Plan

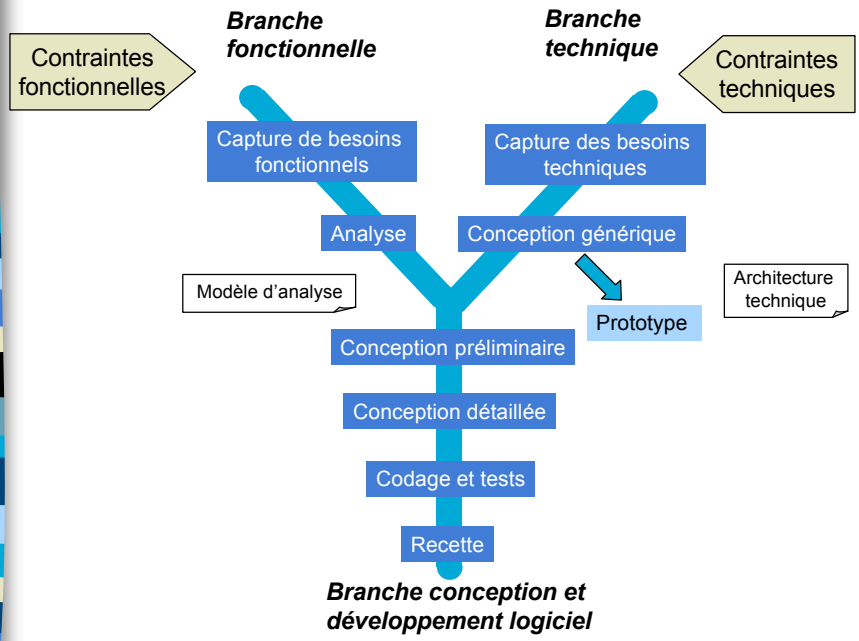
- Avant-propos
- Méthodes et processus
- Processus unifié : caractéristiques essentielles
- Description du processus unifié
- **Illustration : deux déclinaisons du processus unifié**
- Méthodes Agile
- Conclusion

Two Tracks Unified Process

- Processus proposé par Valtech (consulting), présenté dans
  - Pascal Roques, Franck Vallée (2004) UML2 en action (3ème édition), Eyrolles, 386 pp.
- Objectif
  - prendre en compte les contraintes de changement continuel imposées aux systèmes d'information des organisations
- Création d'un nouveau SI
  - deux grandes sortes de risques
    - imprécision fonctionnelle : inadéquation aux besoins
    - incapacité à intégrer les technologies : inadaptation technique
- Evolution du SI
  - deux grandes sortes d'évolutions
    - évolution fonctionnelle
    - évolution technique

Two Tracks Unified Process

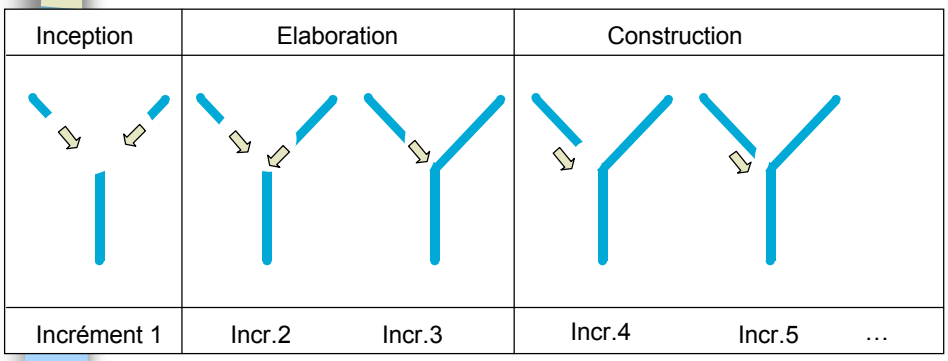
- Principe général
  - toute évolution imposée au système d'information peut se décomposer et se traiter parallèlement, suivant un axe fonctionnel et un axe technique
- TTUP
  - processus unifié (itératif, centré sur l'architecture et piloté par les CU)
  - deux branches
    - besoins techniques : réalisation d'une architecture technique
    - besoins fonctionnels : modèle fonctionnel
  - réalisation du système
    - fusionner les résultats des deux branches du processus



TTUP : commentaires

- Côté fonctionnel
  - relativement classique, indépendant des technologies
  - modèle d'analyse réutilisable
- Côté technique
  - insistance sur l'architecture technique indépendamment des besoins fonctionnels
    - c'est un problème de conception en soi
      - notion de CU techniques
    - dépend de l'existant
    - architecture à base de composant
  - architecture technique réutilisable
- Branche commune
  - conception préliminaire : le plus délicat

Itérations et TTUP

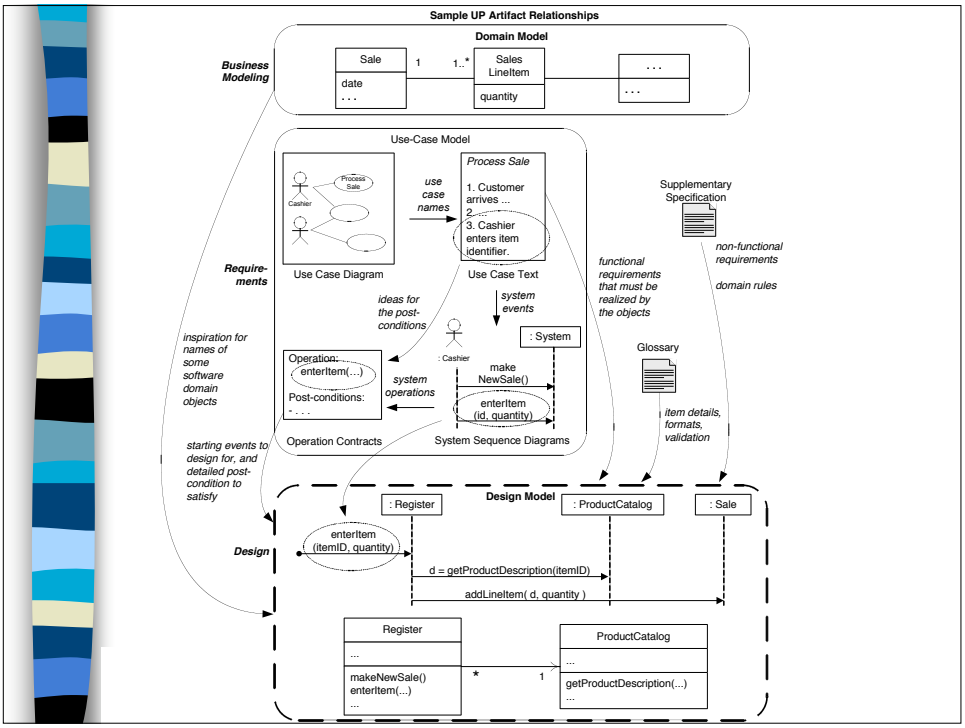


UP Agile (Larman)

- Proposé par Craig Larman, présenté dans
  - Craig Larman (2005) *UML 2 et les Design Patterns* (3e édition), Pearson Education, 655 p.
- Parce que UP est souvent considéré comme complexe, formel et lourd
  - dans sa description générale,
    - essaye de prendre en compte toutes les options possibles, pour toutes les entreprises, et toutes les tailles de projets
      - nombreux artefacts, activités, workflows
    - doit être adapté à chaque projet
      - ce dont tout le monde ne se rend pas forcément compte
  - dans son utilisation
    - application « à l'ancienne »
      - suivi strict des activités : rigidité
    - tendance à la cascade
      - les mauvaises habitudes sont difficiles à perdre

UP agile = les bonnes pratiques de UP

- Itérations courtes (3 semaines max)
- Mode organisationnel léger
  - petit ensemble d'activités et d'artefacts
- Fusion analyse / conception
- Utilisation de UML pour comprendre et concevoir
  - plus que pour générer du code
- Planification adaptative
- Bref, application de UP dans l' « esprit Agile »
  - *vs.* esprit cascade
  - en considérant que UP est agile naturellement dans sa conception (et pour ses concepteurs), mais ne l'est pas dans ses applications
- Transparents suivants
  - liens entre artefacts dans UP agile
  - mise en œuvre physique des réunions



Organisation physique : CU

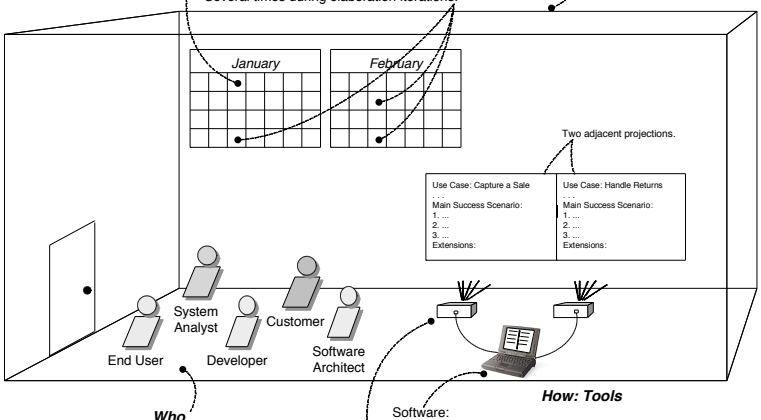
When

Once during inception. Short; do not try to define or polish all requirements.

Several times during elaboration iterations.

Where

At a requirements workshop.



Who

Many, including *end users* and *developers*, will play the role of *requirements specifier*, helping to write use cases.

Led by *system analyst* who is responsible for requirements definition.

How: Tools

Software:

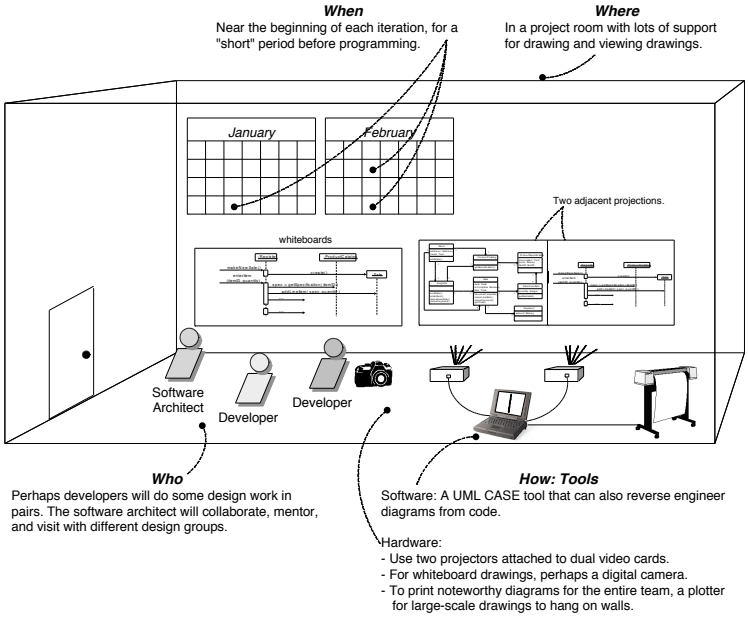
- For use case text, use a web-enabled requirements tool that integrates with a popular word processor.

For use case diagrams, a UML CASE tool.

Hyperlink the use cases; present them on the project website.

Hardware: Use two projectors attached to dual video cards and set the display width double to improve the spaciousness of the drawing area or display 2 adjacent word processor windows .

Organisation physique : modélisation objet



Plan

- Avant-propos
- Méthodes et processus
- Processus unifié : caractéristiques essentielles
- Description du processus unifié
- Illustration : deux déclinaisons du processus unifié
- **Méthodes Agile**
- Conclusion

Petite histoire des méthodes Agile

- Années 90
  - réaction contre les grosses méthodes
  - prise en compte de facteurs liés au développement logiciel
- Fin années 90
  - méthodes
    - d'abord des pratiques liées à des consultants, puis des livres
    - XP, Scrum, FDD, Crystal…
- 1991
  - les principaux méthodologues s'accordent sur le « Agile manifesto »
- Années 2000
  - projets Agile mixent des éléments des principales méthodes

Du rien au monumental à l'agile

- Rien
  - « code and fix »
  - marche bien sur les petits projets, suicidaire ensuite
- Monumental
  - méthodes, processus, contrats : rationalisation à tous les étages
  - problèmes et échecs
    - trop de choses sont faites qui ne sont pas directement liées au produit logiciel à construire
    - planification trop rigide
- Agile
  - trouver un compromis : le minimum de méthode permettant de mener à bien les projets en restant agile
    - capacité de réponse rapide et souple au changement
    - orientation vers le code plutôt que la documentation

Principes communs des méthodes Agile

- Méthodes adaptatives (vs. prédictives)
  - itérations courtes
  - lien fort avec le client
  - fixer les délai et les coûts, mais pas la portée
- Insistance sur les hommes
  - les programmeurs sont des spécialistes, et pas des unités interchangeables
  - attention à la communication humaine
  - équipes auto-organisées
- Processus auto-adaptatif
  - révision du processus à chaque itération

Méthodes agiles

- Simplicité
- Légèreté
- Orientées participants plutôt que plan
- Nombreuses
  - XP est la plus connue
- Pas de définition unique
- Mais un manifeste

Manifeste Agile

- Février 1991, rencontre et accord sur un manifeste
- Mise en place de la « Agile alliance »
  - objectif : promouvoir les principes et méthodes Agile
  - http://www.agilealliance.com/
- Les signataires privilégient
  - les individus et les interactions davantage que les processus et les outils
  - les logiciels fonctionnels davantage que l'exhaustivité et la documentation
  - la collaboration avec le client davantage que la négociation de contrat
  - la réponse au changement davantage que l'application d'un plan
- 12 principes
  - (transparents suivants)

http://agilemanifesto.org/principles.html

Manifeste Agile : 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.

http://agilemanifesto.org/principles.html

Manifeste Agile : 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

(Larman 2005, p.36-37)

Processus agile et modélisation

- Utilisation d'UML
- La modélisation vise avant tout à comprendre et à communiquer
- Modéliser pour les parties inhabituelles, difficiles ou délicates de la conception.
- Rester à un niveau de modélisation minimalement suffisant
- Modélisation en groupe
- Outils simples et adaptés aux groupes
- Les développeurs créent les modèles de conception qu'ils développeront

Dans la suite

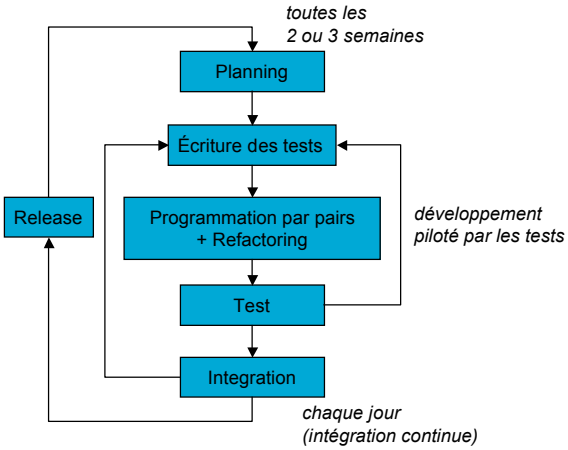
- Description rapide de quelques méthodes
  - XP (Kent Beck)
  - Scrum (Ken Schwaber)
  - Crystal (Alistair Cockburn)
- Autres méthodes
  - UP, FDD (Feature Driven Development) , DSDM (Dynamic System Development Method), …

XP (eXtreme Programming)

- Site officiel
  - http://www.extremeprogramming.org/
- Caractéristiques principales (http://www.design-up.com/methodes/XP/)
  - Le client (maîtrise d'ouvrage) pilote lui-même le projet, et ce de très près grâce à des cycles itératifs extrêmement courts (1 ou 2 semaines).
  - L'équipe autour du projet livre très tôt dans le projet une première version du logiciel, et les livraisons de nouvelles versions s'enchaînent ensuite à un rythme soutenu pour obtenir un feedback maximal sur l'avancement des développements.
  - L'équipe s'organise elle-même pour atteindre ses objectifs, en favorisant une collaboration maximale entre ses membres.
  - L'équipe met en place des tests automatiques pour toutes les fonctionnalités qu'elle développe, ce qui garantit au produit un niveau de robustesse très élevé.
  - Les développeurs améliorent sans cesse la structure interne du logiciel pour que les évolutions y restent faciles et rapides.

http://www.swen.uwaterloo.ca/~kostas/ECE355-05/lectures/Lect3-Ch15-Unit2.ppt

XP : vue d'ensemble



http://web.engr.oregonstate.edu/~cook/classes/cs569.agile.ppt

XP : rôles et responsabilités

- Programmeur
  - écrit les tests et puis code
- Client
  - écrit les histoires et les tests fonctionnels
- Testeur
  - aide le client à écrire les tests et à les exécuter
- Consultant
  - fournit les connaissances spécialisées au besoin
- Coach
  - aide l'équipe par rapport au processus
- Tracker
  - vérifie que l'équipe ne perd pas la bonne direction
- Manager
  - prend les décisions

XP : pratiques (1)

- Le « jeu de la planification »
  - regroupement des intervenants pour planifier l'itération
  - les développeurs évaluent les risques techniques et les efforts prévisibles liés à chaque fonctionnalité (*user story,* sortes de scénarios abrégés)
  - les clients estiment la valeur (l'urgence) des fonctionnalités, et décident du contenu de la prochaine itération
- Temps court entre les releases
  - au début : le plus petit ensemble de fonctionnalités utiles
  - puis : sorties régulières de prototypes avec fonctionnalités ajoutées
- Métaphore
  - chaque projet a une métaphore pour son organisation, qui fournit des conventions faciles à retenir

XP : pratiques (2)

- Conception simple
  - toujours utiliser la conception la plus simple qui fait ce qu'on veut
    - doit passer les tests
    - assez claire pour décrire les intentions du programmeur
  - pas de généricité spéculative
- Tests
  - développement piloté par les tests : on écrit d'abord les tests, puis on implémente les fonctionnalités
  - les programmeurs s'occupent des tests unitaires
  - les clients s'occupent des tests d'acceptation (fonctionnels)
- Refactoring
  - réécriture, restructuration et simplification permanente du code
  - le code doit toujours être propre

XP : pratiques (3)

- Programmation par paires (*pair programming*)
  - tout le code de production est écrit par deux programmeurs devant un ordinateur
  - l'un pense à l'implémentation de la méthode courante, l'autre à tout le système
  - les paires échangent les rôles, les participants des paires changent
- Propriété collective du code
  - tout programmeur qui voit une opportunité d'améliorer toute portion de code doit le faire, à n'importe quel moment
- Intégration continue
  - utilisation d'un gestionnaire de versions (e.g., CVS)
  - tous les changements sont intégrés dans le code de base au minimum chaque jour : une construction complète (*build*) minimum par jour
  - 100% des tests doivent passer avant et après l'intégration

XP : pratiques (4)

- Semaine de 40 heures (35 en France ?)
  - les programmeurs rentrent à la maison à l'heure
  - faire des heures supplémentaire est signe de problème
  - moins d'erreurs de fatigue, meilleure motivation
- Des clients sur place
  - l'équipe de développement a un accès permanent à un vrai client/utilisateur (dans la pièce d'à côté)
- Des standards de codage
  - tout le monde code de la même manière
    - tout le monde suit les règles qui ont été définies
    - il ne devrait pas être possible de savoir qui a écrit quoi

XP : pratiques (5)

- Règles
  - l'équipe décide des règles qu'elle suit, et peut les changer à tout moment
- Espace de travail
  - tout le monde dans la même pièce
    - awareness
  - tableaux au murs
  - matérialisation de la progression du projet
    - par les histoires (*user stories*) réalisées et à faire
      - papiers qui changent de position, sont réorganisés
    - par les résultats des tests
    - …

XP : quelques point du site web

- Planning
  - User stories are written.
  - Release planning creates the schedule.
  - Make frequent small releases.
  - The Project Velocity is measured.
  - The project is divided into iterations.
  - Iteration planning starts each iteration.
  - Move people around.
  - A stand-up meeting starts each day.
  - Fix XP when it breaks.
- Designing
  - Simplicity.
  - Choose a system metaphor.
  - Use CRC cards for design sessions.
  - Create spike solutions to reduce risk.
  - No functionality is added early.
  - Refactor whenever and wherever possible.
- Coding
  - The customer is always available.
  - Code must be written to agreed standards.
  - Code the unit test first.
  - All production code is pair programmed.
  - Only one pair integrates code at a time.
  - Integrate often.
  - Use collective code ownership.
  - Leave optimization till last.
  - No overtime.
- Testing
  - All code must have unit tests.
  - All code must pass all unit tests before it can be released.
  - When a bug is found tests are created.
  - Acceptance tests are run often and the score is published.

XP : avantages

- Concept intégré et simples
- Pas trop de management
  - pas de procédures complexes
  - pas de documentation à maintenir
  - communication directe
  - programmation par paires
- Gestion continuelle du risque
- Estimation permanente des efforts à fournir
- Insistance sur les tests : facilite l'évolution et la maintenance

XP : inconvénients

- Approprié pour de petites équipes (pas plus de 10 développeurs), ne passe pas à l'échelle
  - pour des groupes plus gros, il faut plus de structure et de documentation (ceremony)
- Risque d'avoir un code pas assez documenté
  - des programmeur qui n'auraient pas fait partie de l'équipe de développement auront sans doute du mal à reprendre le code
- Pas de design générique
  - pas d'anticipation des développements futurs

http://web.engr.oregonstate.edu/~cook/classes/cs569.agile.ppt

Scrum

- Scrum : mêlée
- Phases
  - Initiation / démarrage
    - Planning
      - définir le système : *product Backlog* = liste de fonctionnalités, ordonnées par ordred de priorité et d'effort
    - Architecture
      - conception de haut-niveau
  - Développement
    - Cycles itératifs (sprints) : 30j
      - amélioration du prototype
  - Clôture
    - Gestion de la fin du projet : livraison…

http://fr.wikipedia.org/wiki/Scrum

Principes Scrum

- 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.

http://web.engr.oregonstate.edu/~cook/classes/cs569.agile.ppt

Scrum : rôles responsabilités

- Scrum Master
  - expert de l'applicatron de Scrum
- Product owner
  - responsable officiel du projet
- Scrum Team
  - équipe projet.
- Customer
  - participe aux réunions liées aux fonctionnalités
- Management
  - prend les décisions

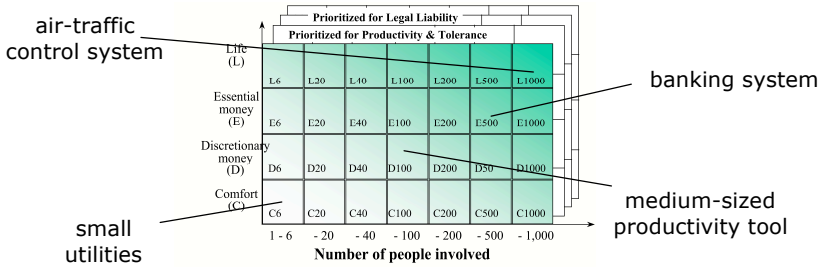
http://web.engr.oregonstate.edu/~cook/classes/cs569.agile.ppt

Scrum : pratiques

- 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

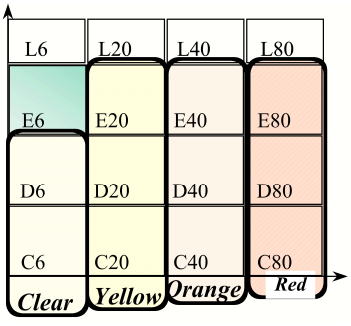
Méthodologies « Crystal »

- Alistair Cockburn (2002). *Agile Software Development.* Addison Wesley
- Le développement logiciel vu comme un jeu coopératif de communication et d'invention
  - des projets différents et des méthodes différentes
  - le projet change à mesure que les gens changent
- Choix de la méthode en fonction de différents facteurs
  - taille en nombre de personnes / criticité pour le client (LEDC)



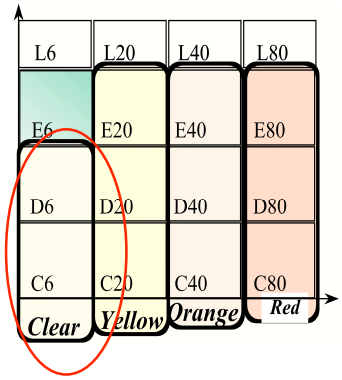
La famille de méthodes Crystal

- Familles conçues à partir de l'observation et des interviews
- Trouver à chaque fois la méthode la moins rigide qui réussira quand même
  - haute productivité, haute tolérance
  - focus sur la communication
- Exemples
  - *Crystal Clear*
  - Crystal Yellow
  - *Crystal Orange*
  - Crystal Red



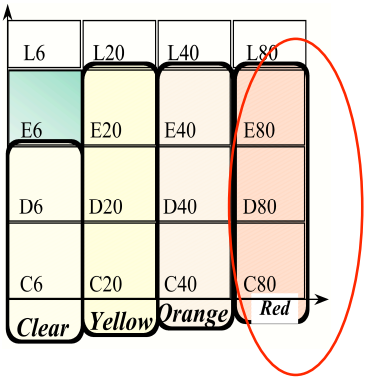
Crystal Clear

- "C6-D6"
- Une seule équipe, à un seul endroit
  - *Good, small teams produce good, small software products*
  - Communication étroite
- Rôles
  - sponsor
  - développeur sénior
  - concepteurs-programmeurs
  - utilisateurs
- Livraison incrémentale tous les 2/3 mois



Crystal Orange

- "C40-E40"
  - plus de structure d'équipes
  - lus de coordination
- Tout le monde dans le même immeuble
- Rôles
  - Sponsor
  - Expert métier
  - Expert IHM
  - Expert techniques
  - Analyste concepteur métier
  - Manager
  - Architecte
  - Concepteur mentor
  - …
- 1-2 ans de développement
- Importance du respect des délais et de l'adaptation au marché



Plan

- Avant-propos
- Méthodes et processus
- Processus unifié : caractéristiques essentielles
- Description du processus unifié
- Illustration : deux déclinaisons du processus unifié
- Méthodes Agile
- **Conclusion**

Conclusion

- « There is no silver bullet » : le retour
  Même si le développement incrémental permet de s'affranchir de beaucoup de problèmes, il y aura quand même des problèmes. Mais ceux-ci seront normalement d'ampleur plus faible, et mieux gérés.
- Toute méthode est adaptable et doit être adaptée
  Mais, lorsque l'on débute, il vaut mieux ne pas trop s'écarter de la voie décrite pour bien comprendre au départ (cf. musique)

Crédits – remerciements

- J.L. Sourrouille (INSA de Lyon)
- Sources
  - http://www.swen.uwaterloo.ca/~kostas/ECE355-05/lectures/Lect3-Ch15-Unit2.ppt
  - http://web.engr.oregonstate.edu/~cook/classes/cs569.agile.ppt

Annexe : gestion des composants et bibliothécaire

- Qualité
  - niveau plus élevé que celui d'une application (surcoût minimum 50%, en pratique 100%)
  - une confiance absolue est nécessaire pour que la réutilisation soit effective
- Une fonction : bibliothécaire
  - participe au choix des produits à généraliser
  - contribue ou procède à la généralisation
  - établit des normes, règles, niveaux de qualité, protocoles de mise en bibliothèque
  - classifie les produits de la bibliothèque (critères)
  - diffuse, informe, facilite l'accès (outils)

Annexe : coders' dojo

- Idée
  - parallèle arts martiaux / conception-programmation OO
    - il faut s'entraîner à appliquer des « routines » connues avant de pouvoir commencer à les utiliser de façon créative, voire à en inventer de nouvelles
    - les débutant doivent apprendre des maîtres
  - un exemple de formation
    - http://www.xpday.net/Xpday2005/CodersDojo.html

(O. Boissier)

Annexe : qualité du logiciel

- Facteur externes (utilisateur)
  - Correction (validité)
    - aptitude à répondre aux besoins et à remplir les fonctions définies dans le cahier des charges
  - Robustesse (fiabilité)
    - aptitude à fonctionner dans des conditions non prévues au cahier des charges, éventuellement anormales
  - Extensibilité
    - facilité avec laquelle de nouvelles fonctionnalités peuvent être ajoutées

(O. Boissier)

Qualité du logiciel

- Facteurs externes (suite)
  - Compatibilité
    - facilité avec laquelle un logiciel peut être combiné avec d'autres
  - Efficacité
    - utilisation optimale des ressources matérielles (processeur, mémoires, réseau, …)
  - Convivialité
    - facilité d'apprentissage et d'utilisation, de préparation des données, de correction des erreurs d'utilisation, d'interprétation des retours
  - Intégrité (sécurité)
    - aptitude d'un logiciel à se protéger contre des accès non autorisés.

(O. Boissier)

Qualité du logiciel

- Facteurs internes (concepteur)
  - Ré-utilisabilité
    - aptitude d'un logiciel à être réutilisé, en tout ou en partie, pour d'autres applications
  - Vérifiabilité
    - aptitude d'un logiciel à être testé (optimisation de la préparation et de la vérification des jeux d'essai)
  - Portabilité
    - aptitude d'un logiciel à être transféré dans des environnements logiciels et matériels différents
  - Lisibilité
  - Modularité