellence technique et à la bonne conception améliore l'agilité.

24

Les principes agiles

11. La simplicité — l'art de maximiser la quantité de travail non effectué —, est essentielle.
12. Les meilleures architectures, exigences, et conceptions émergent des équipes qui s'auto-organisent.
13. À intervalles réguliers, l'équipe réfléchit sur la façon de devenir plus efficace, puis ajuste son comportement en conséquence.

25

Priorités du mouvement agile

- Communication et rétroaction
  - Dans l'équipe et avec le client
- Individus et équipe
  - Le développement est une activité humaine
  - Des développeurs heureux sont plus productifs
- Simplicité
  - Processus et outils
- Processus empirique
  - Modifié dynamiquement en se basant sur des mesures
- État d'esprit
  - Le développement agile est plus qu'un processus

26

Gestion de projet agile

- La planification et le contrôle sont effectués en équipe, et non par un chef de projet
  - Inclut le WBS, échéancier, estimés, etc.
- Le chef de projet ne dit (généralement) aux autres membres de l'équipe quoi faire
- Le chef de projet n'assigne pas les rôles et les responsabilités
- Le rôle du chef de projet inclut l'entraîner le personnel, fournir les ressources nécessaires, éliminer les obstacles, maintenir la vision, promouvoir la méthodologie agile, etc.

27

Les méthodes agiles en pratique

- La recherche montre que les projets ayant le plus de succès possèdent certaines caractéristiques favorisées par les méthodes agiles :
  - Cycle de vie itératif
  - Intégration quotidienne
  - Beaucoup de rétroaction et de participation des utilisateurs/clients
  - Courte durée / taille du projet
  - Livraisons incrémentales

28

Méthodologies agiles

- Processus unifié ([R]UP) agile
- Scrum
- Extreme Programming (XP)
- Et bien d'autres
  - Crystal
  - Evo
  - etc.

29

Processus Unifié

30

Processus Unifié

- Processus de développement itératif qui vise la production d'un système de haute qualité qui rencontre les besoins du client selon un budget et un échéancier prévisibles
  - Un cadre flexible plutôt qu'un processus rigide
  - Peut être adapté et personnalisé par une entreprise ou une équipe selon leurs besoins spécifiques
  - Presque tous les artéfacts (documents, modèles, etc.) et activités sont optionnels
- Pas de plan détaillé
  - Un plan de phase (haut niveau): date de fin de projet, jalons
  - Plan d'itération: développé pour la prochaine itération

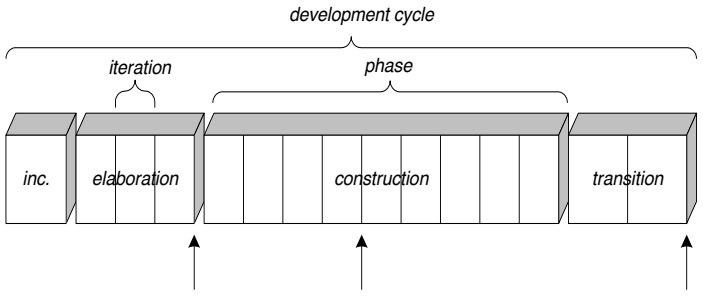
31

Phases du processus unifié

1. Création (*Inception*)
   - Vision approximative, définition de l'étendue du projet, estimés vagues
   - Le projet est-il réalisable?
2. Élaboration
   - Vision raffinée, développement itératif de l'architecture de base, résolution des risques les plus élevés, identification de la plupart des besoins, estimés plus réalistes
3. Construction
   - Implémentation itérative des éléments plus simples / à plus faible risque, préparation pour le déploiement
4. Transition
   - Tests « béta », déploiement

32

Phases et itérations



milestone

An iteration end-point when some significant decision or evaluation occurs.

release

A stable executable subset of the final product. The end of each iteration is a minor release.

increment

The difference (delta) between the releases of 2 subsequent iterations.

final production release

At this point, the system is released for production use.

Source: Craig Larman, "Applying UML and Patterns", Prentice Hall 2004.

33

Phase de création

- La création vise à répondre à plusieurs questions :
  - Quelle est la vision du projet?
  - Le projet est-il réalisable?
  - Que doit-on construire et/ou acheter?
  - Quel est le coût estimé du projet?
  - Devrait-on aller de l'avant?
- Si le projet a suffisamment de valeur, une analyse plus approfondie sera menée durant l'élaboration
- Plusieurs artéfacts peuvent être produits: exposé général du projet, cas d'utilisation , analyse de risques, prototypes, etc.
  - Les artéfacts seront complétés lors des phases subséquentes

34

Phase d'élaboration

- Au cours de l'élaboration, la plupart des besoins deviennent bien compris et définis
- L'implémentation est concentrée sur deux principaux aspects :
  - Les parties du systèmes qui présentent plus de risques
  - Les parties centrales de l'architecture du système (noyau)

35

Phases de construction et de transition

- La construction vise a terminer implémentation du système
  - Peut inclure l'analyse des besoins restants à définir
  - Inclut souvent des tests de système (*alpha-testing*)
- La construction se termine lorsque
  - le système est prêt pour le déploiement
  - tout le matériel de support est prêt (manuel d'utilisation, matériel d'entraînement, etc.)
- La transition vise à implanter le système dans un environnement de production
  - Peut inclure des tests additionnels (*beta-testing*)

36

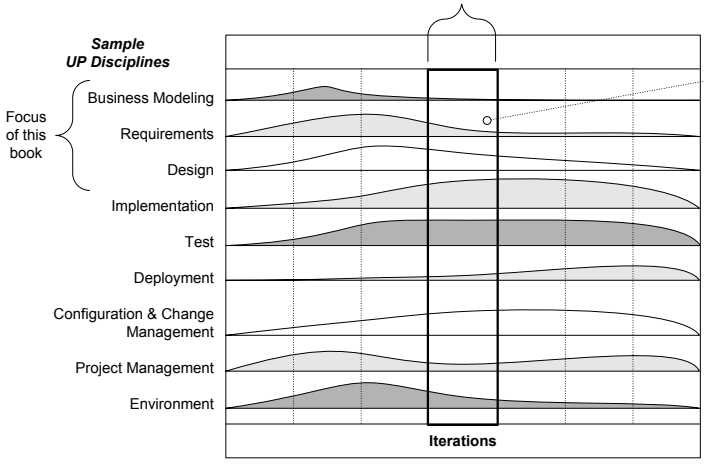
Disciplines du PU

- Disciplines d'ingénierie
  - Modélisation du processus d'affaires: permet de visualiser et définir les concepts importants du domaine de l'application
  - Analyse des besoins
  - Conception
  - Implémentation (inclut les tests unitaires)
  - Test
  - Déploiement
- Disciplines de support
  - Gestion des configurations et des changements
  - Gestion de projet
  - Environnement: mise en place des outils nécessaires et personnalisation du processus (souvent en continu)

37

Disciplines du PU

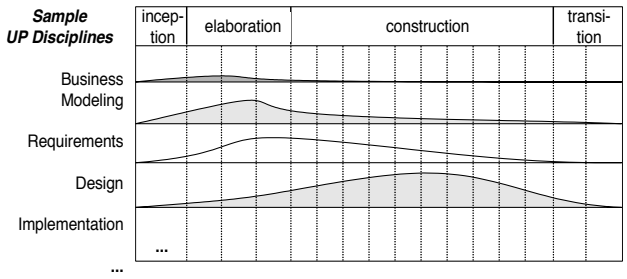
Une itération inclut du travail dans plusieurs disciplines.



Source: Craig Larman, "Applying UML and Patterns", Prentice Hall 2004.

38

Disciplines et phases



L'effort associé à chaque discipline varie selon la phase du développement.

Source: Craig Larman, "Applying UML and Patterns", Prentice Hall 2004.

39

Autres pratiques du PU

- Bâtir un noyau uni durant les premières itérations
- Continuellement vérifier la qualité
  - Tester tôt, souvent et de façon réaliste
- Appliquer les cas d'utilisation lorsqu'ils sont appropriés
- Utiliser la modélisation visuelle (UML)

40

PU vs modèle traditionnel

- Le modèle en cascade ne doit pas être superposé au PU :
  - Création ≠ analyse, élaboration ≠ conception, etc.
    - La majorité de l'analyse est effectuée durant l'élaboration
  - L'analyse, la conception et l'implémentation font partie de chaque phase, à différents degrés
- Les itérations sont nettement plus courtes (mois vs semaines)
- Le PU n'impose pas d'activités ou d'artéfacts

41

Programmation extrême

42

Introduction

- La programmation extrême (Extreme Programming, XP) est une méthodologie agile très répandue
  - Idée principale: pousser à l'extrême les bonne pratiques et valeurs de développement
  - Par exemple: les tests sont utiles donc
    - les tests seront effectués chaque jour
    - les tests seront développés avant de coder
- XP utilise un modèle de développement itératif avec itérations très courtes (1-3 semaines).

43

Pratiques

- Client membre de l'équipe
- Jeu de la planification / histoires d'utilisateurs
- Programmation en paires
- Développement piloté par les tests
- Réusinage fréquent
- Propriété collective du code
- Intégration continue
- (et quelques autres pratiques qui ne seront pas abordées dans le cours)

44

Client membre de l'équipe

- Le client est considéré comme un membre de l'équipe à part entière
  - Le client est ultimement la personne qui devra utiliser le système.
  - Un représentant du client devrait être disponible en tout temps pour répondre aux questions.
- Rôles du client
  - Écrire les histoires d'utilisateurs
  - Écrire les tests d'acceptation
  - Choisir les histoires lors du jeu de la planification

45

Histoires d'utilisateurs

- XP exprime les besoins sous forme d'**histoires d'utilisateurs**
  - Les histoires ne contiennent que suffisamment d'information pour pouvoir estimer l'effort requis pour développer une fonctionnalité
  - L'estimé est produit rapidement pour permettre à l'équipe de prendre des décisions
- Les histoires sont généralement inscrites sur des cartes qui peuvent être manipulées et placées sur un mur ou tableau

46

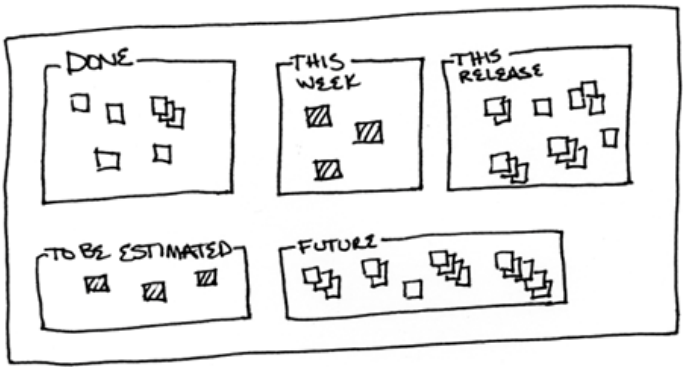
Exemple – Carte d'histoire d'utilisateur

Enregistrement avec compression 8 hrs

Présentement, les options de compressions se trouvent dans une boîte de dialogue subséquente à celle de la sauvegarde. Il faut les déplacer vers le dialogue d'enregistrement lui-même.

47

Exemple – Disposition des cartes



Source: Kent Beck, "Extreme Programming Explained: Embrace Change", Addison-Wesley 2004.

48

Jeu de la planification

- Pour une itération :
  - But : choisir les histoires à implémenter, planifier et allouer les tâches à effectuer
  - Le client choisit les cartes pour la prochaine itération, les développeurs créent les tâches.
  - Une tâche qui est estimée à > 2 jours est divisée à nouveau.
- Pour la livraison d'une version du logiciel
  - But : définir la portée de la prochaine version opérationnelle
  - Typiquement, le client écrit les histoires et les développeurs en font l'estimation au cours d'un atelier d'une demi-journée
  - Les cartes peuvent être choisie en fonction d'une fixe, ou la date peut être déterminée en fonction des cartes sélectionnées.

49

50

Programmation en paires

- Tout le code est écrit par une paire de programmeurs travaillant ensemble sur une même machine
  - Un programmeur écrit le code et les tests pendant que l'autre observe dans le but d'identifier les erreurs et les améliorations potentielles
  - Les rôles sont inversés fréquemment
- Les paires sont redistribuées fréquemment (ex: une fois par jour)
  - Cette pratique permet de distribuer les connaissances du code rapidement à travers l'équipe
- En pratique, le code produit contient moins d'erreur sans affecter la productivité de façon mesurable

51

52

Développement piloté par les tests

- Le code de production est écrit de façon à faire réussir des tests unitaires
  - Avant d'implémenter une fonctionnalité, un test est ajouté. Ce test échoue lorsque la fonctionnalité n'est pas présente.
  - Le code qui fait réussir le test est écrit.
  - Le code est développé en itérations successives d'écriture de tests et de code.
  - L'utilisation d'outils de test automatisés est essentielle à la réussite de cette approche.
- Résultat :
  - un jeu de tests unitaires très complet est développé pour le système entier
  - très peu de code inutile est développé

53

Réusinage fréquent

- But : améliorer le code sans changer sa fonctionnalité
  - Idéalement effectué à l'aide d'outils automatisés.
  - Les tests existants permettent de vérifier que le comportement observable demeure inchangé.
  - Exemples de réusinages : restructurer la hiérarchie des classes, éliminer la duplication
  - Motivation : la conception est plus efficace lorsque effectuée près de son utilisation
  - Résultat : le système demeure plus simple et plus facile à développer pour une plus longue période
- (plus de détails à venir dans la 2^{e} partie du cours)

54

Intégration continue

- « Diviser, régner et intégrer » : tous les changements doivent être testés et intégrés fréquemment
  - Fréquemment = plusieurs fois par jour
  - Peut nécessiter la résolution de conflits si plusieurs programmeurs ont modifié le même code
- 2 stratégies :
  - Intégration asynchrone : les tests sont exécutés après un changement ou à intervalles réguliers, les programmeurs sont avisés si leurs changements causent des régressions.
  - Intégration synchrone : les programmeurs exécutent eux-mêmes les tests et attendent les résultats avant de passer à la prochaine étape (permet de conserver le contexte, favorise la communication).

55

Propriété collective du code

- Une paire de programmeurs peut travailler sur n'importe quelle partie du code et l'améliorer
  - Les développeurs ne sont pas responsables d'un ou plusieurs modules en particulier
  - Motivation : si une partie du code est défectueuse, elle devrait être corrigée le plus tôt possible.
- Attention aux risques par contre!
  - Cette pratique est à éviter si l'équipe n'a pas encore développé une responsabilité collective
  - Un membre de l'équipe ne doit pas modifier du code sans se soucier de celui qui aura à le modifier et le maintenir.

56

XP – Phases de développement

1. Exploration
   - But : produire suffisamment de cartes d'histoires pour une première version, déterminer si le projet est réalisable.
   - Activités : prototypes, programmation exploratoire (preuve de technologie), écriture de cartes et estimation
2. Planification :
   - But : déterminer la date et les cartes de la première livraison
   - Activités : jeu de planification (version), écriture de cartes et estimation
3. Itérations et première livraison :
   - But : Implémenter un système complet prêt à être livré.
   - Activités : programmation, tests, jeu de planification (itérations) élaboration des tâches et estimation

57

XP – Phases de développement

4. « Productisation »
   - But : déploiement
   - Activités : documentation, formation, commercialisation, etc.
5. Maintenance
   - But : corrections, améliorations, nouvelles versions
   - Activités : répétition des phases précédentes

58

XP - Valeurs

- Communication
  - Des problèmes de communication sont à la base de la majorité des difficultés de projet
  - XP fait la promotion de la communication :
    - entre programmeurs : programmation en paires, réunion quotidienne, jeu de la planification
    - avec le client : tests d'acception, jeu de la planification
- Simplicité
  - « faire la chose la plus simple qui puisse fonctionner »
  - S'applique aux besoins, à la conception, artéfacts de projet, etc.

59

XP - Valeurs

- Rétroaction
  - Court terme: développement piloté par les tests unitaires, intégration continue, cartes d'histoires
  - Long terme: tests d'acceptation, itérations courtes (permettent au client de clarifier les besoins)
- Courage
  - Le courage de développer rapidement et d'effectuer des changements rapides découle des autres valeurs et pratiques de la programmation extrême et des outils modernes
  - Par exemple, effectuer des changements architecturaux sans un ensemble de tests et des outils automatisés est difficile et risqué.

60

Scrum

Introduction

- *Scrum* est une méthodologie agile axée sur la gestion de projet
  - Complémentaire à d'autres pratiques agiles
  - L'origine du nom est un terme de Rugby : mêlée
  - Analogie: les membres de l'équipe doivent atteindre l'objectif en équipe, comme les joueurs qui se passent le ballon.

Source: Wikipedia

Caractéristiques

- Processus itératif
  - Itérations plus longues que d'autres méthodologies (30 jours)
- Équipe auto-gérée
  - Pas de processus rigide
  - Le développement est adapté empiriquement
- Réunion debout quotidienne (mêlée, ou *scrum*)
- Démonstrations du système après chaque itération
- Planification impliquant le client après chaque itération

63

Phases

1. Planification
   - Établir la vision du projet, les attentes et assurer le financement
   - Activités : définition du carnet de produit initial, estimés, conception exploratoire, prototypes
2. Mise en scène (*staging*)
   - Identification de plus de besoins, priorisation suffisante pour une première itération
   - Activités : planification, conception exploratoire, prototypes
3. Développement
   - Implémentation d'un système par une série d'itérations de 30 jours (*sprints*)
   - Activités : planification de sprint, définition du carnet de sprint, mêlée quotidienne, revue de sprint
4. Livraison d'une version du système (*release*)
   - Déploiement
   - Activités : formation, documentation, commercialisation, etc.

Avant-match

64

Rôles

- « Scrum Master »
  - Élimine les obstacles
    - Idéalement en < 1 jour (avant la prochaine mêlée)
  - Prend des décisions lorsque nécessaire
    - Idéalement en < 1 heure: une mauvaise décision est préférable à aucune décision
  - Agit comme écran (*firewall*) pour s'assurer que l'équipe n'est pas interrompue par des requêtes venant de l'extérieur
  - Renforce la vision du projet
  - Un Scrum Master inefficace doit être remplacé
- Équipe
  - Scrum recommande que les équipes soient limitées à 7-10 personnes
  - Les grands projets contiennent plusieurs équipes

65

Rôles

- Propriétaire du produit (*Product Owner*)
  - Un représentant du client
  - Assigne les priorités dans le carnet du produit
  - Choisit les besoins à inclure dans une itération
- « Chickens »
  - Personnes qui ne participent pas directement au projet mais dont les intérêts doivent être pris en compte
    - Ex: utilisateurs, clients (autre que le propriétaire du produit), gestionnaires, etc.
  - En référence à une blague :

A pig and chicken discussed the name of their new restaurant. The chicken suggested *Ham n' Eggs*. « No thanks, » said the pig, « I'd be committed, but you'd only be involved! »

66

Carnets (*Backlogs*)

- Carnet de produit
  - Appartient au propriétaire du produit
  - Contient les besoins de haut niveau, leur priorité, leur value d'affaire et un estimé de l'effort requis
  - Durant la planification d'avant-match, tous les intervenants (*stakeholders*) peuvent ajouter des fonctionnalités, des cas d'utilisation, des améliorations et des défauts au carnet de produit
- Carnet de sprint
  - Appartient à l'équipe
  - Contient des tâches et un estimé de l'effort requis / restant

67

Exemple – Carnet du produit

Besoin	Num.	Cat.	État	Pri.	Est.	Sprint
Enregistrement des paiements dans le registre	17	Fonctionnalité	En cours	5	2	1
Plan de financement à long terme	232	Amélioration	Terminé	4	38	1
Calcul de la commission de vente	12	Défaut	Pas débuté	2	14	2
Approbation de crédit lente	321	Problème	En cours	5	2	1

68

Exemple – Carnet de sprint

			Heures de travail restantes				
Besoin	**Resp.**	**État**	**6 362**	**7 322**	**8 317**	**9 317**	**10 306**
Rencontre pour discuter des objectifs	JM/SR	Terminé	20	10	0	0	0
Déplacer les calculs hors du module …	AW	Pas débuté	8	8	8	8	8
Obtenir les données …	TN	Terminé	12	0	0	0	0
Créer et initialiser la base de données	GP	En cours	24	20	30	25	20
…							

69

Sprints

- Deux réunions sont tenues avant le début d'une itération (*sprint*)
  - 1ère réunion : les intervenants priorisent le carnet de produit, habituellement en fonction de la valeur d'affaires et des risques.
  - 2ème réunion : le propriétaire du produit et l'équipe déterminent comment atteindre les objectifs demandés, et produisent le carnet de sprint
    - Le carnet de sprint est mis à jour à mesure que l'itération progresse.

70

Mêlée quotidienne (*scrum*)

- Tenue à chaque jour, à la même heure et au même endroit
  - Doit débuter à l'heure, il est fréquent d'imposer des amendes aux retardataires qui sont ensuite utilisées comme dons de charité
- Doit répondre à 5 questions :
  - Qu'avez-vous fait depuis le scrum précédant?
  - Qu'allez-vous faire entre maintenant et le scrum suivant?
  - Qu'est-ce qui entrave l'atteinte des buts de l'itération en cours?
  - Y a t'il des tâches à ajouter au backlog?
    - Tâche manquantes, pas de nouveaux besoins
  - Avez-vous appris ou décidé quelque chose de nouveau qui serait utile aux autres membres de l'équipe?
- Aucune autre discussion n'est permise
  - Le Scrum Master peut recentre la discussion au besoin

71

Mêlée quotidienne (*scrum*)

- Idéalement tenue debout en cercle pour encourager la brièveté.
- Doit se tenir près d'un tableau où les tâches sont inscrites
- Dure en moyenne 15-20 minutes pour une équipe de 7-10 personnes
  - Des réunions plus longues sont communes au début d'une itération
- Les membres qui sont absents doivent participer par haut-parleur
- Les non-membres de l'équipe (*chickens*) sont à l'extérieur du cercle
  - Ils ne parlent pas et ne posent pas de questions
  - Exception: commentaires des gestionnaires (ex: explication de la pertinence du travail de l'équipe)

72

Revue de sprint

- À la fin de chaque itération, le Scrum Master organise la revue de sprint
  - Maximum 4 heures
  - Tous les intervenants participent à la revue
- L'équipe fait la démonstration du système au client
  - Le client est informé des fonctions du système, de la conception, des forces et faiblesses du système, de l'effort de l'équipe et des problèmes potentiels à venir
  - Le client peut donner ses commentaires
  - Pas de transparents, le produit doit être montré
  - Les engagements sont pris lors de la planification du prochain sprint, et non pas lors de la revue.

73

Combinaisons de méthodes

74

PU + XP

- La majorité des pratiques de XP sont compatibles avec celles du PU :
  - Ex: documentation minimale, TDD / vérification continue de la qualité
  - Plusieurs pratiques de XP sont une spécialisation des pratiques du PU, et peuvent donc être utilisées dans un projet basé sur le PU
- Différences :
  - Modélisation : Le PU considère une demi-journée de modélisation acceptable pour une itération de 2 semaines, alors que XP suggère une limite de 10-20 minutes.
  - Risques: Le PU met l'emphase sur les risques élevés dans les premières itérations, alors qu'il n'y a pas d'effort similaire en XP.
  - Besoins : Le PU permet et encourage la création de documents d'analyse détaillés (de façon incrémentale), pour les cas où le client n'est pas disponible sur le site. En XP, le client fait partie de l'équipe et de tels documents ne sont pas utilisés.

75

PU + Scrum

- Les pratiques de Scrum sont compatibles avec celles du PU
  - Carnet du produit (Scrum) ≈ plan de projet (PU)
  - Carnet de sprint (Scrum) ≈ plan d'itération
  - Plusieurs pratiques de Scrum sont une spécialisation des pratiques du PU, et peuvent donc être utilisées dans un projet basé sur le PU
  - Lorsque des artéfacts sont requis en Scrum, les versions du PU sont acceptables
- Conflit potentiel
  - Le PU indique certaines dépendances entre activités optionnelles (ex: la vision d'un projet est définie avant l'analyse, etc.). Scrum rejette le concept d'ordre prévisible des étapes.

76

XP + Scrum

- La plupart des pratiques sont compatibles
  - Ex: mêlée (Scrum) est un raffinement de la réunion de XP
  - La démo à la fin d'une itération (Scrum) s'inscrit dans le cadre des efforts de communication et rétroaction de XP
  - Les histoires d'utilisateurs de XP sont souvent utilisées dans les carnets de Scrum
- Conflits:
  - Les itérations de 30 jours sont trop longues pour XP
  - Scrum impose un seul représentant du client, alors que XP en supporte plusieurs

77