Cours Génie Logiciel – M1 Informatique

- Partie I - Introduction au Génie Logiciel
- Partie II - Test logiciel – Principes et méthodes
- Partie III – Ingéniérie des exigences

Prof. Bruno Legeard

bruno.legeard@univ-fcomte.fr

Présentation du cours Génie Logiciel – 1/3

- Objectifs du cours:
  - Développer des connaissances de base sur le cycle de développement du logiciel et les principes du GL
    - Qualité du logiciel
    - Processus et méthode de développement – Méthodes agiles / SCRUM
    - Test de logiciel
    - Ingéniérie des exigences
  - Le Génie Logiciel concerne les projets de taille moyenne à importante (de quelques dizaine de milliers de lignes de code à des millions de lignes de code) pour lesquels le travail en équipe est nécessaire ➔ c'est-à-dire la très grande majorité du développement logiciel aujourd'hui.
  - Dans le cours de GL, on ne traite pas de la programmation, mais des conditions de réussite des projets

Présentation du cours Génie Logiciel – 2/3

- **Partie I - Introduction au Génie Logiciel**
  - Qualité du logiciel
  - Processus de développement Agile – SCRUM
  - Outillage du Génie Logiciel
- **Partie II - Test logiciel – Principes et méthodes**
  - Fondamentaux des tests
  - Tester pendant le cycle de vie du logiciel
  - Techniques statiques
  - Techniques de conception de test
  - Gestion des tests
- **Partie III - Ingéniérie des exigences**
  - Fondamentaux
  - Identification, analyse et spécification des exigences

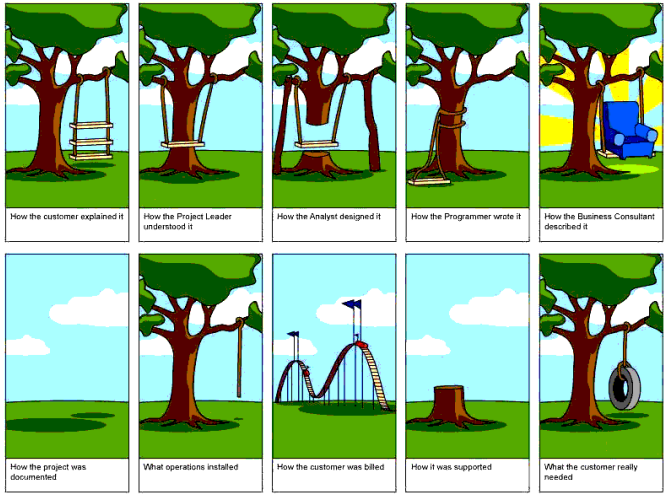
Présentation du cours Génie Logiciel - 3/3

- Comment est répartie la notation (Contrôle continu) ?
  - 30% - Questions de cours – Evaluation en TD
  - 30% - TP + Projets
  - 40% - Examen final

- Si ABI, alors 0 sur l'exam manqué
- Report 1ère session vers 2ème session : 40% de la note
- Quelles sont les documents accessibles sous Moodle ?
  - Support CM / TD / TP + correction après leur réalisation
- Le Génie Logiciel nécessite d'en connaitre certains principes et définitions (vus en cours) ➔ il y aura des questions de cours lors des contrôle continus et de l'Exam.

Ressources du cours - Moodle

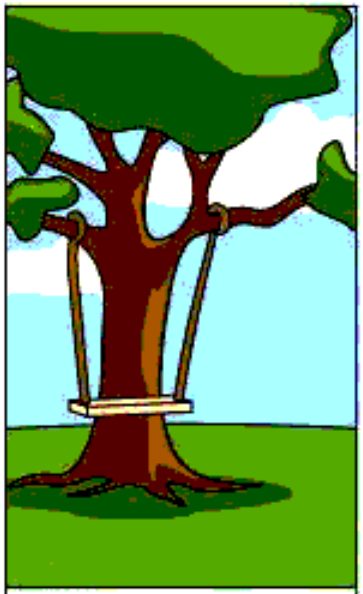
Clé d'inscription : M1_GL_2015



Partie I

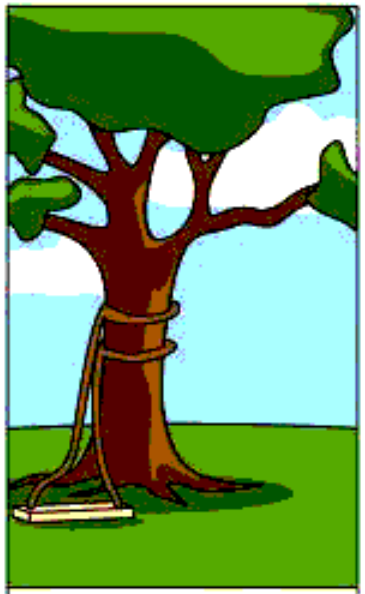
Introduction au Génie Logiciel

How the customer explained it



How the Project Leader understood it

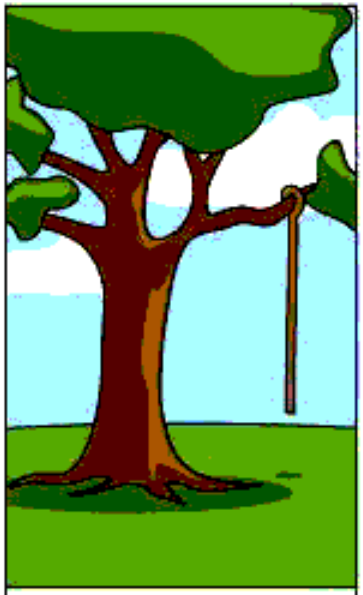
How the Analyst designed it



How the Programmer wrote it

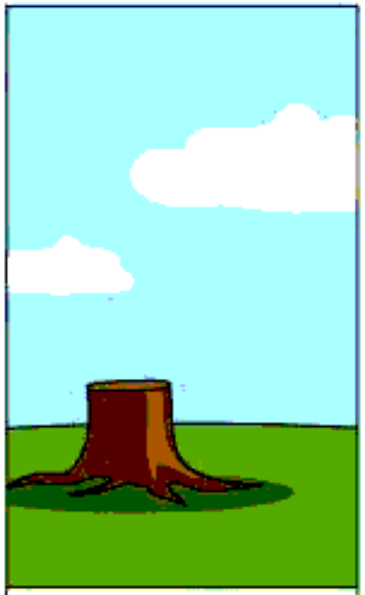
How the Business Consultant described it

How the project was documented

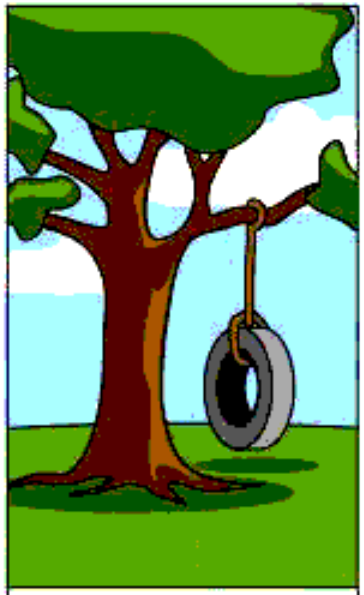


What operations installed

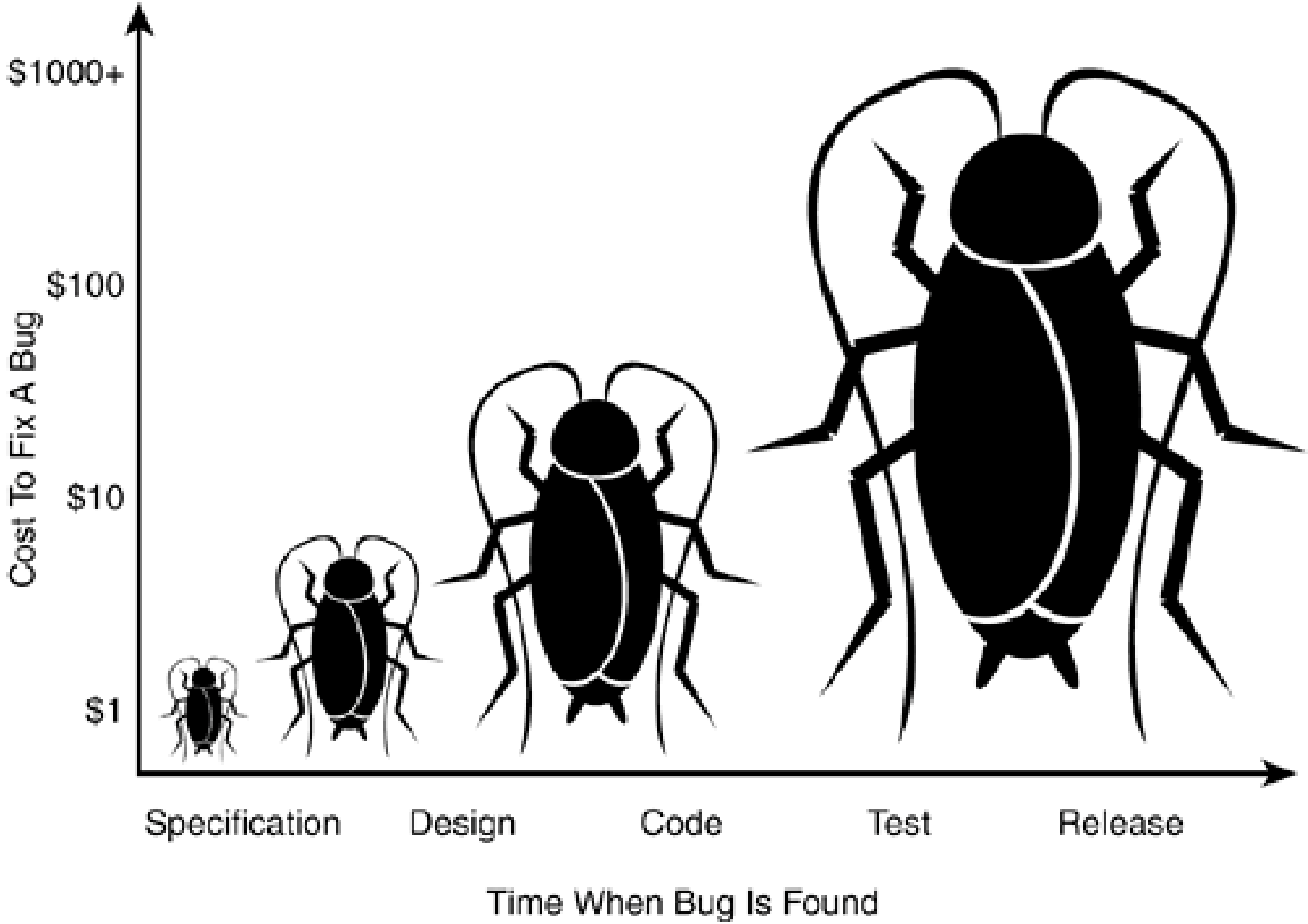
How the customer was billed



How it was supported



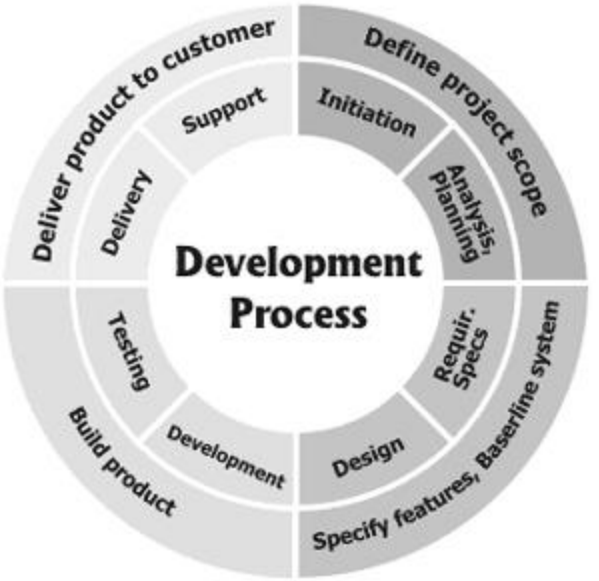
What the customer really needed


$1000+
$100
$10
$1
Cost To Fix A Bug
Specification
Design
Code
Test
Release
Time When Bug Is Found

Plan du cours Génie Logiciel – Partie I
Introduction au Génie Logiciel

1. Qu'est-ce que le Génie Logiciel
2. Critères de qualité du logiciel
3. Processus de développement
4. Processus Agile - SCRUM
5. Outillage du Génie Logiciel

- Ouvrage de référence : I. Sommerville, « Software Engineering », 9th edition, Addison-Wesley, 2010



Section 1

Qu’est-ce que le Génie Logiciel

L'impact des problèmes de qualité logicielle…

LE FIGARO.fr

12 décembre 2010

"La Banque Postale n'a pas payé 700 000 personnes retraitées dans la région PACA ce mois !"

"…les Tests fonctionnels réalisés sur les nouveaux formats de paiements sont concernés…"

L'impact des problèmes de qualité logicielle…

Richmond Times-Dispatch

20 mai 2011

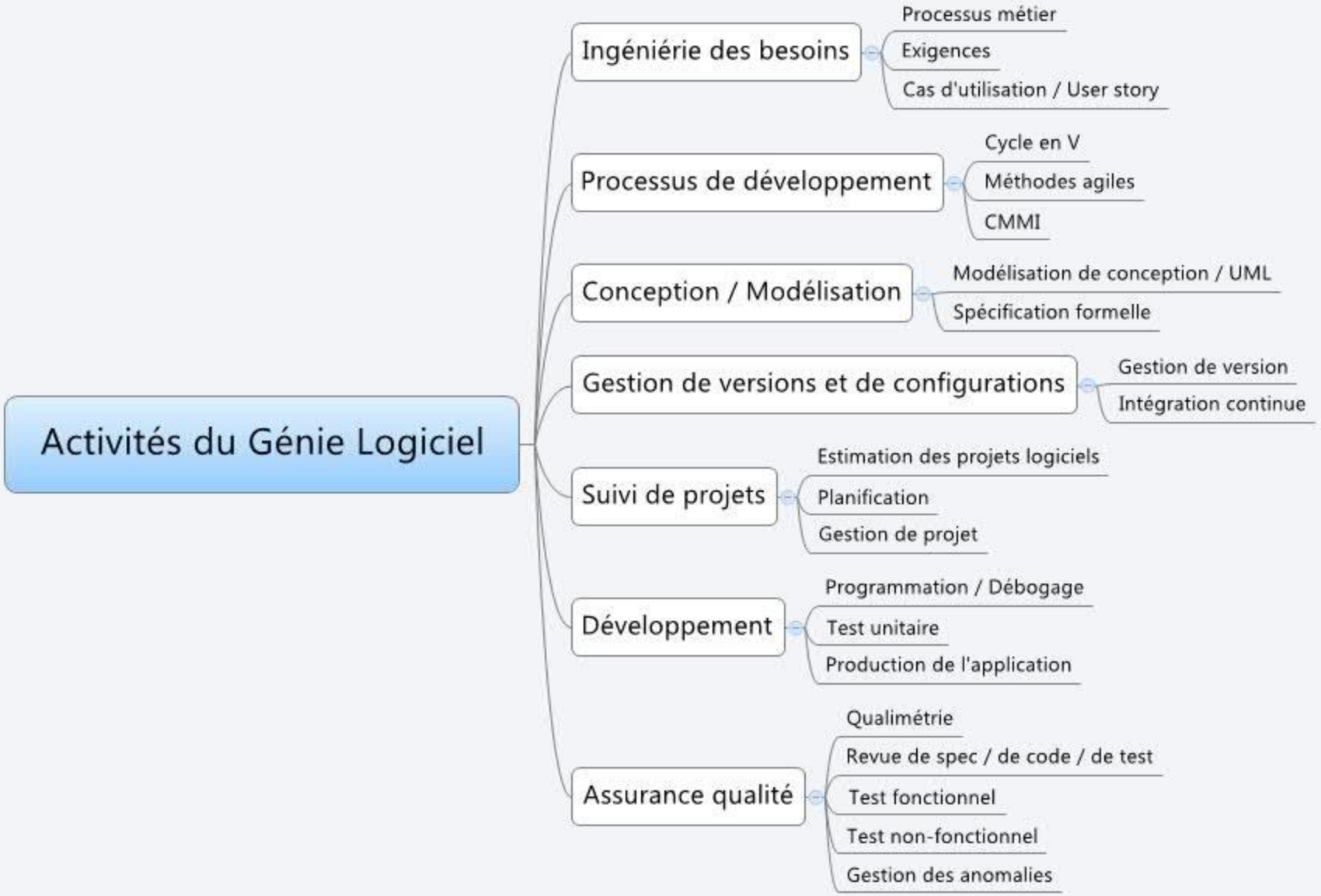
"Les pannes informatiques vont coûter $5 millions à La compagnie d'assurance Northrop Grumman."

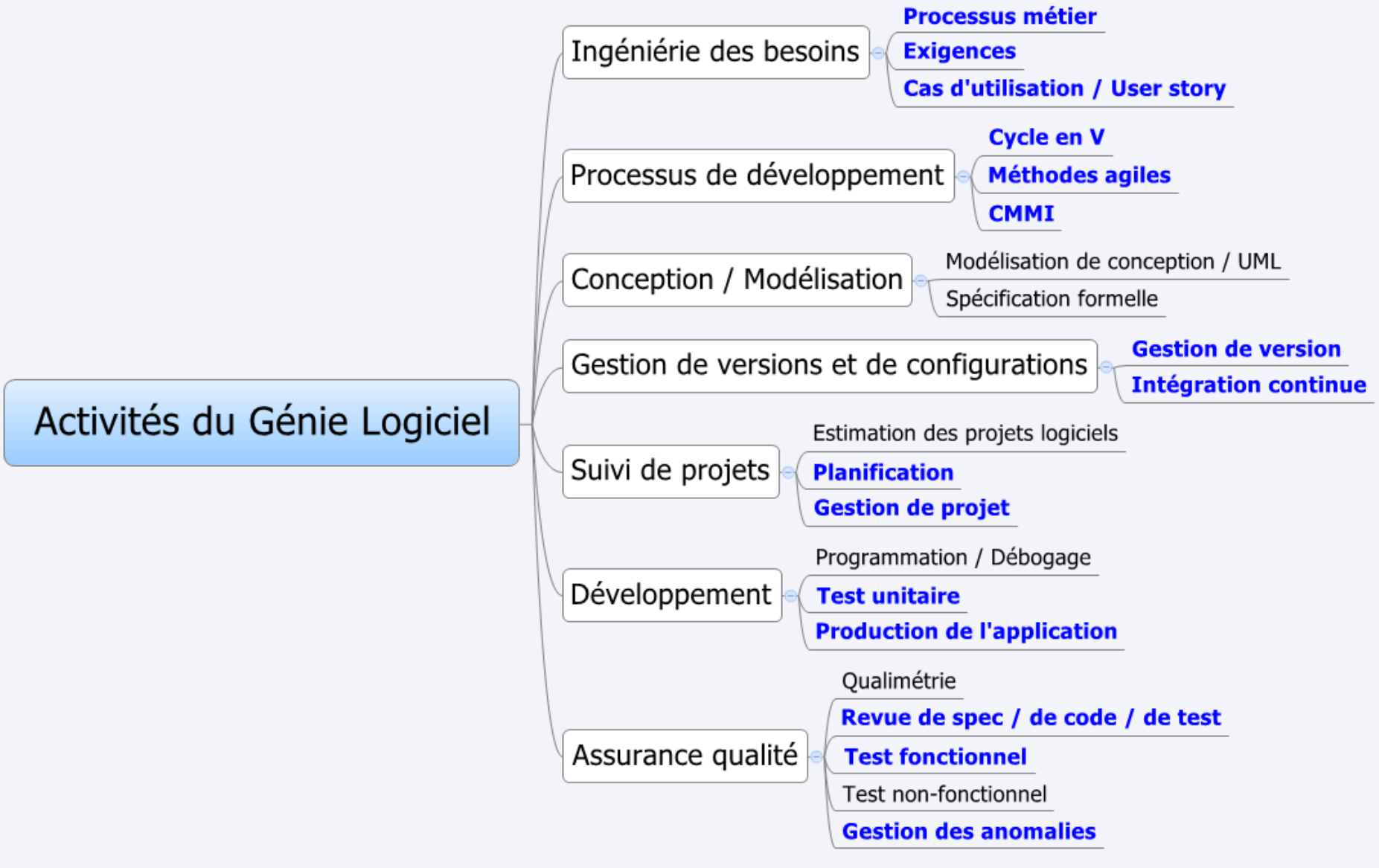
"…sous contrat avec le gouvernement pour la modernisation de son IT …les disfonctionnements ont affecté 26 des 89 branches…"

Le risque logiciel

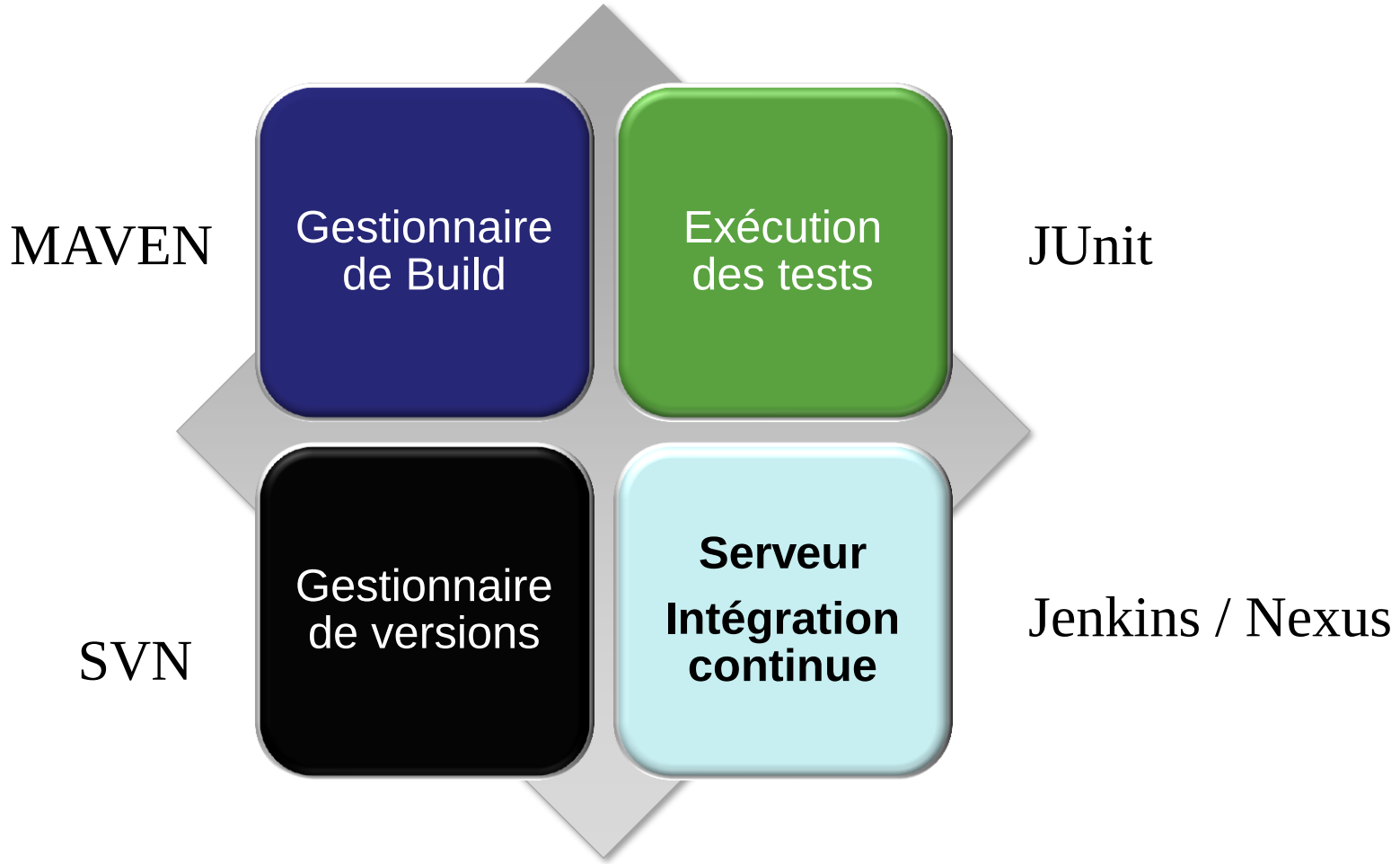
- Diffusion massive du logiciel (systèmes d'information, logiciels embarqués, pilotage, systèmes sécuritaire, …)
- L'impact sur la vie quotidienne et sur l'économie des entreprises des problèmes de non-qualité logicielle est de plus en plus grande

➔ **La qualité du logiciel est un enjeu majeur de l'ingéniérie du logiciel. C'est la raison d'être du Génie Logiciel !**


Activités du Génie Logiciel
Ingéniérie des besoins
Processus métier
Exigences
Cas d'utilisation / User story
Processus de développement
Cycle en V
Méthodes agiles
CMMI
Conception / Modélisation
Modélisation de conception / UML
Spécification formelle
Gestion de versions et de configurations
Gestion de version
Intégration continue
Suivi de projets
Estimation des projets logiciels
Planification
Gestion de projet
Développement
Programmation / Débogage
Test unitaire
Production de l'application
Assurance qualité
Qualimétrie
Revue de spec / de code / de test
Test fonctionnel
Test non-fonctionnel
Gestion des anomalies


Activités du Génie Logiciel
Ingéniérie des besoins
Processus métier
Exigences
Cas d'utilisation / User story
Processus de développement
Cycle en V
Méthodes agiles
CMMI
Conception / Modélisation
Modélisation de conception / UML
Spécification formelle
Gestion de versions et de configurations
Gestion de version
Intégration continue
Suivi de projets
Estimation des projets logiciels
Planification
Gestion de projet
Développement
Programmation / Débogage
Test unitaire
Production de l'application
Assurance qualité
Qualimétrie
Revue de spec / de code / de test
Test fonctionnel
Test non-fonctionnel
Gestion des anomalies

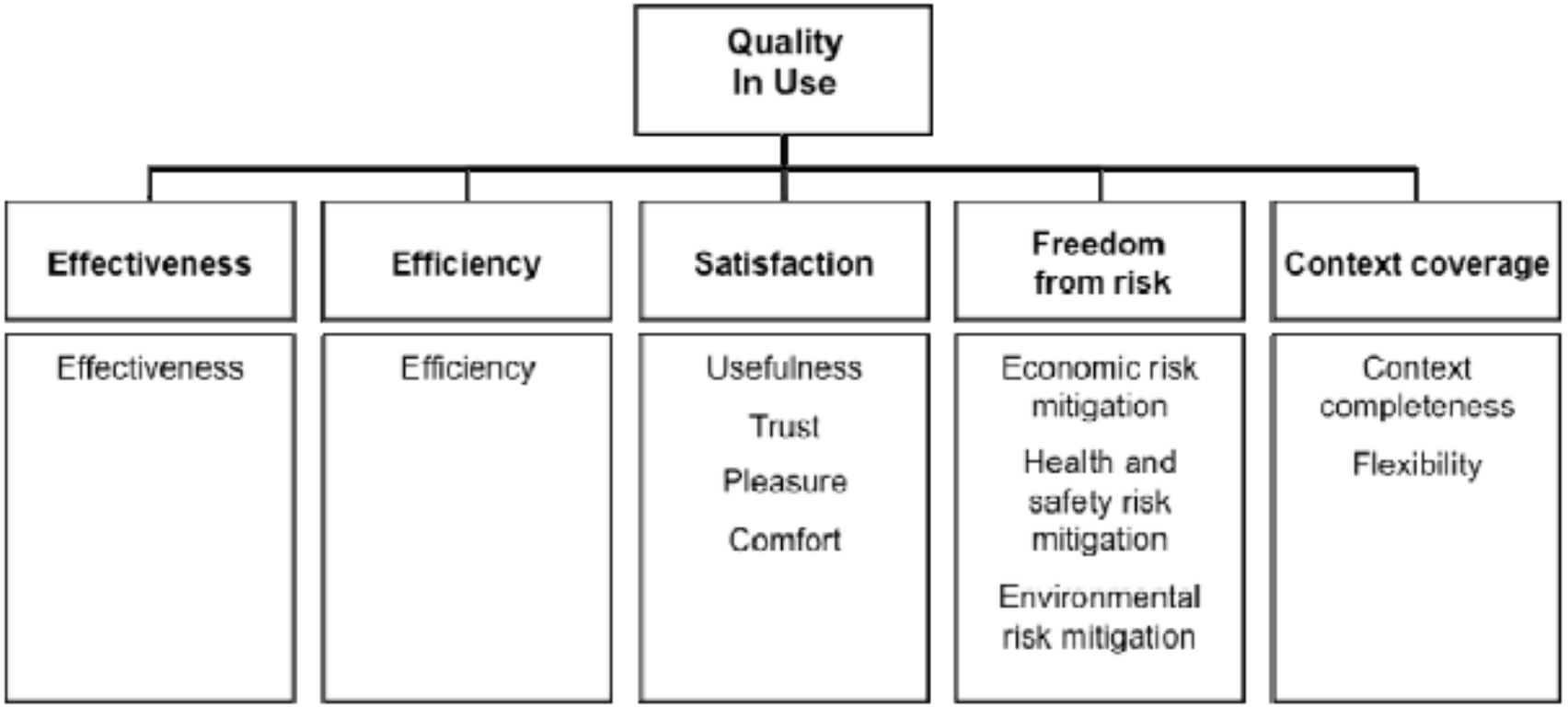
9h TP Génie Logiciel - Outillage



Section 2

Critères de qualité du logiciel (De quoi parle t'on ?)

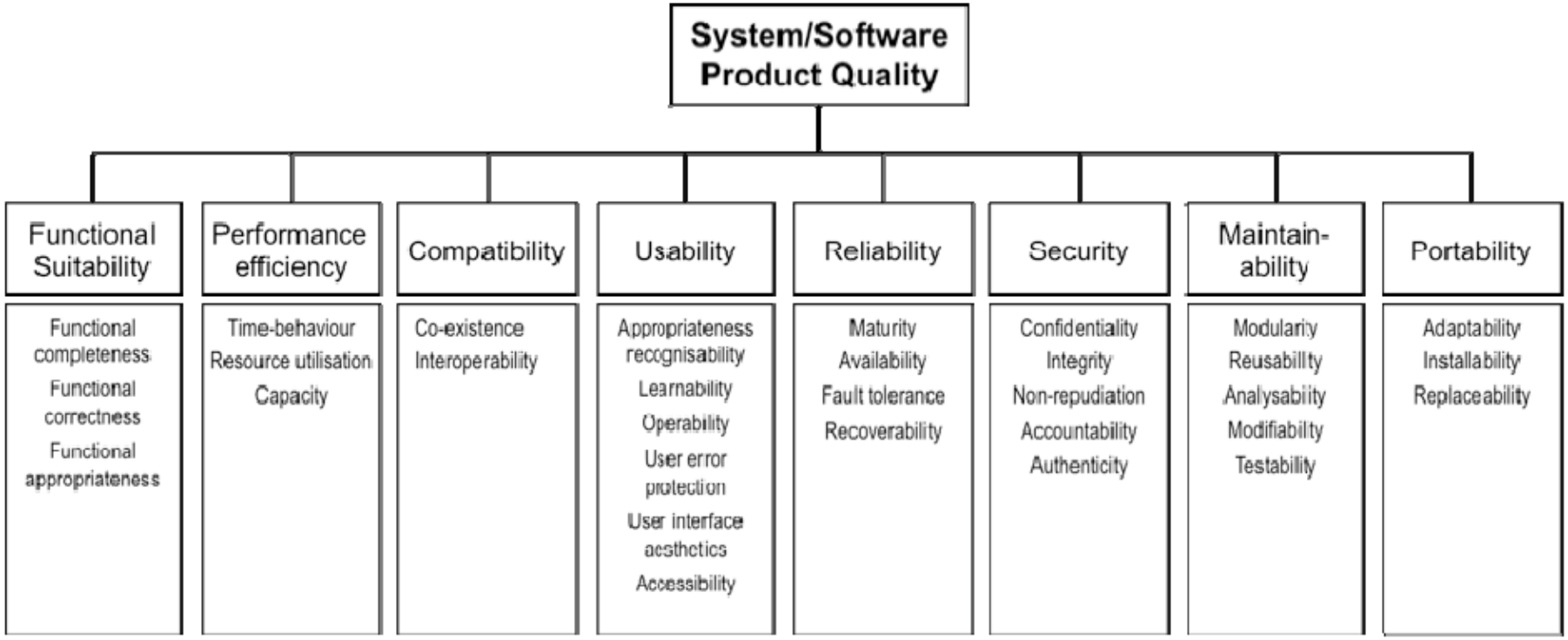
Standard ISO 25010:2011 - Qualité ressentie



Efficacité = Obtenir un Resultat conforme aux objectifs

Efficience = Bien utiliser les ressources pour arriver au resultat

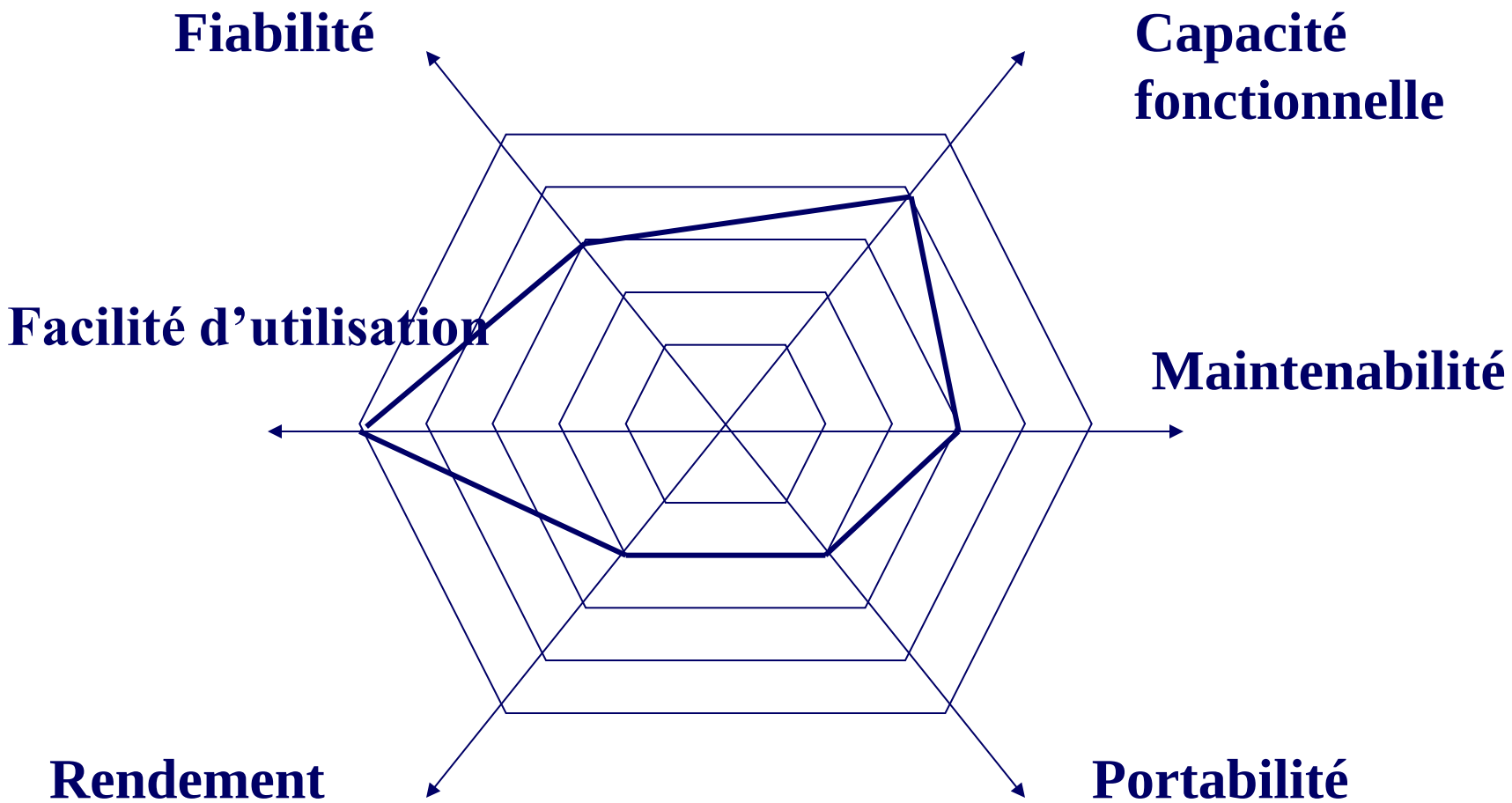
Standard ISO 25010:2011 – Caractéristiques de la qualité du logiciel



Le compromis de la qualité

- Les critères de qualité peuvent être contradictoires, pour chaque projet, le choix des compromis doit s'effectuer en fonction du contexte.
  - facilité d'emploi vs intégrité
  - efficacité vs extensibilité
- La qualité s’obtient pas la mise en place de procédure et méthode dés la phase de spécification
- La démarche qualité ne consiste pas à corriger les défauts mais à les prévenir

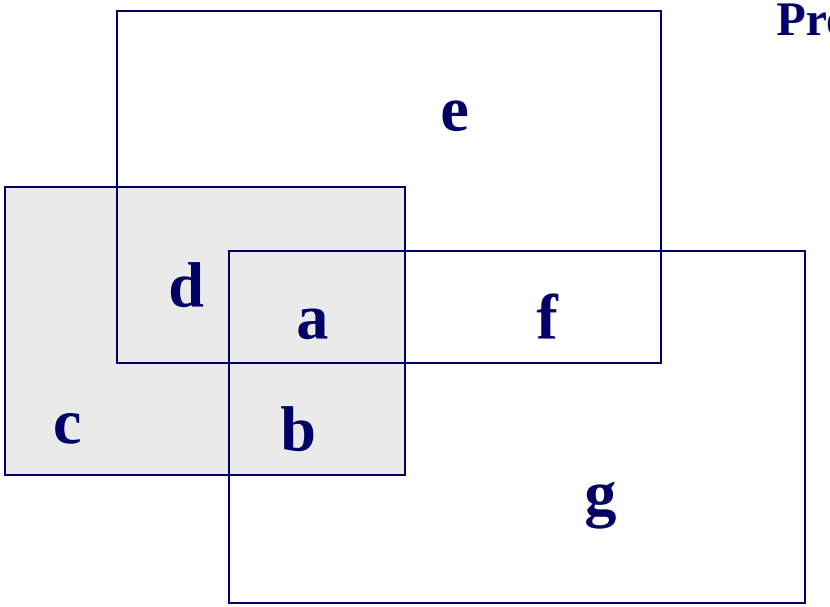
Caractéristiques de qualité



L’enjeu de la réponse aux besoins du client

Produit réalisé

Besoins exprimés



a : satisfaction
b : non conformité
c : insatisfaction client
d : satisfaction hasardeuse
e : prestations inutiles hors spécification
f : spécifications et prestations inutiles
g : spécifications inutiles

Spécifications rédigées

Facteurs de non qualité du logiciel

- Mauvaises spécifications
  - Vagues, incomplètes, instables
- Mauvaises estimations
  - Fausses, oublis, précisions insuffisantes
- Mauvaise répartition des tâches
  - Organisation inadaptée, contraintes omises
- Mauvais suivi
  - Ecarts non détectés à temps
- Mauvaise réalisation technique
  - Codage, tests, documentation
- Problèmes humains
  - Mauvaise distribution des travaux
  - Conflits, rétention d'information
- Manque d'expérience du métier de Chef de projet

Assurer la qualité du logiciel

- Assurer que le niveau de qualité requis est atteint
- Définition des standards de qualité et les procédures permettant d'assurer et de vérifier le niveau requis
- Vise à développer une culture qualité dan les équipes de développement et de maintenance : "chacun est responsable"

La qualité du logiciel : a quel coût ?

Le triangle Coût / Délai / Périmètre

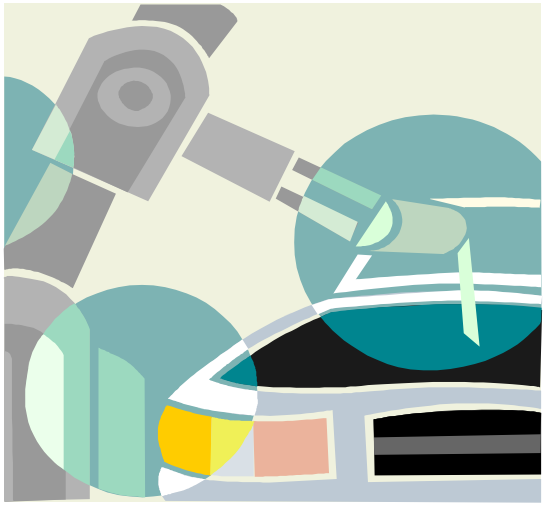
Tenir les engagements de qualité pour un certain coût et dans un certain délai !

Décaler la livraison pour tenir le périmètre

On augmente le coût !

Tenir les délais au détriment de la qualité

On dégrade la qualité, et on crée des coûts (potentiellement beaucoup plus grand) pour plus tard.



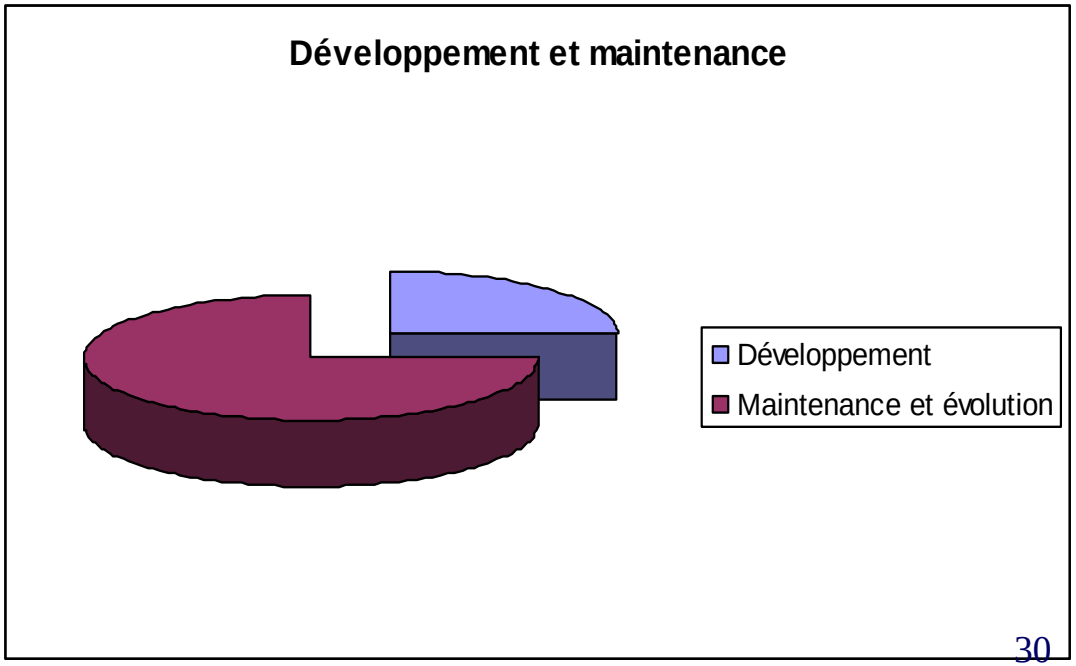
Section 3

Cycle de vie du logiciel

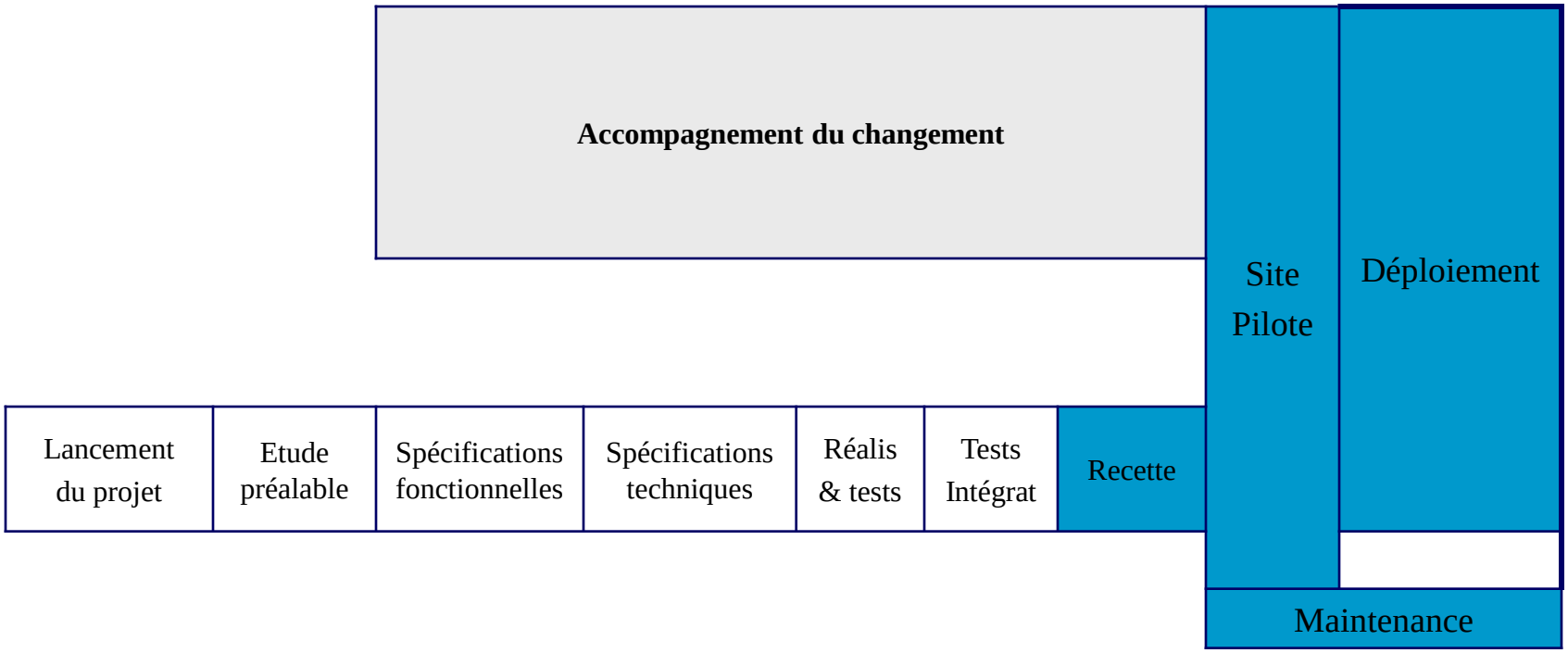
Cycle de vie du logiciel - Définition

- Cycle de vie du logiciel : suite de tâches pour spécifier, concevoir, implémenter, valider et maintenir les systèmes logiciels.

Pas seulement développer, mais aussi maintenir et faire évoluer



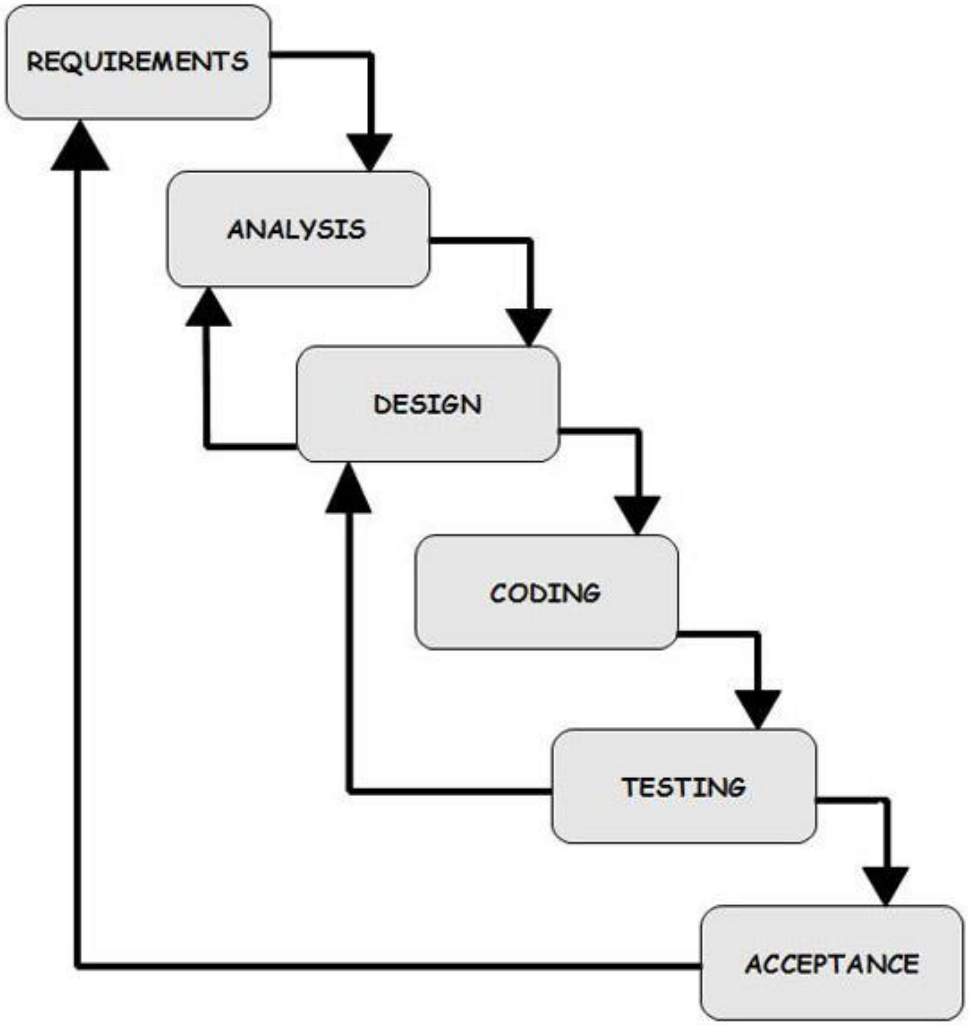
Vue globale des processus



Le processus de développement (cycle de vie)

- Une suite d'étapes pour le développement et la maintenance d'un système logiciel
  - Spécification
  - Conception
  - Codage
  - Validation
  - Evolution
- Un modèle de cycle de vie constitue une représentation abstraite du cycle. Le processus peut être vu suivant un angle particulier (ex. Qualité, Gestion des risques, …)

Processus linéaire : modèle en cascade

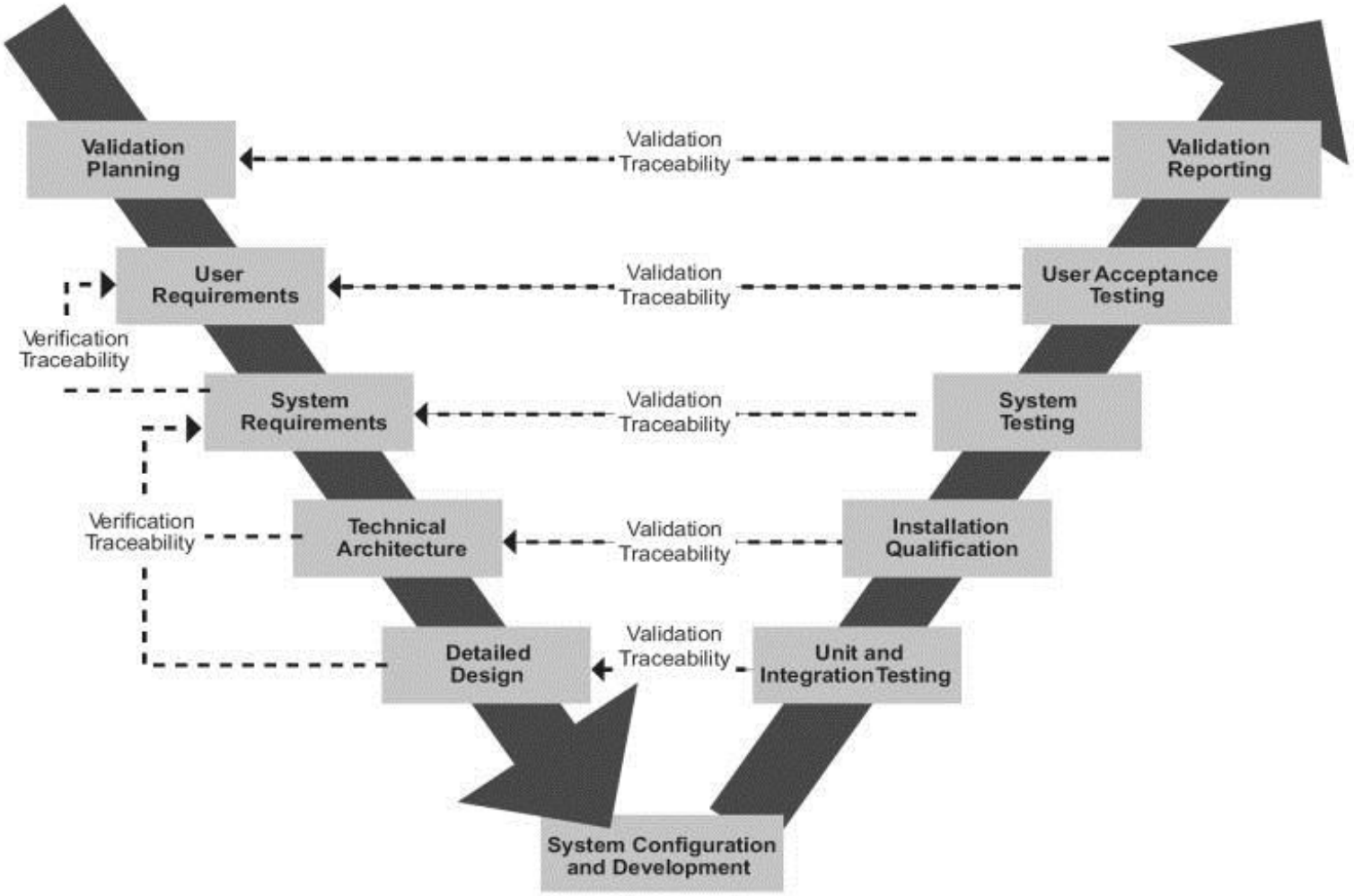


Les phases du modèle en cascade

- Expression de besoins et cahier des charges
- Spécification fonctionnelle détaillée
- Conception générale et détaillée
- Implémentation et test unitaire
- Intégration et test de conformité
- Mise en œuvre opérationnelle et maintenance

- Exemple : Merise (MCT, MCD, MOT, MLD, …)

Processus linéaire – Modèle en V



Les limites du modèle en cascade

- L'inconvénient principal des modèles séquentiels est « l'effet tunnel » vis-à-vis du client : après l'expression de besoins, suit un ensemble de phases (spécifications détaillées, conception et codage) qui conduit à ne pouvoir expérimenter qu'après une longue période :
  - les besoins peuvent avoir été imparfaitement analysés,
  - Les besoins peuvent évoluer parce que le contexte change.

- ☞ Le modèle en cascade n'est approprié que lorsque l'expression de besoins est parfaitement claire et non soumise à évolution.

Introduction au Génie Logiciel
Problèmes récurrents (process)

- **Attentes et retards :**
  - Le temps pour démarrer, staffer, valider, décider, les retards dans les tests, dans le déploiement, dans l'analyse… pour au final retarder la livraison finale.

- **Transmission d'informations :**
  - Le temps pour obtenir une réponse, l'accès au client, aux équipes… Il peut s'agir en premier lieu des déplacements de personnes, mais cela concerne aussi la perdition d'information qui guette à chaque transmission de documentation.

- **Processus inutiles / Excès de fonctionnalités :**
  - Chaque action engagée devrait être source de valeur pour le Client et doit être attendue par les équipes qui en sont les destinataires.
  - Le développement de fonctions non attendues, non utilisées est le plus gros gaspillage.

Introduction au Génie Logiciel
Problèmes récurrents (équipe)

- **Passages d'une tâche à une autre :**
  - Cumuler les tâches et les projets est coûteux d'un point de vue cognitif, outre la surcharge potentielle, c'est le temps qu'il faut pour se remettre en contexte qui constitue du temps gaspillé.

- **Travail partiellement fait**
  - Avec le travail partiellement fait, on ne sait jamais si ça marche ! C'est donc risqué et ça peut devenir obsolète. Pure perte et danger, sans compter le temps pour revenir dessus !

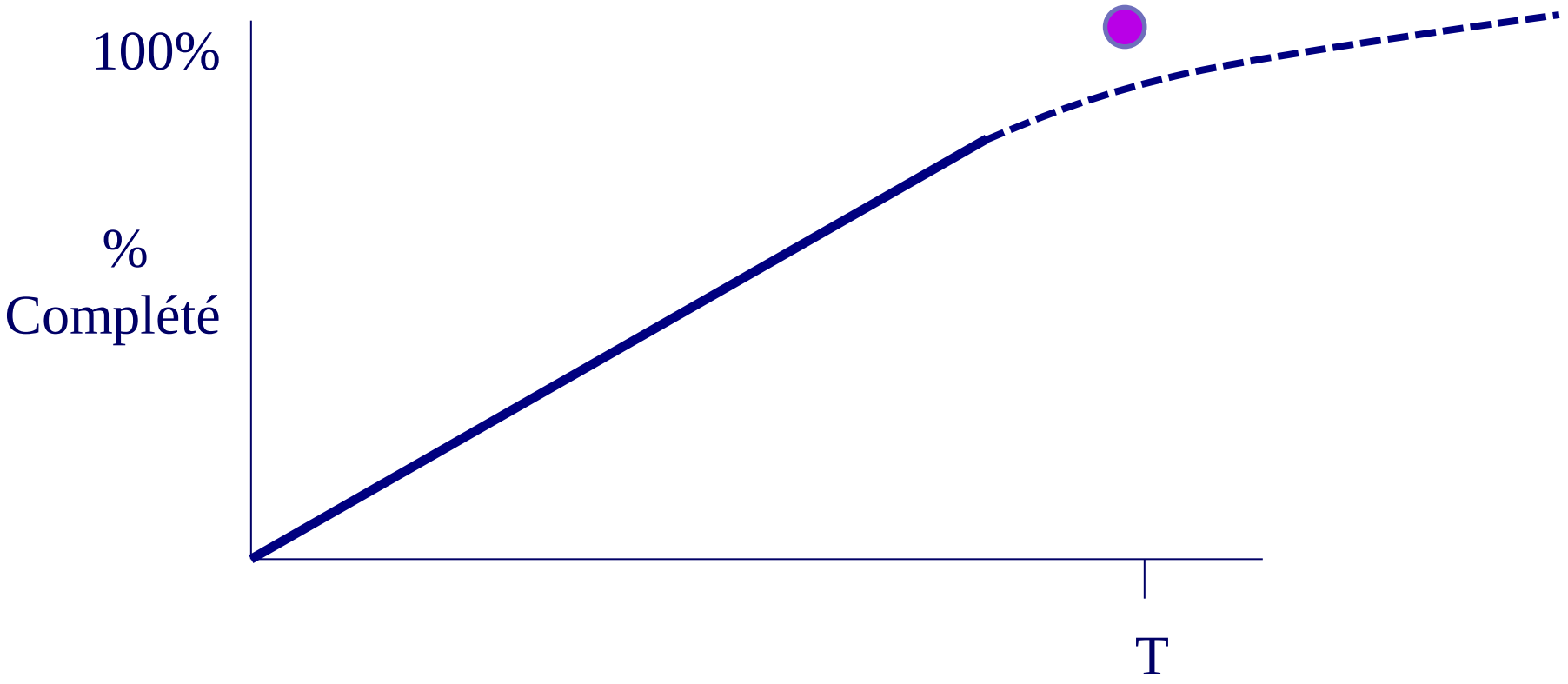
- **Défauts :**
  - Les défauts, anomalies n'ont pas de valeur. L'accumulation (files d'attentes ingérables et surchargées…) ou une détection trop tardive, font exploser les coûts et les délais, sans compter la satisfaction Client et l'image que vous renvoyez.

Introduction au Génie Logiciel
Cause des problèmes

- Les mécanismes du cycle en cascade sont fondés sur :
  - le suivi du projet,
  - le contrôle du projet,
  - pas sur le produit.
- Les méthodes qui en découlent sont appelées **méthodes prédictives** :
  - une hypothèse de base est posée (spécification),
  - cette hypothèse de base est maîtrisée
  - l'équipe projet s'engage à la réaliser.
- Il y a une tradition de diviser le rôle de chef de projet en deux :
  - chef de projet MOA (Maîtrise d'Ouvrage) : le client
  - chef de projet MOE (Maîtrise d'Œuvre) : le fournisseur
- Organisation souvent inefficace et conflictuelle :
  - l'information est divisée
  - les responsabilités (souvent partagées) sont rejetées vers l'autre groupe
  - la MOA en demande toujours plus
  - la MOE en propose toujours moins

Le syndrome des 90%

- Il est très difficile d'estimer la charge restante

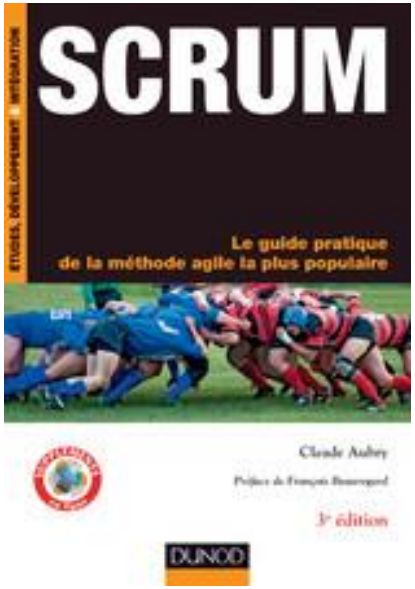


Introduction au Génie Logiciel
Quelle(s) solution(s) ?

- Utiliser des **méthodes empiriques** :
  - ce qui doit être fait n'est pas complètement maîtrisé
  - il faut avancer en acceptant et en considérant l'incertitude.

- Règles fondamentales de ce type de méthodes :
  - la transparence : diffuser en permanence les informations
  - l'inspection : maintenir des indicateurs et en tirer de manière périodique des enseignements
  - l'adaptation : ne pas hésiter à se remettre en cause

⇨ **Apparition des méthodes dites « agiles »**



Section 4

Processus de développement agile

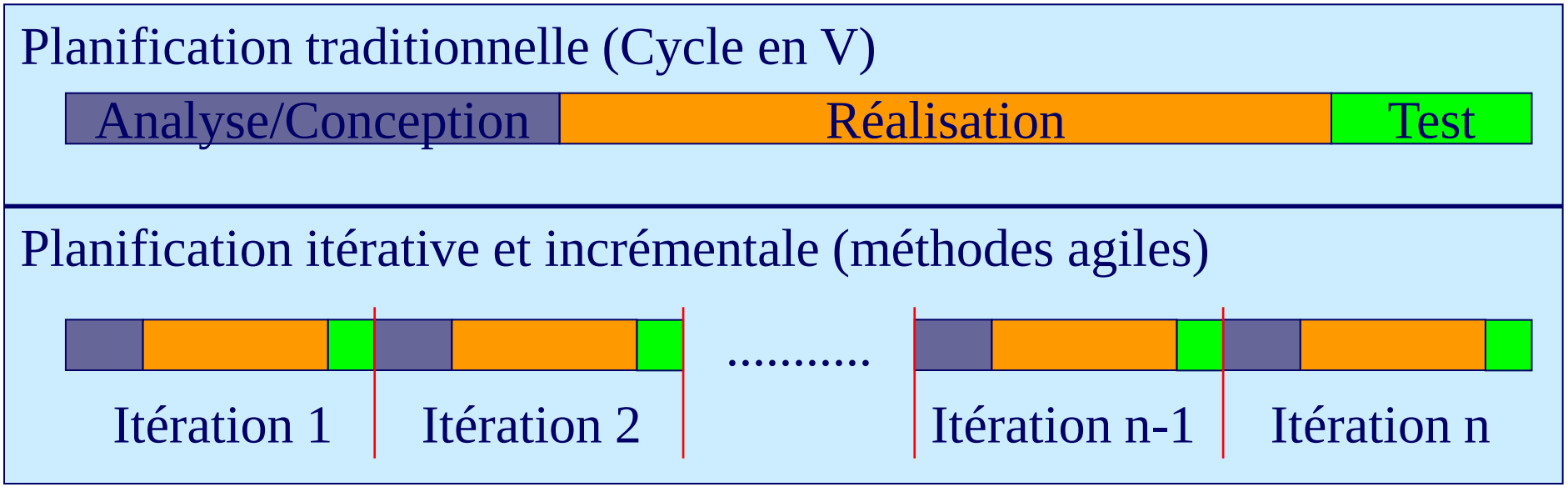
Agile Manifesto

Personnes et interactions	Plutôt que	Processus et outils
Un produit opérationnel	Plutôt que	Documentation exhaustive
Collaboration avec le client	Plutôt que	Négociation d'un contrat
Adaptation au changement	Plutôt que	Suivi d'un plan

Méthodes Agiles
12 principes élémentaires

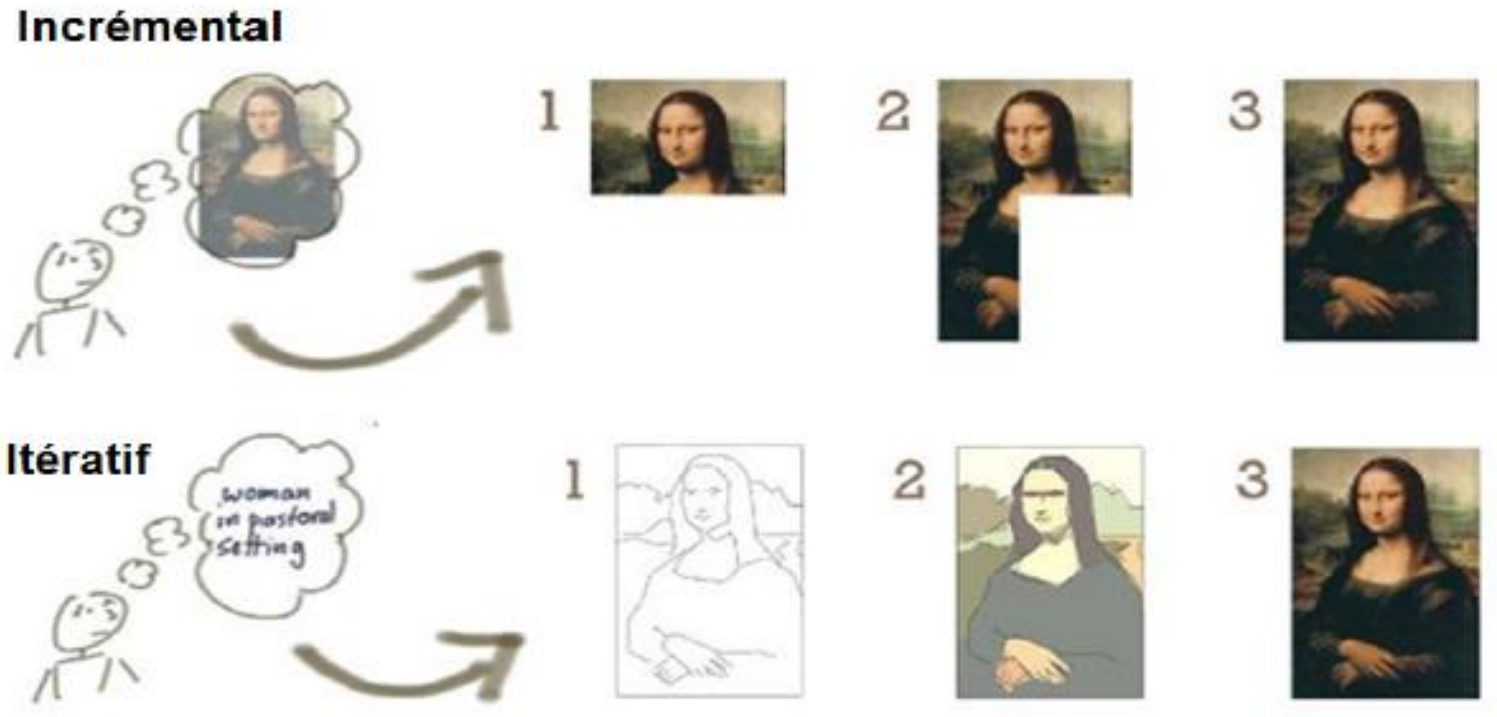
- Livrer rapidement et régulièrement des fonctionnalités à grande valeur ajoutée.
- Mesurer l'avancement d'un projet en fonction de ce qui est réellement opérationnel.
- Ne pas combattre le changement des besoins.
- Organiser le développement en cycles de quelques semaines ou mois en privilégiant les plus courts.
- Faire travailler les utilisateurs (ou leurs représentants) et les développeurs ensemble quotidiennement.
- Favoriser le dialogue en face à face.
- Ecouter et faire confiance à l'équipe de développement.
- Faire confiance à l'auto-organisation des équipes.
- Adopter un rythme de développement soutenable.
- Viser l'excellence technique et la bonne conception.
- Minimiser la quantité de travail inutile.
- Réfléchir et modifier son comportement pour devenir toujours plus efficace.

Méthodes Agiles
Cycle de développement



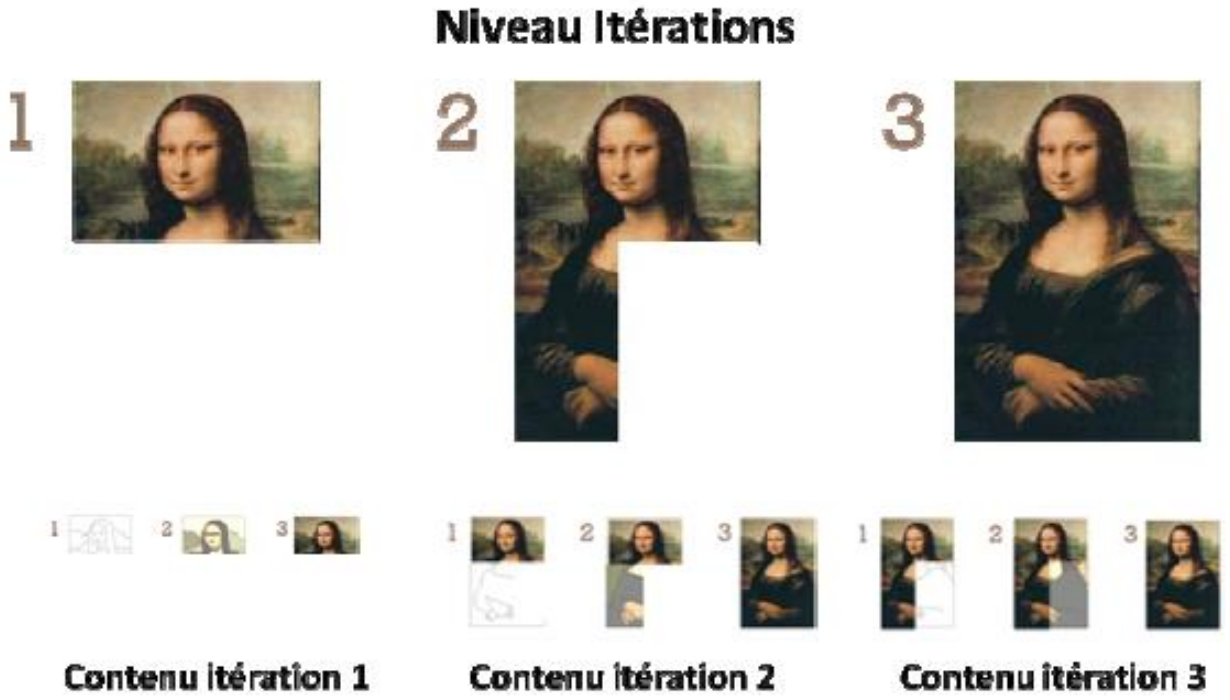
- Méthodes agiles : processus itératif et incrémental

Méthodes Agiles
Cycle incrémental vs itératif



(Jeff Patton)

Méthodes Agiles
Cycle incrémental et itératif



(Jean-Pierre Vickoff)

Une famille de méthode agile

- ‘Agile’ regroupe plusieurs approches :
  - **Extreme Programming (XP)**
  - **Scrum**
  - DSDM
  - Crystal
  - Kanban
  - …
- Fort développement depuis 2005

eXtreme Programming
Organisation de l'équipe

- **Le manager :**
  - Assure que l'équipe dispose des moyens nécessaires à son fonctionnement
  - Assume les responsabilités envers la hiérarchie
  - Fait l'interface avec le reste de l'organisation
- **Le client :**
  - Détermine les fonctionnalités du logiciel
  - Gère les priorités
  - Valide les développements réalisés d'un point de vue fonctionnel
- **Le coach :**
  - Valide, corrige et conseille la mise en œuvre de la méthode
  - Facilitateur mais pas un donneur d'ordres.
- **Les développeurs :**
  - Participent à l'estimation des coûts de développement
  - Rendent compte du point de vue technique
  - Assurent le développement du produit

eXtreme Programming
Les valeurs

- 4 valeurs à toujours respecter :
  - **Communication :**
    Au sein de l'équipe de développement comme avec le client
  - **Feedback :**
    Itérations rapides permettant une forte réactivité
  - **Simplicité :**
    code simple (KISS), livrables ne contenant que les exigences du client,…
  - **Courage :**
    Il faut parfois faire des choix difficiles (cela est facilité par les 3 premières valeurs)

eXtreme Programming
Pratiques de développement

- **Restructuration du code (refactoring)**
  - Investir pour le futur en maitrisant la dette technique
- **Conception simple**
  - Faciliter la reprise du code et l'ajout de fonctionnalités
- **Tests unitaires**
  - Détecter au plus tôt les erreurs et la régression
  - Test first (TDD) pour améliorer la testabilité et simplifier l'architecture
- **Intégration continue**
  - Accélérer la détection de dysfonctionnements tout en ajoutant continuellement de la valeur
- **Règles de codage**
  - Améliorer la lisibilité et la reprise du code (ex: java → Standards SUN,…)

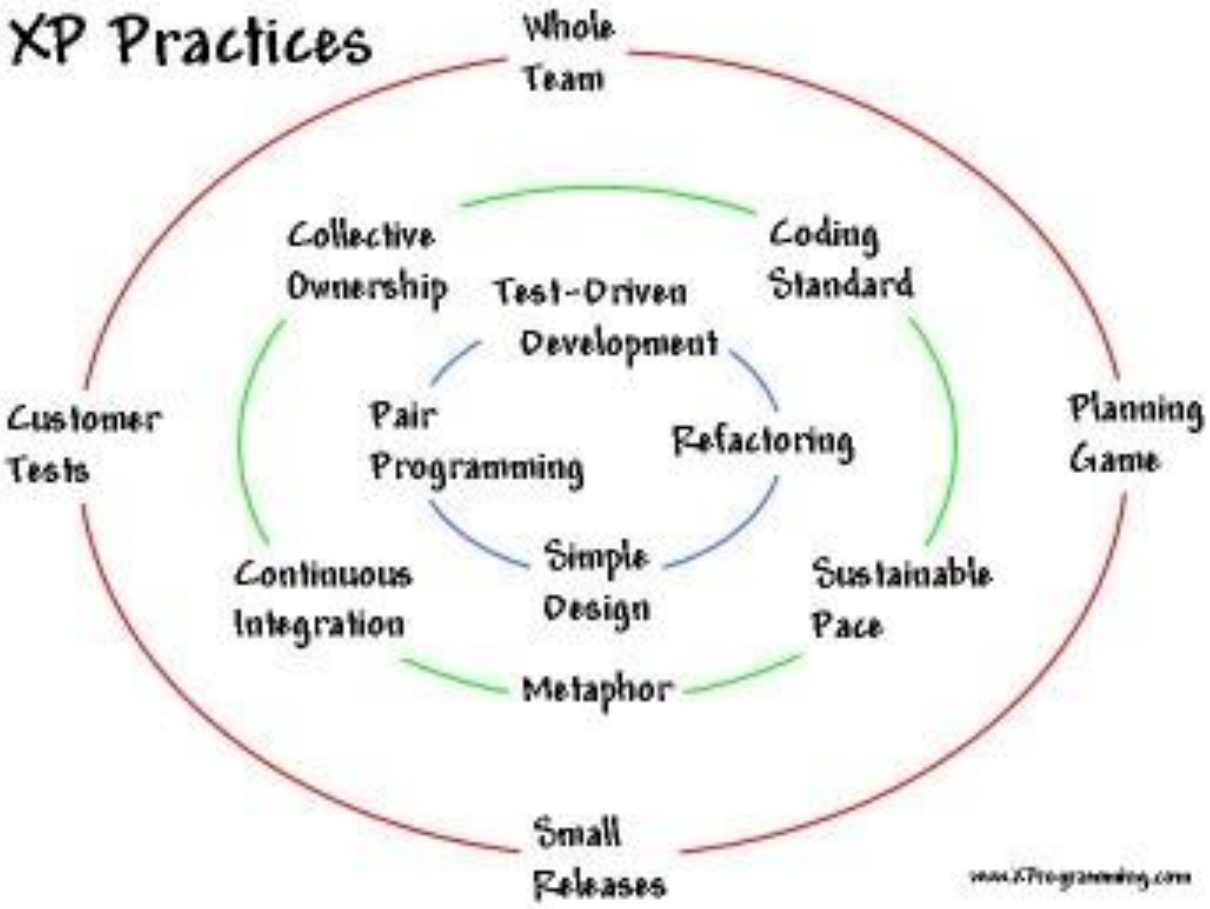
eXtreme Programming
Pratiques d'équipe

- **Responsabilité collective du code**
  - Rendre les développeurs plus polyvalents
- **Travail en binôme**
  - Assembler/partager les compétences
  - Prévenir les erreurs
  - Créer une motivation mutuelle
- **Langage commun (métaphore)**
  - Faciliter la compréhension et l'adhésion au groupe
- **Rythme régulier**
  - Atténuer les effets « rush » aux effets incontrôlables (dans le temps et dans les résultats espérés)

eXtreme Programming
Pratiques de gestion de projet

- **Client sur site**
  - Accélérer les prises de décisions (et les bonnes !)
- **Tests d'acceptation**
  - Garantir la conformité par rapport aux attentes du client
- **Livraisons fréquentes**
  - Planification itérative et incrémentale des tâches
  - Démontrer la valeur ajoutée de façon continue
  - Organisation par itérations de développement

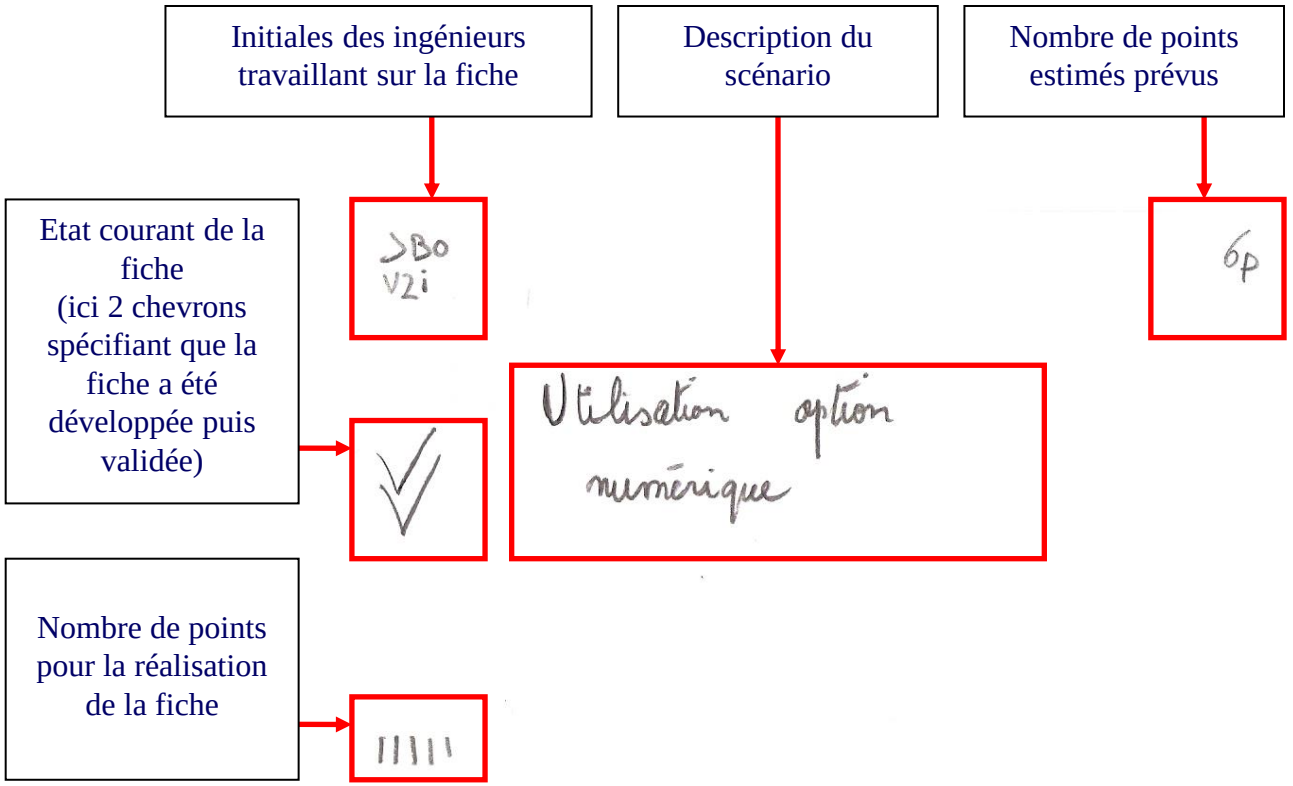
XP
Synthèse des pratiques



eXtreme Programming
Définition des tâches

- Pour chaque cas d'utilisation (fonctionnalité) :
  - Définition d'une fiche qui correspond à une tâche de développement
- Sur chaque fiche, on renseigne :
  - La description du scénario correspondant
  - Un ordre de priorité
  - L'état courant de la fiche (en développement, développé, validé,…)
  - La durée prévue pour la réalisation (en jours ou en nombre de points : un point = une unité de temps ou un niveau de difficulté)
  - La durée écoulée depuis le début de la réalisation de la fiche
  - Le(s) développeur(s) qui réalise(nt) le développement
- Les fiches sont rédigées pour la plupart au début du projet, puis tout au long du projet en fonction des changements

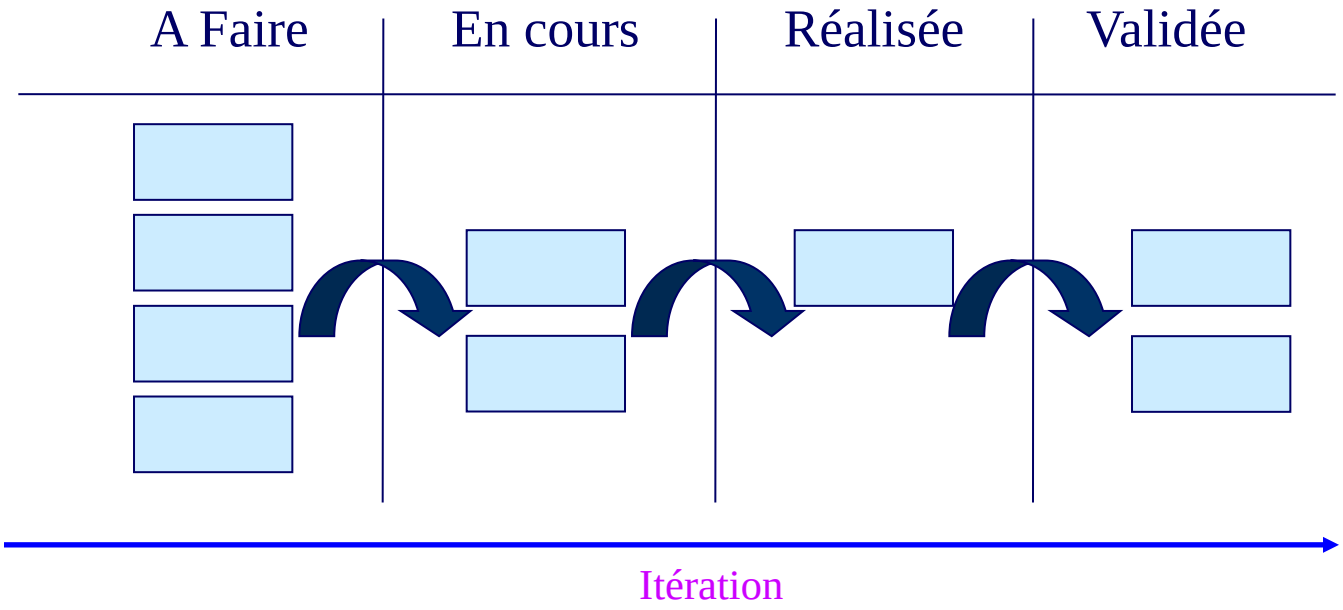
eXtreme Programming
Exemple de fiche



eXtreme Programming
Cycle de vie des fiches

- En fin d'itération, on connaît le nombre de points réalisés, on peut donc calculer la vélocité de l'itération courante et estimer celle de la prochaine itération, et par récurrence prévoir les délais de réalisation du projet global.

- Le rapport vélocité prévue / vélocité effective permet d'obtenir une métrique quant au rendement de l'équipe et prévoir d'éventuelles adaptations pour l'améliorer.

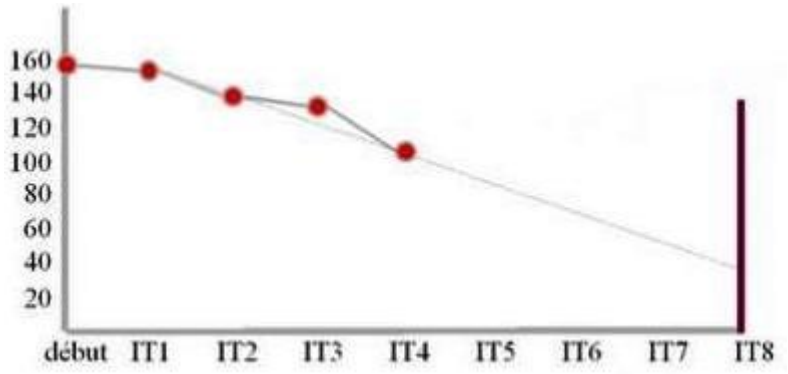


eXtreme Programming
Suivi de l'avancement du projet

- Montrer l'avancement réel (tendance), en analysant ce qui est complètement fini et ce qu'il reste à faire.
- Permettre la remise en question sur la façon de continuer et de rendre compte :
  - Estimation de la date de fin.
  - Réajustement des objectifs fonctionnels en cas de dépassement.

- A noter que cela nécessite d'identifier et d'estimer tous les développements prévus, ce qui prend du temps et n'est pas toujours très pertinent en début de projet.

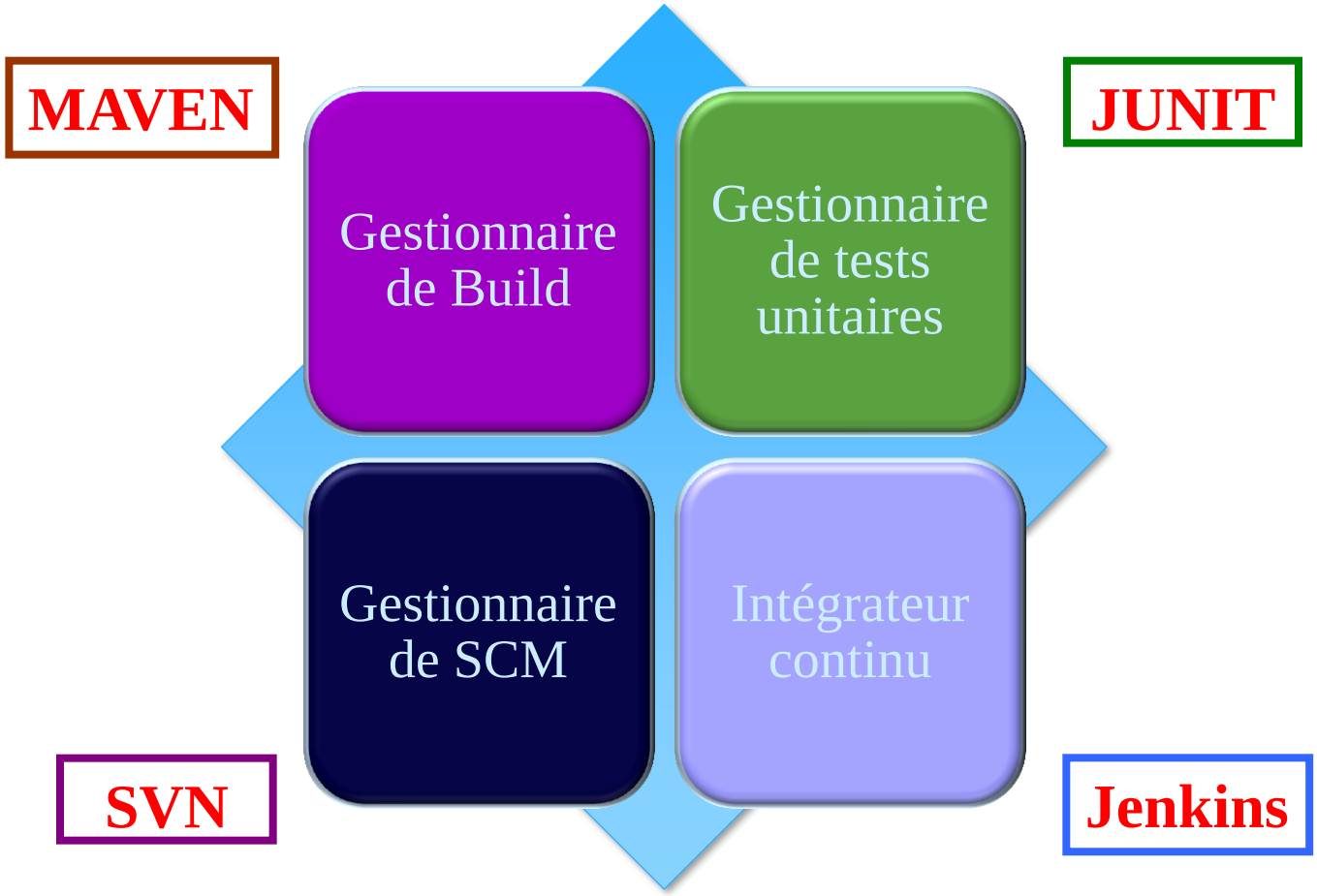
Diagramme de type "burndown"



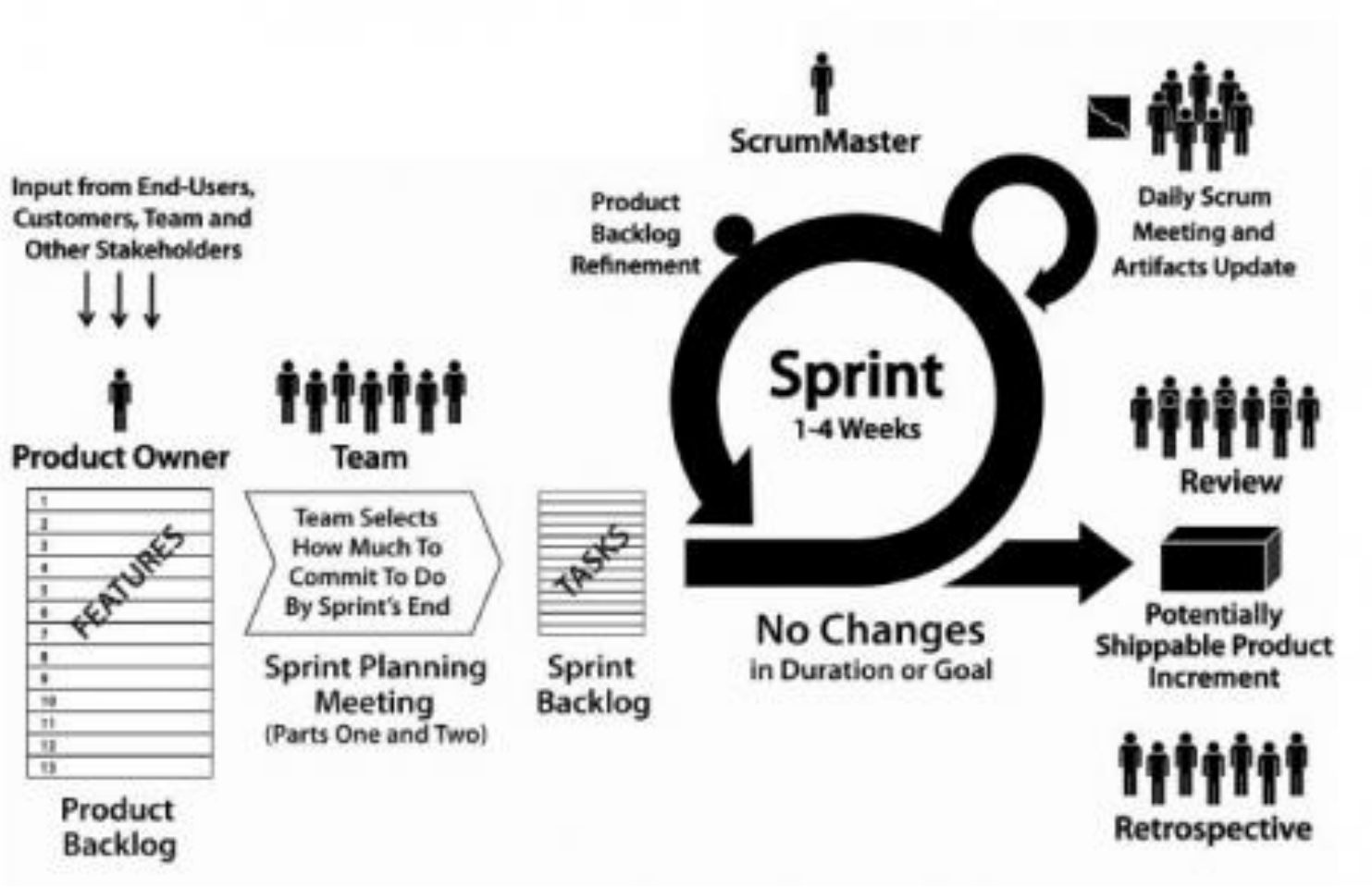
eXtreme Programming
Cycle agile standard

- Le client écrit ses besoins sous forme de scénarii (user story).
- Les développeurs évaluent le coût de chaque scénario (planning game).
- Le client choisit les scénarii à intégrer à la prochaine livraison (en accord avec les développeurs) et propose les tests d'acceptation (fonctionnels).
- Chaque binôme de développeurs prend la responsabilité d'une tâche.
- Le binôme écrit les tests unitaires (structurels) correspondant au scénario à implémenter puis procède à l'implémentation proprement dite. (Test Driven Development)
- Le binôme restructure le code (refactoring).
- Le binôme rend compte au client qui valide le développement.

Outils de l'agilité



Scrum


ScrumMaster
Input from End-Users, Customers, Team and Other Stakeholders
Product Backlog Refinement
Daily Scrum Meeting and Artifacts Update
Sprint
1-4 Weeks
Product Owner
Team
Review
FEATURES
Team Selects How Much To Commit To Do By Sprint's End
TASKS
No Changes
in Duration or Goal
Potentially Shippable Product Increment
Sprint Planning Meeting
(Parts One and Two)
Sprint Backlog
Product Backlog
Retrospective

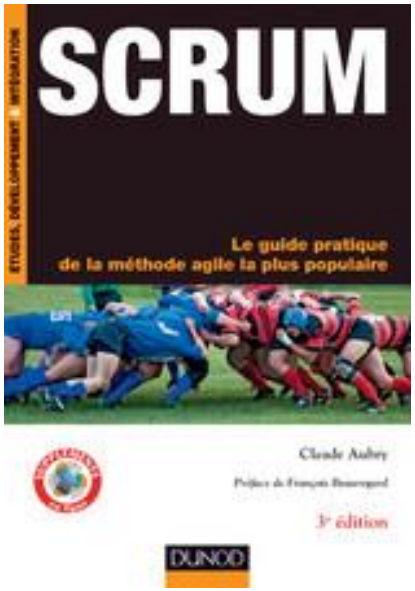
Différences XP / Scrum

- Les itérations Scrum plus longues que celles de XP – 3-4 semaines vs 1-2 semaines)
- Scrum n'autorise pas les changements pendant les itérations, et XP est plus malléables
- XP est plus prescriptif sur les pratiques de développement (cf précédent). XP plus « puriste ».
- Scrum plus orienté méthode de gestion de projets, et XP plus orienté méthode de développement.

Mais ce sont des méthodes très proches.

Méthodes agiles
Pour en savoir plus…

- Méthodes agiles :
  - http://www.agilealliance.org/
- XP :
  - http://www.extremeprogramming.org/
  - http://www.xp-france.net/
- SCRUM :
  - http://www.controlchaos.com/
  - http://www.scrumalliance.org/
- Pour la détente :
  - http://www.risacher.com/la-rache/



Section 5

SCRUM

Synthèse des pratiques agiles

PROJET	CODE	EQUIPE
Livraisons fréquentes : l'équipe vise la mise en production rapide d'une version minimale du logiciel, puis elle fournit ensuite régulièrement de nouvelles livraisons en tenant compte des retours du client.	**Conception simple** : on ne développe rien qui ne soit utile tout de suite.	**Programmation en binômes** : les développeurs travaillent en binômes, ces binômes étant renouvelés fréquemment.
Planification itérative : un plan de développement est préparé au début du projet, puis il est revu et remanié tout au long du développement pour tenir compte de l'expérience acquise par le client et l'équipe de développement.	**Tests unitaires** : les développeurs mettent en place une batterie de tests de non régression qui leur permettent de faire des modifications sans crainte.	**Responsabilité collective du code** : chaque développeur est susceptible de travailler sur n'importe quelle partie de l'application.
Client sur site : le client est intégré à l'équipe de développement pour répondre aux questions des développeurs et définir les tests fonctionnels.	**Restructuration (refactoring)** : le code est en permanence réorganisé pour rester aussi clair et simple que possible.	**Rythme régulier** : l'équipe adopte un rythme de travail qui lui permet de fournir un travail de qualité constant tout au long du projet.
Tests de recette : les testeurs mettent en place des tests automatiques qui vérifient que le logiciel répond aux exigences du client. Ces tests permettent une validation automatique du logiciel.	**Intégration continue** : l'intégration des nouveaux développements est faite chaque jour.	**Métaphore** : les développeurs s'appuient sur une description commune du design.
	Règles de codage : les développeurs se plient à des règles de codage strictes définies par l'équipe elle-même.	

Méthodes agiles – M1/GL
Objectifs d'apprentissage

- Objectif 1 – Je connais, je comprends et je sais mettre en oeuvre les principales pratiques clés des méthodes agiles de développement logiciel (type SCRUM)
- Objectif 2 – Je suis capable d'adopter une posture constructive de coopération et de dialogue qui favorise l'efficacité collective de l'équipe agile
- Objectif 3 – Je sais mettre en oeuvre des tests unitaires automatisés et analyser la couverture de code
- Objectif 4 – Je suis capable d'automatiser la construction de l'application et de mettre en place l'intégration continue

Méthodes agiles – M1/GL
Activités d'apprentissage

- TD – Planning Game sur le projet CUITEUR – 3h + 1h
  - ➔ O1 & O2
- TP outillage du GL – 9h + 9h
  - ➔ O3 & O4
- ➔ Projet de compilation en équipe SCRUM – 7h30 + 20h
  - ➔ O1, O2, O3 & O4
- Auto-apprentissage (Forum, Vidéos de conférences sur l'agilité) - TP outillage du GL – 10h
  - ➔ O1

Méthodes agiles – M1/GL
Evaluation des acquis

- Evaluation GL du projet de compilation (20%) :
  - Mise en oeuvre de SCRUM ➔ O1 & O2
    - » Combien de pratiques mises en oeuvre
    - » Comment les pratiques ont été mises en oeuvre
  - Mise en oeuvre des outils du GL ➔ O3 & O4
    - » Quantité de tests unitaires et couverture du code
    - » Mise en oeuvre du build automatique et de l'intégration continue
- Contrôle des connaissances #1 (30%)
  - Cible : O1, O2, O3 et O4

Scrum on a Page

Roles

Product Owner
Set Priorities
Manage Product Backlog

Scrum Master
Teach Scrum
Manage Process
Protect Team
Enforce Rules
Remove Blocks

Team
Develop Product
Organize Work
Report Progress

Stakeholders
Observe & advise

Artifacts

Product Backlog
List of requirements
Owned by product owner
Anybody can add to it
Prioritized by business value
Can change without affecting the active sprint

Sprint Goal
One sentence summary
Defined by Product Owner
Accepted by Team

Sprint Backlog
Decomposed task list
Driven by a portion of Product Backlog
Owned by Team
Only Team modifies it

Blocks List
List of blocks & pending decisions
Owned by Scrum Master
Blocks stay on list until resolved

Increment
Version of the product
Potentially shippable
Working functionality
Tested & documented according to project definition of "DONE"

Meetings

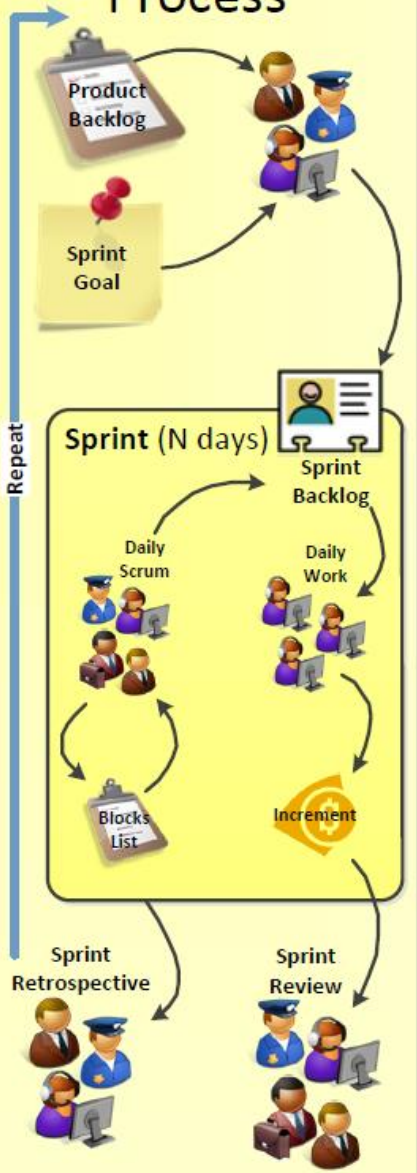
Sprint Planning
Part A
Time-boxed to 4 hours
Run by Scrum Master
Declare Sprint Goal
Top of Product Backlog presented by Product Owner to Team
Team asks questions & selects topmost features
Part B
Time-boxed to 4 hours
Run by Scrum Master
Team decomposes selected features into a Sprint Backlog
Team adjusts +/- features by estimates against sprint capacity

Daily Scrum
Time-boxed to 15 minutes
Run by Scrum Master
Attended by all
Stakeholders do not speak
Same time/place every day
Answer 3 questions:
1) What I did yesterday?
2) What I'll do today?
3) What's in my way?
Team updates the Sprint Backlog
Scrum Master updates the Blocks List

Sprint Review
Time-boxed to 2 or 4 hours
Run by Scrum Master
Attended by all
Informal, informational, Discussion
Team demonstrates increment
All discuss

Sprint Retrospective
Time-boxed to 1 or 2 hours
Run by Scrum Master
Attended by Team and Product Owner
Discuss process improvements, successes and failures
Adjust process

Process



www.andrewsiemer.com / (661) 600-2355 / asiemer@hotmail.com