



LOG3000

Processus du génie
logiciel

Plan de cours



Plan de cours

LOG3000 – Processus du génie logiciel

Département de génie informatique et génie logiciel

Hiver 2022

3 Crédits

3 / 1,5 / 4,5

<https://moodle.polymtl.ca/course/view.php?id=987>

Chargés de laboratoire

Hakim Mektoub (hakim.mektoub@gmail.com)
Souleima Zghab (souleima.zghab@polymtl.ca)

Chargés de cours

Nom	Mouna Abidi
Bureau	
Courriel	mouna.abidi@polymtl.ca
Disponibilité	Sur rendez-vous

Enseignant

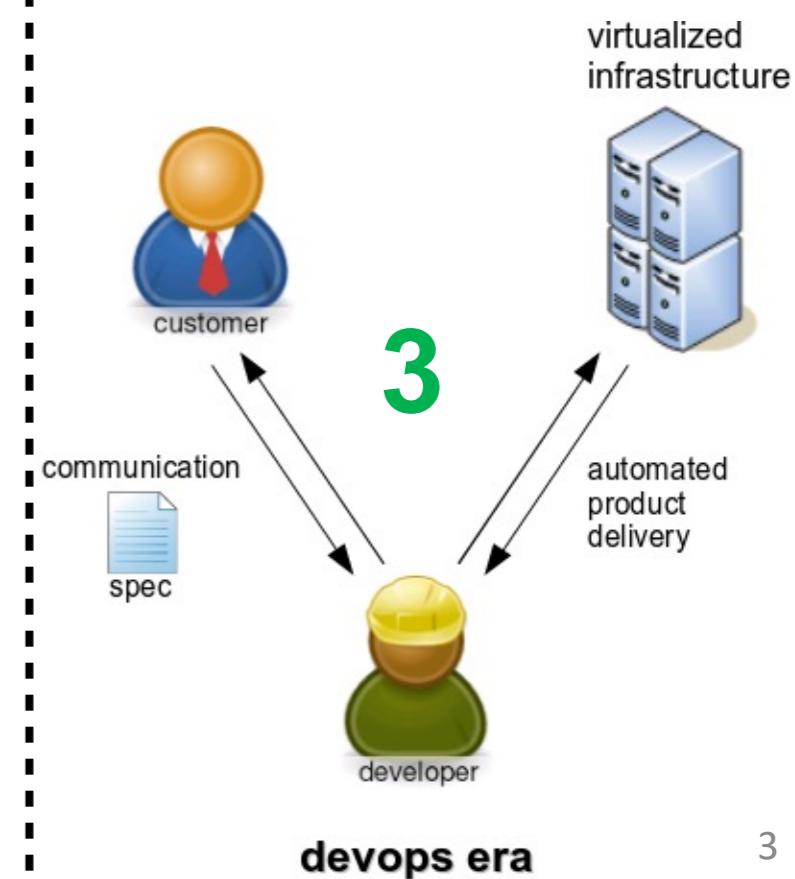
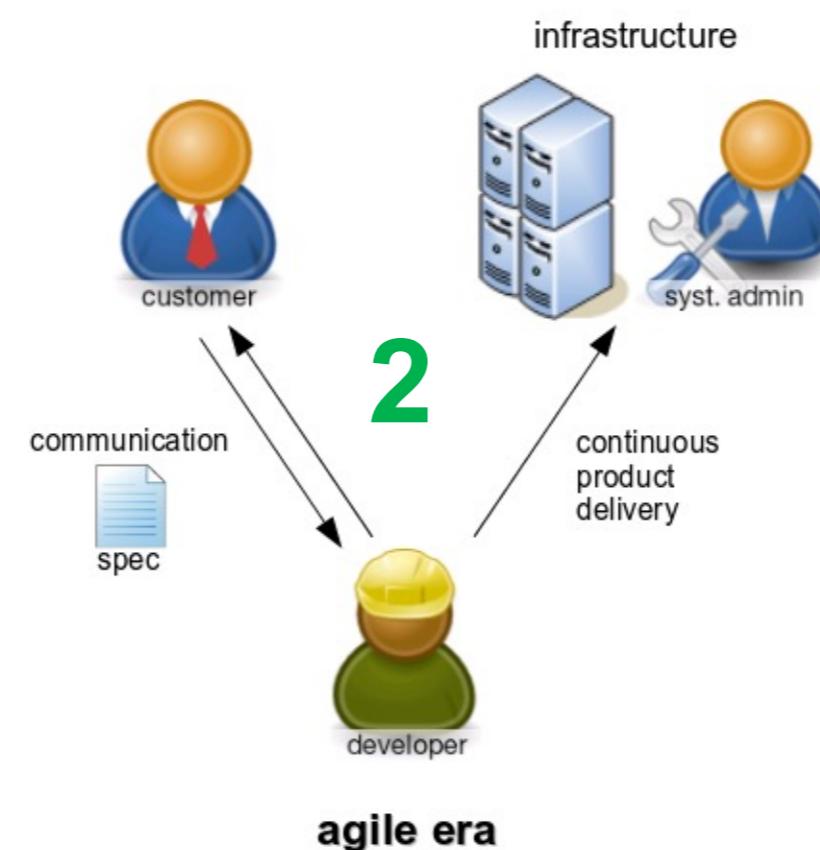
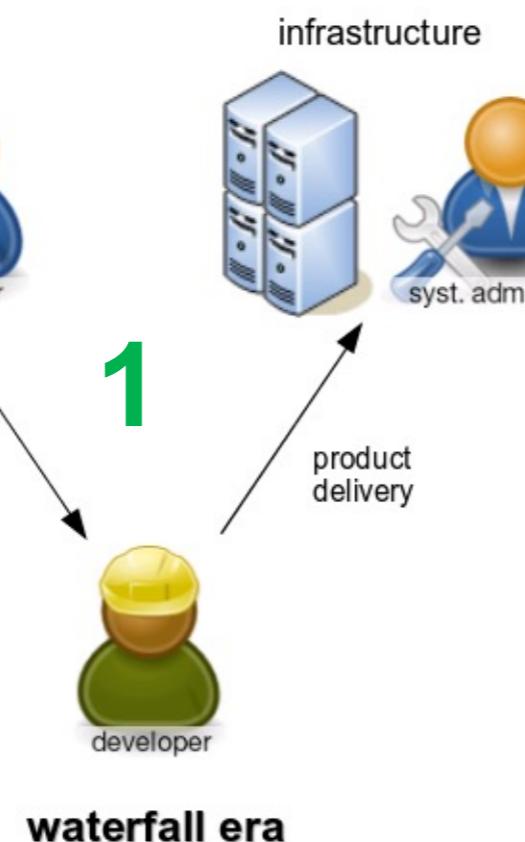
Nom	Mohammad Hamdaqa
Bureau	M-4019
Courriel	mhamdaqa@polymtl.ca
Disponibilité	Sur rendez-vous

Description de l'annuaire

Activités de base d'un processus de génie logiciel. Phases du cycle de vie du logiciel. Outils et méthodes de support au processus. Survol des différentes philosophies de processus de développement de logiciel: discipliné, agile et DevOps. Disciplines techniques du processus du génie logiciel : requis, analyse et conception, implémentation, test. Gestion des projets. Activités DevOps: gestion de configuration, intégration continue, infrastructure-comme-code, stratégies de déploiement/mise-en-production et surveillance post-déploiement.

waterfall => agile => DevOps

1. Pourquoi les processus?
2. Exemples de problèmes
3. Utilités des processus



PARTIE 2 du cours

PARTIE I du cours

Déroulement du cours

Classe inversée (en mode hybride)

- Vidéos pour présenter le contenu théorique;
- Mini-tests pour vérifier la comprehension;
- Discussions, activités, questions et réponses dans la class en présentiel (durée réduite à environ une heure);
- Continuation de discussions, questions et réponses sur Slack;
- TP (en présentiel) pour mettre en pratique le contenu théorique

Ce format peut être modifié
au cours du

Outil à retenir

- Courriels polymtl.ca
- Moodle
- Slack (facultatif)

3

LOG3000-H22 ▾

- @ Mentions & reactions
- : More
- ▼ Channels
 - # general
 - # questions_cours
 - # questions_tps
 - # random
 - # recherche_equipes
- + Add channels
- Direct messages

Calendrier de l'alternance des laboratoires du baccalauréat						
DIMANCHE	LUNDI	MARDI	MERCREDI	JEUDI	VENDREDI	SAMEDI
JANVIER	9 intro/requis Début des cours	10	11	12	13 intro/requis	14
	16 cycle de vie/agile	17	18 TP Groupe B2	19 TP Groupe B2	20	21 cycle de vie/agile
	23 processus/requis	24	25 TP Groupe B1	26	27	28 processus/requis Énoncé TP2 disponible Remise TP1
FÉVRIER	30 comcept./impl./test	31	1 TP Groupe B2	2 TP Groupe B2	3	4 comcept./impl./test
	6 gestion de projet	7	8 TP Groupe B1	9	10	11 gestion de projet Remise TP2
	13 measurement	14	15	16	17	18 measurement
	20	21	22	23	24	25 intra (pas de cours)
MARS	27	28	1	2	3	4 Énoncé TP3 disponible

 POLYTECHNIQUE MONTRÉAL
LE GENIE EN PREMIÈRE CLASSE

Plan de cours

LOG3000 – Processus du génie logiciel
Département de génie informatique et génie logiciel
Hiver 2022
3 Crédits
3 / 1,5 / 4,5
<https://moodle.polymtl.ca/course/view.php?id=987>

Chargés de laboratoire
Hakim Mektoub (hakim.mektoub@gmail.com)
Souleima Zghab (souleima.zghab@polymtl.ca)

Chargés de cours
Nom **Mouna Abidi**
Bureau
Courriel mouna.abidi@polymtl.ca
Disponibilité Sur rendez-vous

Enseignant
Nom **Mohammad Hamdaqa**
Bureau M-4019
Courriel mhamdaqa@polymtl.ca
Disponibilité Sur rendez-vous

Description de l'annuaire
Activités de base d'un processus de génie logiciel. Phases du cycle de vie du logiciel. Outils et méthodes de support au processus. Survol des différentes philosophies de processus de développement de logiciel: discipliné, agile et DevOps. Disciplines techniques du processus du génie logiciel : requis, analyse et conception, implémentation, test. Gestion des projets. Activités DevOps: gestion de configuration, intégration continue, infrastructure-comme-code, stratégies de déploiement/mise-en-production et surveillance post-déploiement.

TP

- **TPI: chevauchement avec LOG3900 (Olivier Gendreau)**
- TP2 à TP5: en binômes!
- Indiquez vous équipes/binômes dans le Google Sheet:
 - shorturl.at/ryBN0

Examens

- Intra: à distance (“take-home”)
- Final: à déterminer
(probablement en présentiel)

Quelques autres choses

- Gérez bien votre temps
- Gérez vos attentes
- Prenez soin de vous

En présentiel

- Suivre les recommandations sanitaires
- Suivre les directives pour du test et du isolement
- Garder une distanciation

Pour nous contacter



Enseignant
Mohammad Hamdaqa
mhamdaqa@polymtl.ca



Chargée de cours
Mouna Abidi
mouna.abidi@polymtl.ca



TP-B1
Hakim Mektoub
hakim.mektoub@gmail.com



Chargés de laboratoire
TP-B2
Souleima zghab
souleima.zghab@polymtl.ca



LOG3000

Processus du génie
logiciel

Motivation

- Pourquoi les processus?
- Exemples de problèmes
- Utilités des processus

Ingénierie Logicielle == Programmation?

```
@IBOutlet weak var documentNameLabel: UILabel?  
  
var document: UIDocument?  
  
override func viewDidAppear(_ animated: Bool) {  
    super.viewDidAppear(animated)  
  
    // Access the document  
    document?.open(completionHandler: { (success) ->  
        if success {  
            // Display the content of the document.  
            self.documentNameLabel.text = self.document?.name  
        } else {  
            // Make sure to handle the failure case.  
            // message to the user.  
        }  
    })  
}
```

System Shock reboot that raised \$1.3M on Kickstarter is now on ‘hiatus’

“We took the wrong path”

By Colin Campbell | @ColinCampbellx | Feb 16, 2018, 2:00pm EST

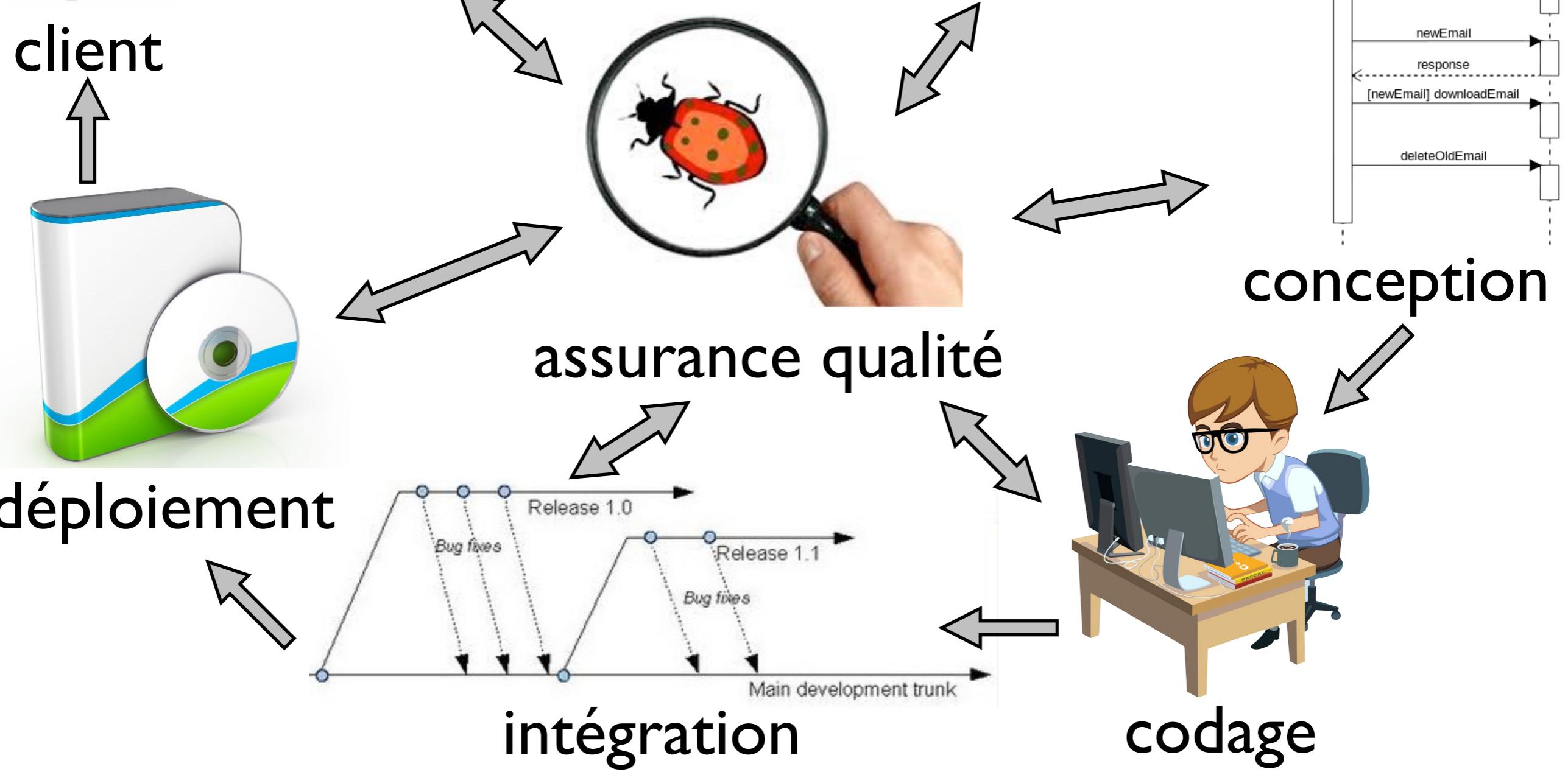
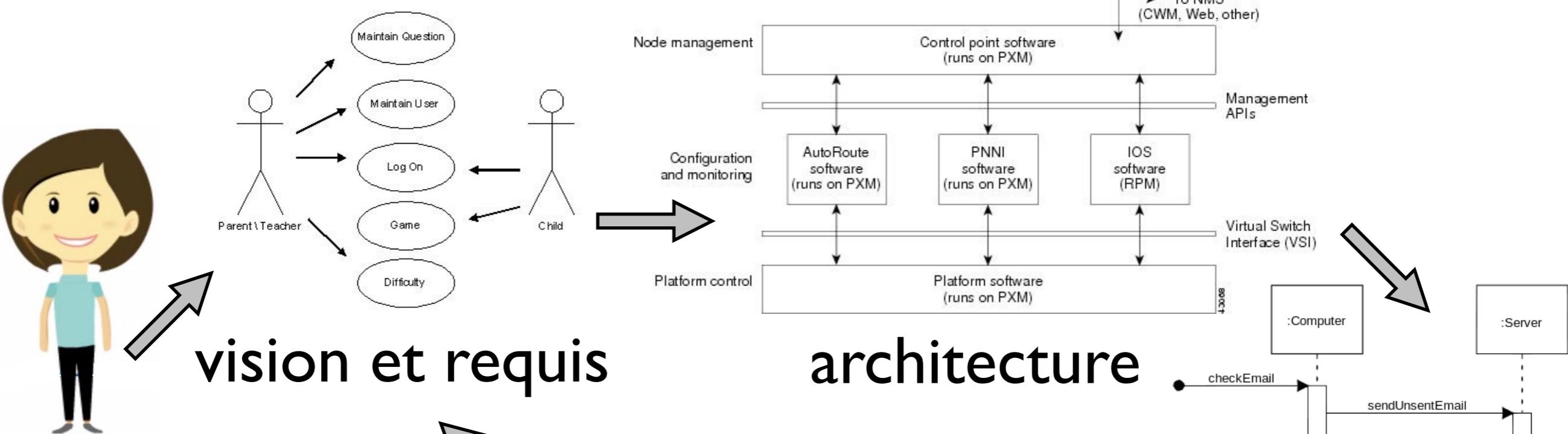
f t  SHARE

“mission **creep**” and **unrealistic** ambitions had eaten up the Kickstarter **funds**”



“Nightdive originally created its remake in the Unity engine, but **switched to the Unreal Engine** last year.”

““Maybe we lost our **focus**,” he wrote. “The **vision** began to change. We moved from a Remaster to a completely new game.”



What Makes A Great Software Engineer?

Paul Luo Li^{*+}, Andrew J. Ko*, Jiamin Zhu⁺

Microsoft⁺
Seattle, WA
{pal,jiaminz}@microsoft.com

ICSE 2015

The Information School
University of Washington
ajko@uw.edu

Abstract—Good software engineers are essential to the creation of good software. However, most of what we know about software-engineering expertise are vague stereotypes, such as ‘excellent communicators’ and ‘great teammates’. The lack of specificity in our understanding hinders researchers from reasoning about them, employers from identifying them, and young engineers from becoming them. Our understanding also lacks breadth: what are all the distinguishing attributes of great engineers (technical expertise and beyond)? We took a first step in addressing these gaps by interviewing 59 experienced engineers across 13 divisions at Microsoft, uncovering 53 attributes of great engineers. We explain the attributes and examine how the most salient of these impact projects and teams. We discuss implications of this knowledge on research and the hiring and training of engineers.

Index Terms—Software engineers, expertise, teamwork

I. INTRODUCTION

Software engineering research has considered a vast number of factors that affect project outcomes, from process and tools, to programming languages and requirement elicitation. We rarely give consideration, however, to one of the most fundamental components of software engineering: the engineers themselves. Specifically, what makes a software engineer great? This basic question is at the foundation of nearly every part of our world’s rapidly growing software ecosystem: employers want to hire and retain great engineers, universities want to train

In this study, we sought to remedy the lack of specificity, breadth, and rigor in prior work by investigating the following questions about *software engineers*:

- What do expert software engineers think are attributes of great software engineers?
- Why are these attributes important for the engineering of software?
- How do these attributes relate to each other?

To answer these questions, we performed 59 semi-structured interviews, spanning 13 Microsoft divisions, including senior interviews with architect-level engineers with over 25 years of experience. The contribution of this effort is a thorough, specific, and contextual understanding of software engineering expertise as viewed by expert software engineers.

In the rest of this paper, we detail our current understanding of software engineering expertise. We then discuss our interview and analysis methodology, the attributes we discovered, and the implications of this knowledge for software engineering research, practice, and training.

II. RELATED WORK

Much of our knowledge of software engineering expertise has come from studying *new* engineers rather than experienced engineers. For example, the closest work to ours is Hewner and Guzdial’s

« Mais on est vraiment super bon! »

- Voici ce qui fait un bon ingénieur logiciel, selon des recherches faites chez Microsoft (2015):
 1. Est prêt à s'améliorer, à **s'adapter**;
 2. connaît bien les **gens** et l'organisation;
 3. est capable de **communiquer** avec les autres;
 4. produit du matériel **simple** et intuitif.
- Où sont les compétences techniques?
 - Être bon programmeur se trouverait vers la fin de la liste.
- Une **équipe** de programmeurs moyens bien organisée réussit d'habitude mieux que des **cracks** qui travaillent de manière isolée.
 - Un peu comme une équipe de hockey, de football.

Pourquoi un (autre) cours sur les processus?

- Comment produire un logiciel de qualité?
 - Qualité en utilisation
 - Nombre de bogues; → Ex.: Tests et correction d'erreurs
 - répond aux besoins des clients; → Ex.: Rencontres
 - facilité d'utilisation; → Ex.: Prototypes d'interface
 - etc.
 - Qualité du produit
 - Maintenabilité; → Ex.: Design flexible aux changements
 - tolérance aux pannes; → Ex.: Mécanismes de gestion d'erreurs
 - sécurité; → Ex.: Revues de codes par un expert
 - etc.
 - Qualité des données
 - Traçabilité des données; → Ex.: Suivi post-livraison
 - portabilité des données; → Ex.: Fonctionnalités d'exportation
 - etc.

Source : Norme ISO 25000 (2005)

Pourquoi un (autre) cours sur les processus?

- Comment s'assurer de respecter les **dates limites**?
 - Planifier chaque étape du projet;
 → Ex.: Jalons intermédiaires
 - baser les estimations sur ce que l'on sait des projets précédents;
 → Ex.: Collecte de données en cours de projet
 - avoir toujours un produit fonctionnel;
 → Ex.: Intégration continue
 - faire une mise en production fréquente;
 → Ex.: Livraison continue
 - connaître l'avancement du travail de chacun;
 → Ex.: Réunions de suivi
 - gérer les imprévus;
 → Ex.: Planifier des alternatives
 - etc.

Pourquoi un (autre) cours sur les processus?

- Comment s'assurer de bien utiliser les ressources?
 - Garantir un effort constant tout au long du projet;
 → Ex.: Paralléliser les activités
 - planifier des activités de complexité égales;
 → Ex.: Design modulaire du code
 - limiter les heures supplémentaires;
 → Ex.: Cycles de temps fixe et court
 - répartir les responsabilités de chacun;
 → Ex.: Définir les compétences de chaque personne
 - prévoir les activités d'**opération** dès l'implementation;
 → Ex.: Infrastructure-as-code
 - etc.

Pourquoi un (autre) cours sur les processus?

- Donc: Nous avons besoin de planifier et d'automatiser!

Afin que ça sorte chaque jour à midi, on va adopter des technologies DevOps pour établir une routine de mise en production automatisée.

Si un module est écrit en mandarin, il faut s'assurer que son code soit compréhensible.

Le logiciel doit servir pendant vingt ans, il faut qu'il soit bien documenté.

On est déjà débordé par d'autres projets. Il faut choisir une approche qui minimise l'effort requis.

Un bogue pourrait tuer quelqu'un. Il faut choisir un design qui facilite les tests.



Source : The Simpsons,
épisode S10E19,
« Mom and Pop Art »

Exemples classiques de problèmes

- Vol 2904 de la Lufthansa (1993): Atterrissage de travers non géré par le logiciel
 - Inversion des moteurs contrôlée par logiciel → Possible si les deux trains d'atterrissage sont sur le sol.
 - Sauf que ... important vent de travers → Pression sur les trains d'atterrissage insuffisante → Incapacité d'inverser les moteurs
 - Accident en bout de piste → 2 morts et 68 blessés.

Quels sont les problèmes de processus?



1. Pourquoi les processus?

2. Exemples de problèmes

3. Utilités des processus

Exemples classiques de problèmes

- Incident de Dharhan (1991): Anti-missile Patriot rate sa cible de 600 mètres
 - **Erreur de conception** de l'horloge → Perte de précision lorsque le logiciel est en fonction trop longtemps.
 - **Tests trop limités** → Tests sur quelques heures, alors qu'en réalité le logiciel doit fonctionner des centaines d'heures.
 - **Communication déficiente** → L'armée israélienne avait découvert le problème et l'on contourné → Le message ne s'est pas rendu au champ de bataille.
 - Résultat → Missile non-intercepté → 28 morts.

Quels sont les problèmes de processus?



15

Source : wikipedia

Exemples classiques de problèmes

1. Pourquoi les processus?

2. Exemples de problèmes

3. Utilités des processus

- Therac-25 (1985-1987): Machine de radiothérapie canadienne avec défauts logiciels
 - Réutilisation de code déficient → Code fonctionnel dans l'ancienne machine à cause de blocages matériels → **Code réutilisé non testé** parce que l'ancienne machine n'avait jamais provoqué d'accident.
 - Problème rare, difficile à reproduire (« *race condition* ») + **Suivi inadéquat des plaintes** = Plus de deux ans pour découvrir qu'il y avait un problème.
 - Résultat → Application de 100x la dose prévue → Six incidents connus résultants en plusieurs morts.

Quels sont les problèmes de processus?



Exemples classiques de problèmes

- Mars Climate Orbiter (1999): Mélange entre Newton-secondes (SI) et Livre-secondes (Impérial)
 - Exigences initiales envoyées au sous-contractant en **impérial** → Client s'attendait finalement à des données **métriques**.
 - Problème de trajectoire détecté lors du vol entre la Terre et Mars, mais les inquiétudes des navigateurs n'ont pas été prises au sérieux..
 - Résultat → Perte de la sonde → 330 M\$ perdus.

Quels sont les problèmes de processus?



Exemples classiques de problèmes

- Ariane 5 vol 501 (1996): Perte de contrôle de la fusée
 - Ariane 4 a été un succès retentissant → **Code de contrôle réutilisé sans être testé.**
 - Le nouveau contexte était différent → Raccourci 64-bits vers 16-bits dans le vieux contexte a causé un crash dans le nouveau.
 - Le module ayant causé le crash n'était plus utile pour Ariane 5. Il a été néanmoins exécuté ... → Le crash du module a causé un crash de tout le système → Impossible de contrôler la fusée.
 - Résultat → Perte de la fusée → 370 M\$ perdus.

Quels sont les problèmes de processus?



- Voici de quoi a l'air une ligne de code valant 370 M\$:

```
P_M_DERIVE(T_ALG.E_BH) := UC_16S_EN_16NS (TDB.T_ENTIER_16S  
((1.0/C_M LSB_BH) * G_M_INFO_DERIVE(T_ALG.E_BH)));
```

Exemples classiques de problèmes

- Projet LASCAD (1992): Système de répartition des appels 9-1-1 pour la ville de Londres (Royaume-Uni).
 - Deuxième essai, alors imposition d'un échéancier trop serré (11 mois). 85% des soumissionnaires ont dit que cet **échéancier était irréaliste**.
 - Contrat octroyé par appel d'offre à la **soumission la plus basse** ... qui était deux fois plus basse que toutes les autres.
 - Le **soumissionnaire n'avait aucune expérience** dans des systèmes d'envergure et encore moins dans le domaine visé.
 - Un système comportant beaucoup de bogues a été lancé en production et a échoué au bout de neuf jours: beaucoup d'appels ont été perdus.
 - Impact : Difficile à évaluer, mais possiblement 46 morts.

Quels sont les problèmes de processus?



Exemples classiques de problèmes

1. Pourquoi les processus?

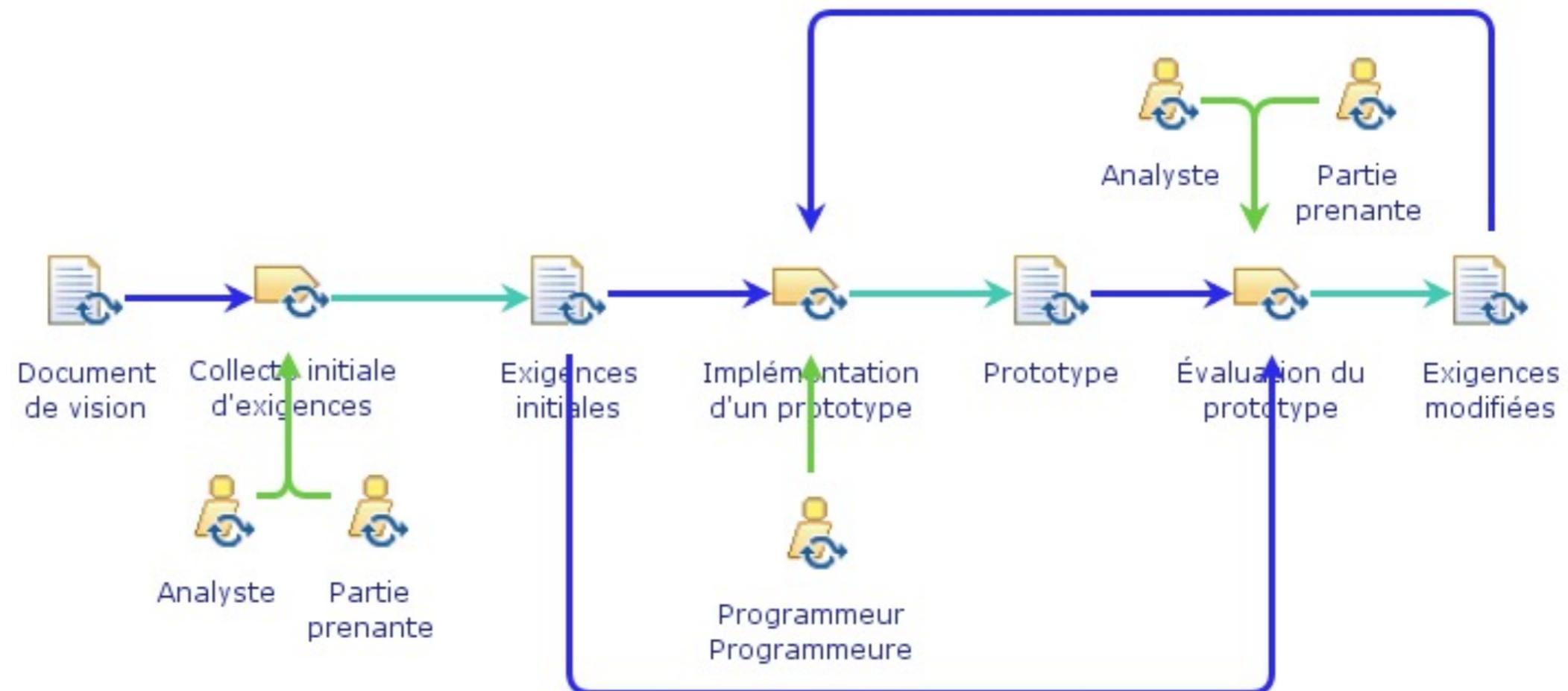
2. Exemples de problèmes

3. Utilités des processus

- Nouveaux problèmes avec l'internet et le nuage: la traçabilité et la sécurité des données.
 - Janvier 2009: Heartland Payment Systems rapporte un vol d'environ 100 millions de cartes provenant de 650 banques.
 - Décembre 2009: le site de jeux RockYou! rapporte le vol d'une base de données de 32 millions de noms d'utilisateurs et de mots de passe pas encrypté.
 - Avril-mai 2011: Sony PSN rapporte le vol de 77 millions d'utilisateurs, suivi de 25 millions d'autres un mois plus tard.
 - Octobre 2013: Adobe rapporte le vol de 130 millions de dossiers d'utilisateurs.
 - Novembre 2013: Target se fait voler 70 millions de dossiers d'utilisateurs et se voit convoquer par le congrès américain. Target rapporte 1 M\$ de frais par jour dans les deux mois suivant cet événement.
 - Juillet 2015: AshleyMadison fait face à un chantage par des hackers ayant volé les informations de 37 millions d'utilisateurs.
 - Avril 2016: Panama Papers: Mossack Fonseca se fait voler 11,5 millions de documents confidentiels (2,6 To de données!).
 - Décembre 2016: Yahoo se fait avertir que les données de 1,5 milliards d'utilisateurs sont vendues à des tiers.
 - **Problème de processus: Dans la majorité des cas, des avertissements ont été ignorés!**

Quel est un processus ?

- Exemple simple de modèle de processus :



➤ Un processus complet réel peut avoir des centaines d'éléments:

1. Rôles
2. Activités
3. Artéfacts

1. Pourquoi les processus?

2. Exemples de problèmes

3. Utilités des processus

À quoi servent les processus?

- Communication
 - **Interne**: Le processus permet aux employés de comprendre comment le logiciel est développé, à quoi sert leur travail, quelles informations sont disponibles, etc.
 - **Externe**: Le processus peut être évalué et donner lieu à des certifications (ex.: CMMI, ISO 15504). Parfois le processus est demandé par le client.
- Évaluation des pratiques en place
 - **Diagnostic**: On peut détecter des problèmes dans les manières actuelles de développer le logiciel.
 - **Amélioration**: On peut introduire de nouvelles pratiques susceptibles d'aider au développement du logiciel.
- Planification du travail
 - **Haut niveau**: Le processus présente un point de vue plus générique que le plan de projet. Un processus peut servir à la génération de plusieurs projets.
 - **Orienté produit**: Le processus se concentre sur le produit, tandis que le plan de projet se concentre sur les ressources.
- Etc.

À quoi servent les processus?

Objectif du cours:

Vous faire **réfléchir** sur votre manière de développer le logiciel.

1. Pourquoi les processus?

2. Exemples de problèmes

3. Utilités des processus



1. Le document
d'exigences

2. Qualité des
exigences

3. Autres formes
de document
d'exigences

LOG3000

Processus du génie logiciel

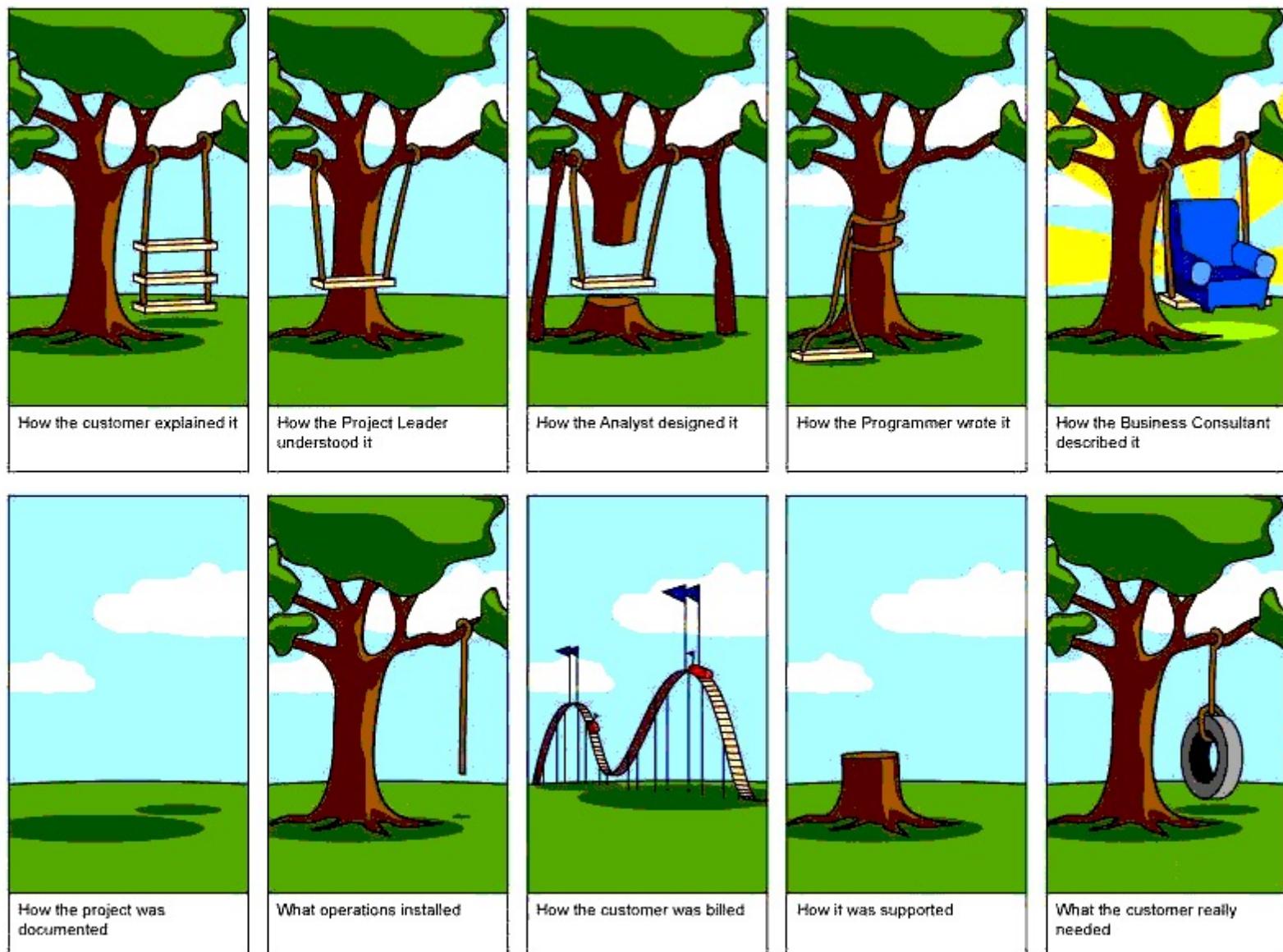
Qualité des exigences

1. Le document
d'exigences

2. Qualité des
exigences

3. Autres formes
de document
d'exigences

Comprendre les exigences = essentiel



Le document traditionnel d'exigences

- SRS: *Software Requirements Specification*
 - Le SRS traditionnel a pour but de servir de dépôt pour toutes les exigences du logiciel.
 - Le SRS traditionnel transforme les besoins flous et abstraits des parties prenantes (*stakeholders*) en **fonctionnalités concrètes et mesurables**.
 - Les exigences du SRS traditionnel sont utilisées pour:
 - Définir la **portée** du projet;
 - fractionner le projet en **paquetages**;
 - servir de base pour **l'analyse et la conception**;
 - définir les tests de **validation**;
 - rédiger le **contrat** avec le client (\$\$\$);
 - etc.

Types d'exigences

- Exigences fonctionnelles: Décrit les **fonctionnalités** du produit logiciel.
 - Ex.: Le menu principal doit offrir à l'utilisateur l'option de création d'un fichier.
 - Ex.: Le système doit permettre de déplacer la vue de la caméra par un glissement à deux doigts.
- Exigences non-fonctionnelles: Décrit les **qualités** du produit logiciel.
 - Ex.: Le système doit être en panne moins de 9 heures par année (99.9 % « *uptime* »).
 - Ex.: Le serveur doit être en mesure de gérer au moins cinq jeux en même temps sans perte de performance.

1. Le document
d'exigences2. Qualité des
exigences3. Autres formes
de document
d'exigences

Distinction
importante
parce qu'on ne
gère pas une
fonctionnalité
comme une
qualité !

Fonctionnalité ou qualité?

- Règle du pouce:
 - Si cela peut être construit durant une tâche bien définie, c'est une **fonctionnalité**.
 - S'il s'agit de quelque chose à surveiller durant une bonne partie du projet, c'est une **qualité**.

Exigence	Fonctionnalité ou qualité?
Le système doit permettre à l'utilisateur de sauvegarder son travail.	
Le système doit offrir un temps de latence maximal de 2 secondes entre l'envoi d'une requête et la réception de la réponse.	
Un nouvel utilisateur doit être capable d'utiliser tout l'éditeur après une formation d'une durée maximale d'une heure.	
Les tests unitaires doivent couvrir au moins 90 % du code.	
Le système doit permettre l'authentification d'un utilisateur grâce à l'identité OpenID de l'utilisateur.	
Chaque fenêtre présentée par le système doit permettre à l'utilisateur d'y chercher des chaînes de caractères.	
Le système doit permettre à l'utilisateur de défaire et refaire (« <i>undo/redo</i> ») toutes les actions effectuées dans l'éditeur.	

Gabarit pour le SRS traditionnel

- Norme IEEE 830: Définit un gabarit standard pour la structure du SRS.
 1. Introduction: objectif, lexique des termes, références externes.
 2. Description générale: interfaces, contraintes, hypothèses.
 3. Exigences
 - 3.1. Exigences spécifiques: fonctionnalités.
 - 3.2. Attributs du système: qualités comme fiabilité, disponibilité, sécurité, maintenabilité, portabilité, etc.

Qualité des exigences

- Le SRS traditionnel est un document **semi-formel**: Il doit être le plus clair et le plus compréhensible possible, tout en utilisant un langage naturel.
 - Exemple formel: $\forall x, y \in \mathbb{N}^*, x^y \in \mathbb{N}^*$
 - Exemple informel: Le logiciel doit être beau.
- Deux niveaux de qualité à considérer:
 - Qualité des exigences individuelles;
 - qualité du document au complet

Référence de base : « *Identifying and Measuring Quality in a Software Requirements Specification* » de Davis et al. (1993).

Qualité des exigences individuelles

- **Qualité de la langue:** Syntaxe et grammaire.

	Le system doigt permettre a utilisteur d'execeauter recherches en paralelles.
	Le système doit permettre à un utilisateur d'exécuter plusieurs recherches en parallèle.

Pourquoi?

- **Structure d'écriture:** Le <sujet> doit <complément> ...

	Le système authentifie un utilisateur sur le serveur.
	Le système doit permettre à un utilisateur de s'authentifier sur le serveur.

Pourquoi?

Qualité des exigences individuelles

- Utilisation du pluriel: attention aux ambiguïtés.

	Les utilisateurs doivent être capables d'exécuter plusieurs recherches en parallèle.
	Le système doit permettre à un utilisateur d'exécuter plusieurs recherches en parallèle.
	Chaque utilisateur doit être capable d'exécuter plusieurs recherches en parallèle.

Pourquoi?

- Utilisation du négatif: à éviter le plus possible.

	Le système ne doit pas empêcher à un utilisateur de s'identifier une fois ou moins.
	Le système ne doit pas permettre à un utilisateur de s'authentifier deux fois en même temps.
	Le système doit empêcher un utilisateur de s'authentifier deux fois en même temps.

Pourquoi?

Qualité des exigences individuelles

- Éviter les exigences composées.

	Les mots de passe doivent être hachés et la base de données ne doit pas être accessible aux utilisateurs.
	Les mots de passe doivent être hachés. La base de données ne doit pas être accessible aux utilisateurs.

Pourquoi?

- Les exigences doivent être **testables**.

	Le système doit être facile à utiliser.
	Le système doit comprendre un manuel d'utilisateur qui détaille chaque fonctionnalité. Le système doit offrir des info-bulles (« <i>tooltip</i> ») pour chaque fonctionnalité. Le système doit permettre à l'utilisateur de naviguer à travers les contrôles grâce à la touche de tabulation.

Pourquoi?

Qualité des exigences individuelles

- **Indépendance du design.**

	Le système doit pouvoir être fermé en tout temps grâce à un bouton avec un 'X' sur le coin en haut à droite de la fenêtre.
	Le système doit pouvoir être fermé en tout temps grâce à un bouton sur la fenêtre.

Pourquoi?

- **Les exigences doivent être faisables.**

	Le système ne doit jamais tomber en panne.
	Le système doit être accessible 99 % du temps.

Pourquoi?

Qualité des exigences individuelles

- **Les exigences doivent être compréhensibles.**

	Le SoBC doit permettre un ITR sur la BD avec un temps de latence adéquat selon IEEE 14764-1998-3.
	Le système doit permettre de lister toutes les entrées de la base de données en moins de 15 ms.

Pourquoi?

Qualité du document complet (1/2)

C01B

Qualité des exigences

1. Le document d'exigences

2. Qualité des exigences

3. Autres formes de document d'exigences

- **Complétude:** Tous les besoins exprimés sont là.
- **Correct:** Pas d'exigences superflues.
- **Cohérence interne:** Pas de contradiction entre exigences.
- **Cohérence externe:** Pas de contradiction avec les documents externes fournis.
- **Traçabilité:** Numéro unique pour chaque exigence.
 - Attention à la numérotation automatique! Un numéro assigné à une exigence devrait rester le même **pour tout le projet**.
 - Utilisez la même approche que la numérotation des bogues.
- **Organisation:** Faciliter la recherche d'information.
 - Ne pas faire un fourre-tout d'exigences pêle-mêle. Approches possibles d'organisation des exigences fonctionnelles:
 - Selon les futurs paquetages;
 - selon l'organisation du travail;
 - selon les types d'utilisateur, d'utilisatrice;
 - etc.

Qualité du document complet (2/2)

- **Concision:** Niveau de détail adéquat.
- **Ambiguïté:** Définir un glossaire et **réutiliser toujours les mêmes termes.**
 - Ne pas utiliser des termes différents pour parler des mêmes notions.
 - Ex.: compte utilisateur, profil utilisateur, données de l'utilisateur, etc.
- **Redondance:** Comme pour le code, à éviter autant que possible.
 - Définir le concept une fois, et y faire référence par la suite grâce aux **numéros uniques.**

Autres formes d'exigences

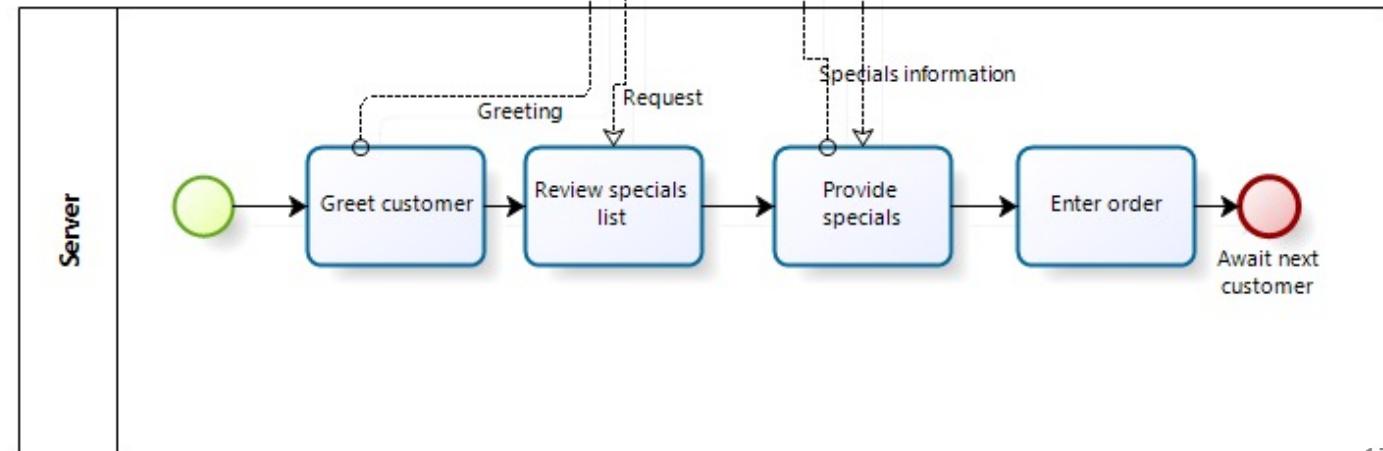
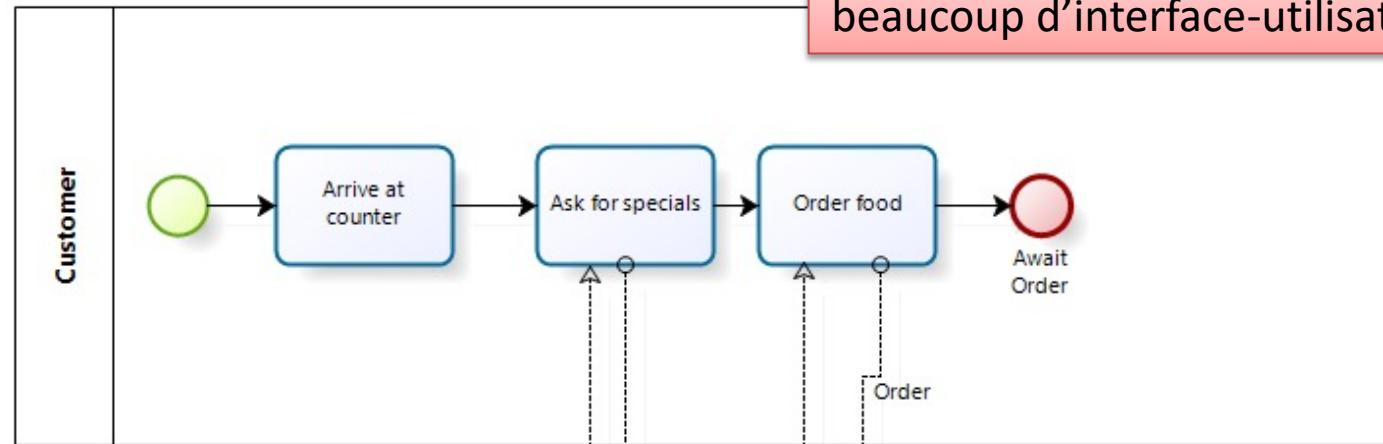
- Récit utilisateur (*user story*): Similaires aux exigences traditionnelles, mais un peu plus haut niveau → Populaires en Agile.
- Structure: En tant que <rôle>, quant <contexte>, je veux faire <fonctionnalité> afin de <objectif>.

As a user, when closing the application,
I want to be prompted to save if I have made any change
in my data since the last save,
so that I can preserve useful work and discard erroneous
work.

Autres formes d'exigences

- Cas d'utilisation, scénario, processus d'affaire, etc.: Présente une série de fonctionnalités avec chemins alternatifs.

Utile pour les logiciels ayant beaucoup d'interface-utilisateurs



1. Le document
d'exigences2. Qualité des
exigences3. Autres formes
de document
d'exigences

Exigences en langage formel

- Exigences mathématiques formelles

$\text{patient} \neq \text{no_patient}$

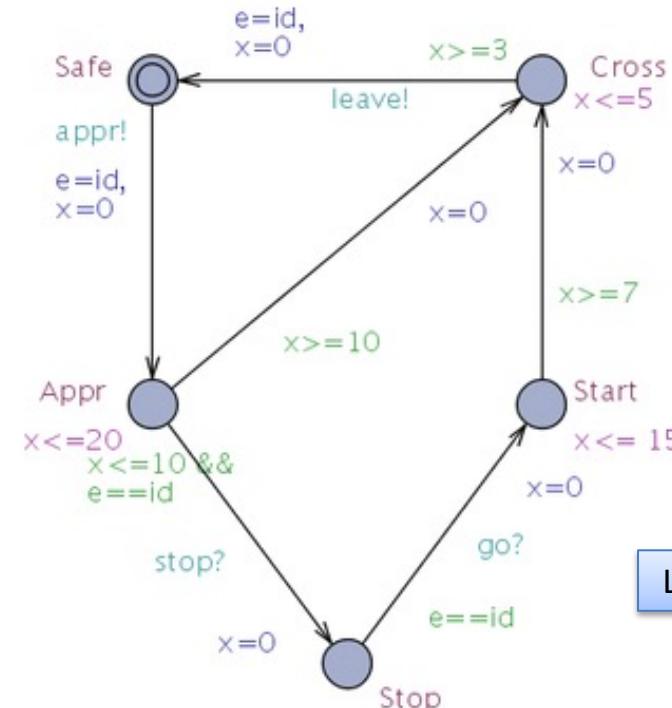
$\text{field}^* = \text{field}?$

$\text{fields}' = \text{fields} \cup \{ \text{field}' \mapsto \text{prescribed}' \}$

$\text{mode} = \text{therapy} \Rightarrow \text{counters}' = \text{counters} \cup \{ \text{field}' \mapsto \text{accumulated}' \}$

Langage Z, exemple tiré de Jacky et al. (1997)

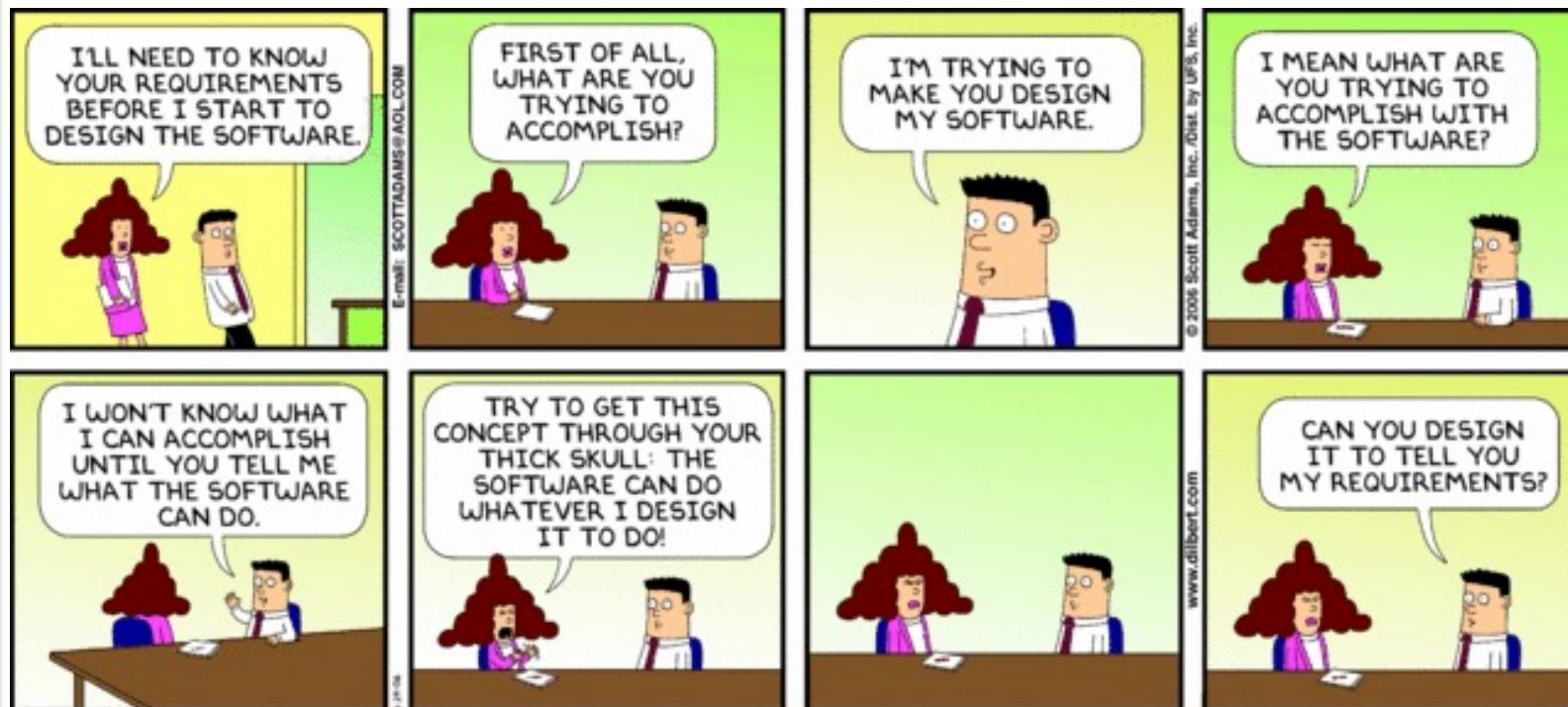
- Exigences en modèles formels



Langage UPPAAL, tutoriel d'utilisation

1. Le document d'exigences
2. Qualité des exigences
3. Autres formes de document d'exigences

- Obtenir de bonnes exigences n'est pas toujours facile...



- Prochainement: approches de collecte des exigences.

LOG3000

Processus du génie logiciel

1. Le travail de l'ingénieur
2. Introduction aux cycles de vie
3. Cascade
4. Incrémental
5. Transformation.
6. Spirale
7. Conclusion sur les cycles de vie

Le cycle de vie

1. Le travail de l'ingénieur

2. Introduction aux cycles de vie

3. Cascade

4. Incrémental

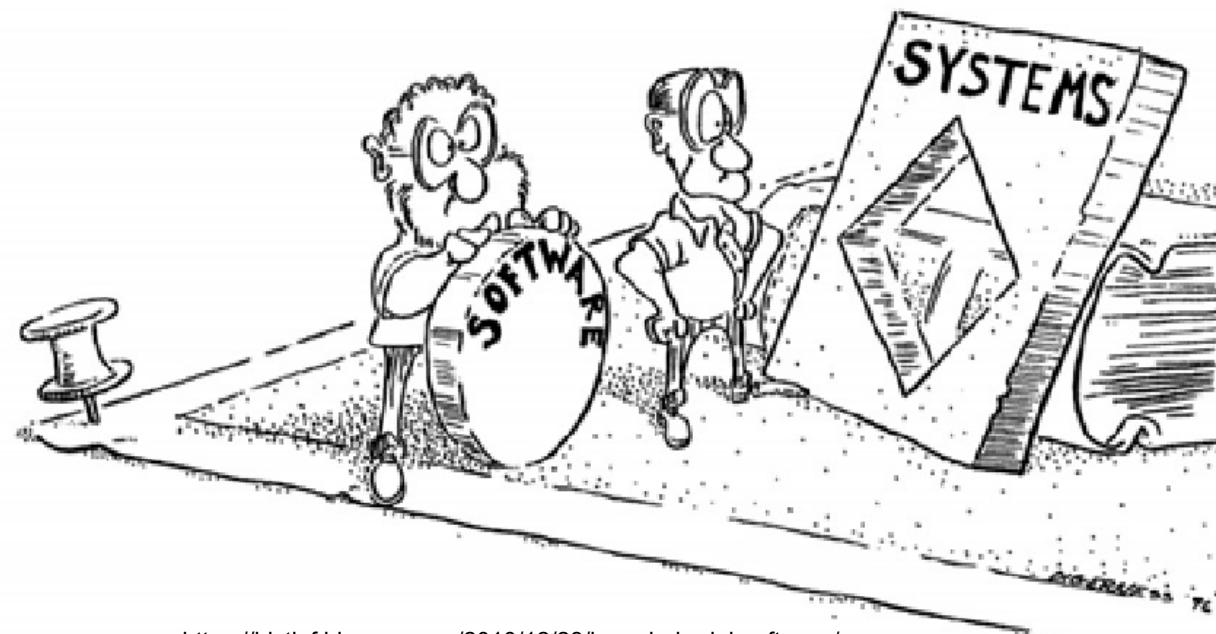
5. Transformation.

6. Spirale

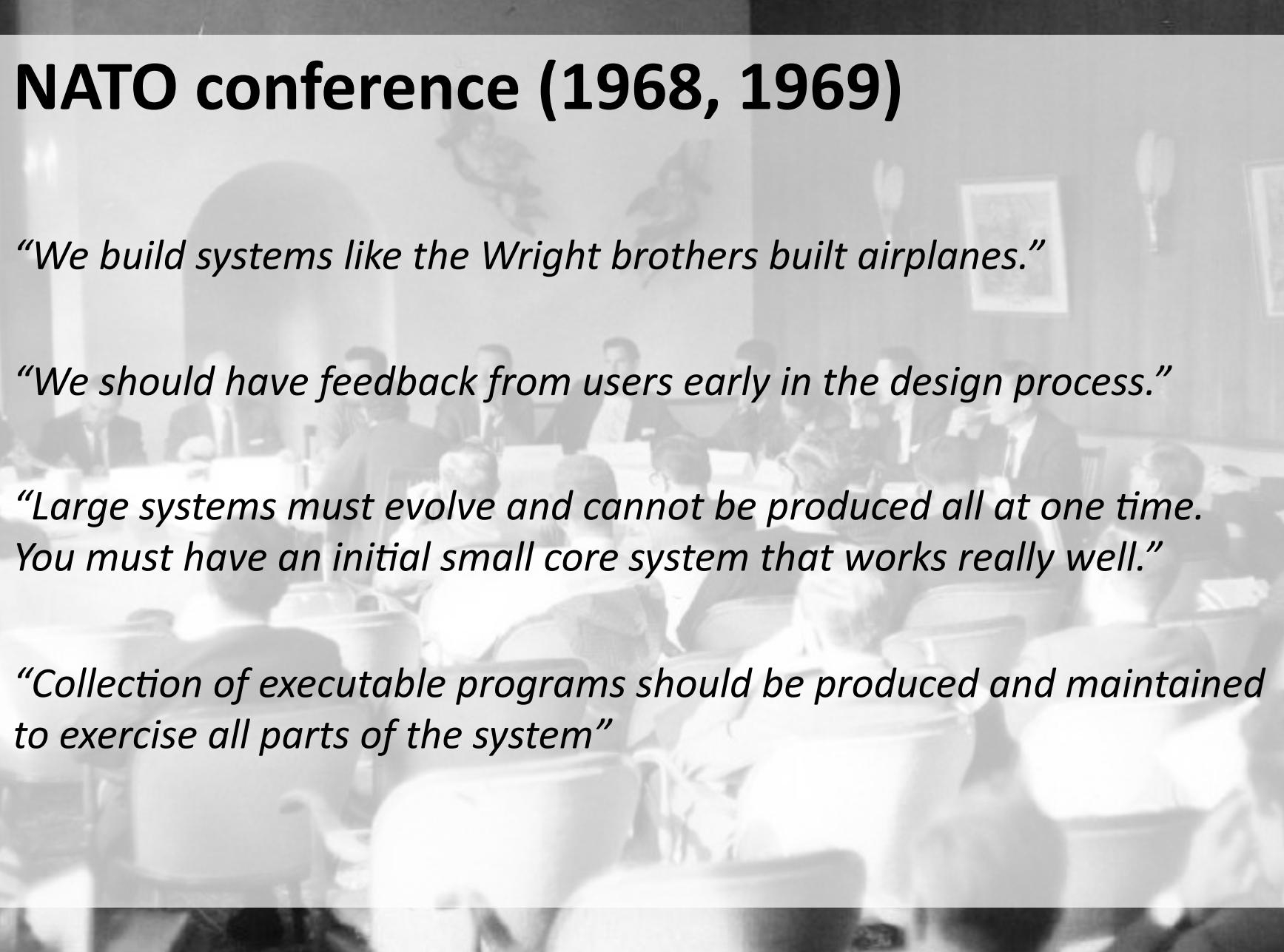
7. Conclusion sur les cycles de vie

Origine du terme « génie logiciel »

- Crise du logiciel
 - programmation par essais-erreurs;
 - réalisation par assemblage individuel;
 - très peu de contrôle sur les ressources;
 - la nouveauté prône sur la fiabilité;
 - peu de partage entre les équipes.



NATO conference (1968, 1969)



“We build systems like the Wright brothers built airplanes.”

“We should have feedback from users early in the design process.”

“Large systems must evolve and cannot be produced all at one time. You must have an initial small core system that works really well.”

“Collection of executable programs should be produced and maintained to exercise all parts of the system”

NATO conference (1968, 1969)

*“The phrase ‘software engineering’ was deliberately chosen as being provocative, in implying the need for software manufacture to be based on the types of **theoretical foundations** and **practical disciplines**, that are traditional in the established branches of engineering.”*

L'objectif: Construire des bases **théoriques** et **pratiques** reconnues pour le développement logiciel, comme pour les autres génies.

SOFTWARE ENGINEERING

Report on a conference sponsored by the

NATO SCIENCE COMMITTEE

Garmisch, Germany, 7th to 11th October 1968

1. Le travail de l'ingénieur

2. Introduction aux cycles de vie

3. Cascade

4. Incrémental

5. Transformation.

6. Spirale

7. Conclusion sur les cycles de vie

L'ingénierie traditionnelle



Le génie logiciel tel que perçu en 1968



1. Le travail de l'ingénieur

2. Introduction aux cycles de vie

3. Cascade

4. Incrémental

5. Transformation.

6. Spirale

7. Conclusion sur les cycles de vie

Comment construire un processus?

- Les processus sont complexes: C'est plus facile si on part d'un modèle de base.
- **Cycle de vie:** Ensemble des phases par lesquelles passe une entité de sa création à sa destruction.



- Vue de très haut niveau.
- Phases avec frontières floues.

1. Le travail de l'ingénieur

2. Introduction aux cycles de vie

3. Cascade

4. Incrémental

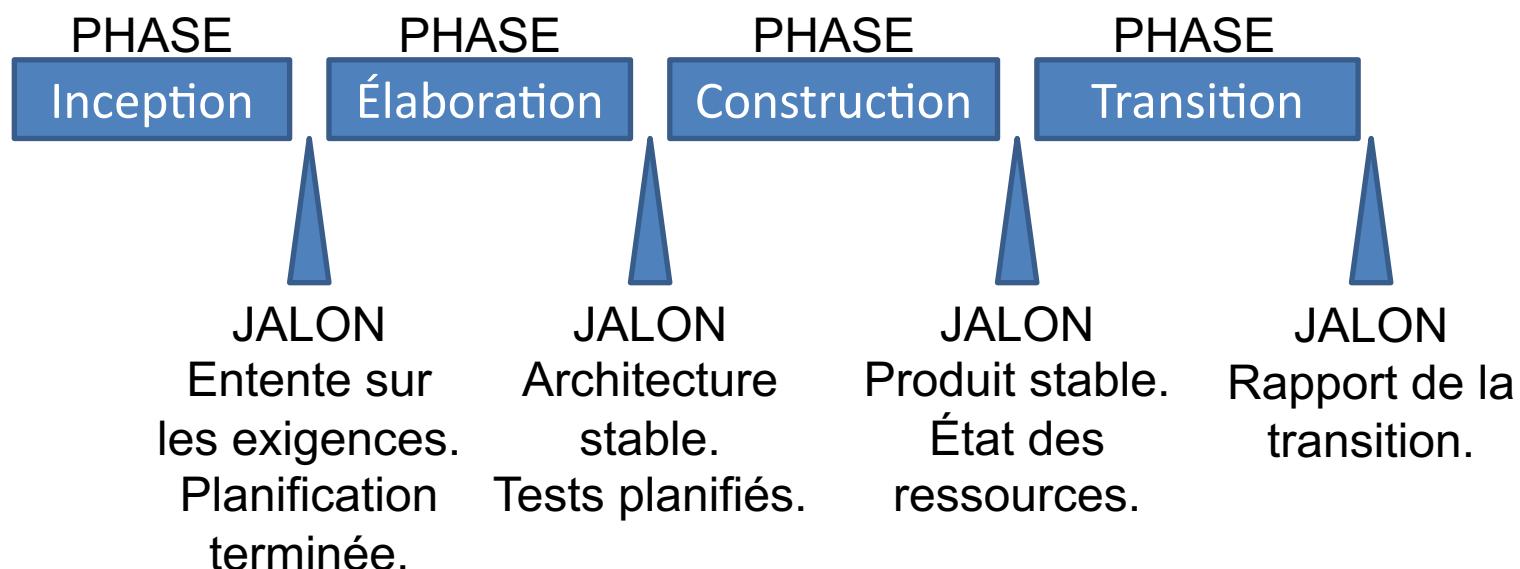
5. Transformation.

6. Spirale

7. Conclusion sur les cycles de vie

Éléments du cycle de vie

- Phase: Le temps entre deux jalons.
- Jalon (*milestone*): Définit des objectifs et des **artéfacts** à compléter, et décide s'il faut passer à la phase suivante.
- Ex.: Pour un cycle de vie cascade:



1. Le travail de l'ingénieur

2. Introduction aux cycles de vie

3. Cascade

4. Incrémental

5. Transformation.

6. Spirale

7. Conclusion sur les cycles de vie

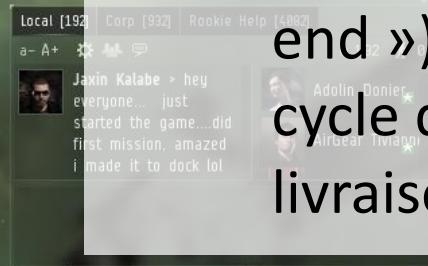
Le processus est un compromis

- **Chaque cycle de vie a ses forces et ses faiblesses.** Le choix du cycle de vie dépend du contexte/risque:
 - Dans mon contexte, avec quelles faiblesses/risques suis-je capable de vivre (trop complexe, requis pas clairs, requis vont changer, beaucoup d'argent impliqué, ...)?
 - Dans mon contexte, de quelles forces ai-je besoin ?
- Note: la plupart des organisations n'ont qu'un seul cycle de vie et il n'est pas facile (ni recommandé) de le changer.
 - Certaines organisations commencent à intégrer différents gabarits de processus basés sur des cycles de vie différents, dépendant de la complexité du projet.



Exemple : CCP Games et le jeu en ligne Eve Online.

- Processus 1 : **L'interface** doit être changée rapidement, donc utilisation d'un **cycle de vie rapide** permettant de livrer des mises-à-jour à chaque mois.
- Processus 2 : Changer le **moteur** (« engine », « back-end ») est très **risqué**, alors il vaut mieux prendre un **cycle de vie avec plus d'étapes de vérification** et 1-2 livraisons par année.



Stop Ship
Click here to stop your ship

EVE

1. Le travail de l'ingénieur

2. Introduction aux cycles de vie

3. Cascade

4. incrémental

5. Transformation.

6. Spirale

7. Conclusion sur les cycles de vie

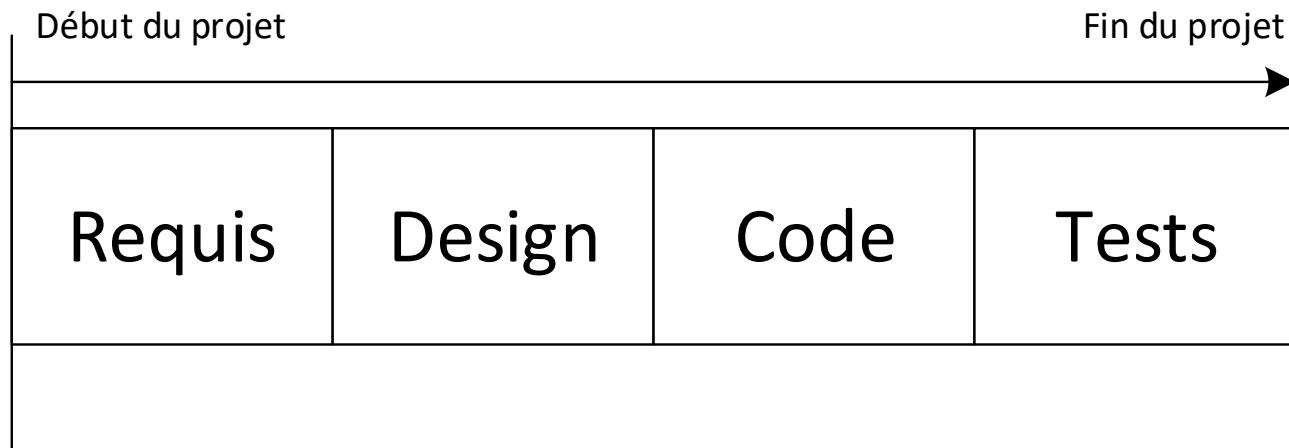
Cycles de vie de base

- Modèle cascade (waterfall), en V, W, ou X
- Modèle incrémental
- Modèle transformationnel
- Modèle spirale
- Etc.



Modèle cascade (waterfall)

- Présenté par W. Royce en 1970 comme un modèle commun, mais défectueux.
 - Quand même faisable **si toutes les informations requises sont disponibles au début du projet.**
- Avantage: Gestion « facile ».
 - Toutes les tâches se planifient (diagrammes de Gantt, réseaux de PERT).
 - Points de contrôle précis permettant de décider si on va de l'avant (*go/no go*).



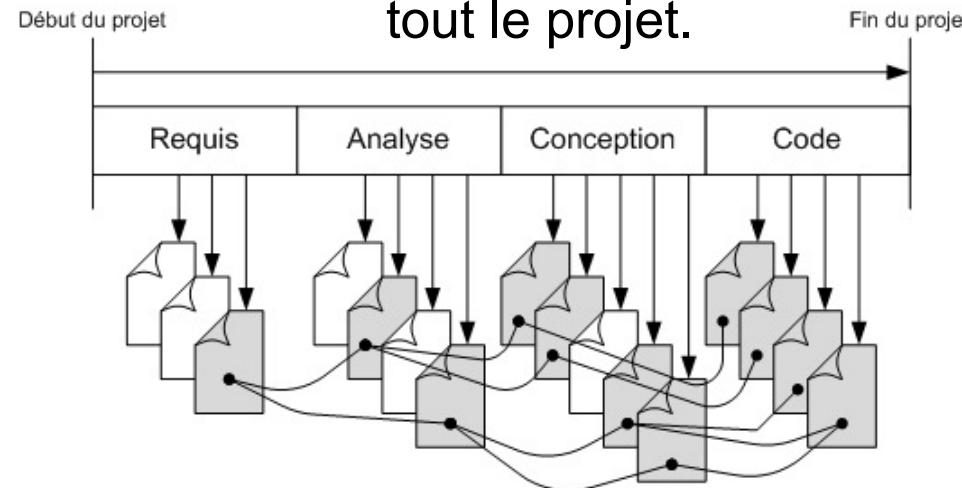
Modèle cascade (waterfall)

- Cascade peut être approprié lorsque:
 - Faible incertitude au niveau des besoins du client (« client sait ce qu'il veut »);
 - faible incertitude technique;
 - réalisation répétitive;
 - peu d'aspects novateurs;
 - faible participation du client/utilisateurs;
 - pas d'urgence de voir le produit fini.
- **Gestion entièrement prévisible à l'avance**
→ Requiert une gestion élaborée.

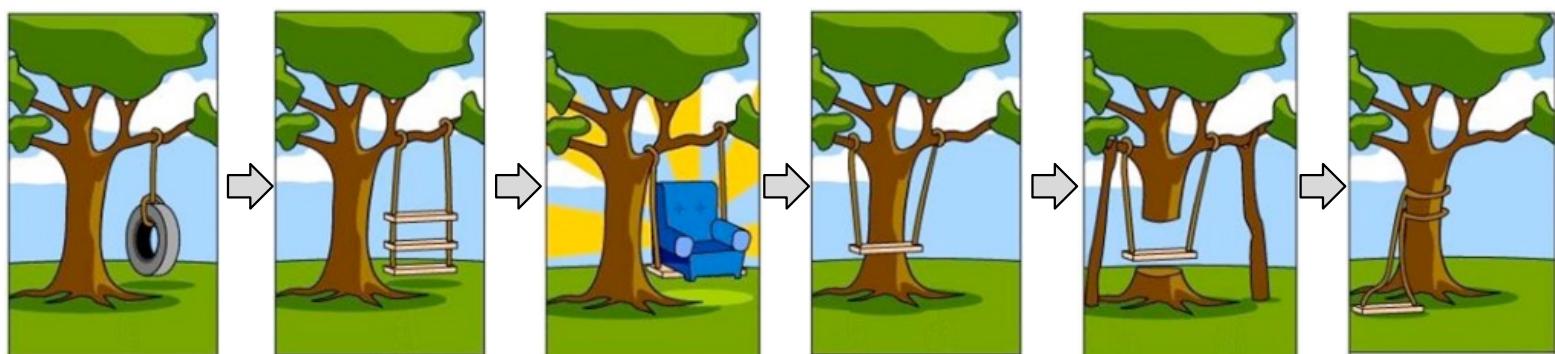
1. Le travail de l'ingénieur
2. Introduction aux cycles de vie
- 3. Cascade**
4. Incrémental
5. Transformation.
6. Spirale
7. Conclusion sur les cycles de vie

Faiblesses du modèle cascade (waterfall)

Un changement mineur peut avoir des répercussions à travers tout le projet.



La distance entre la vision initiale du client et le produit final implémenté peut être grande



Ce que le client voulait

Comment le client l'a expliqué

Ce que le marketing a vanté au client

Ce que les exigences ont décrit

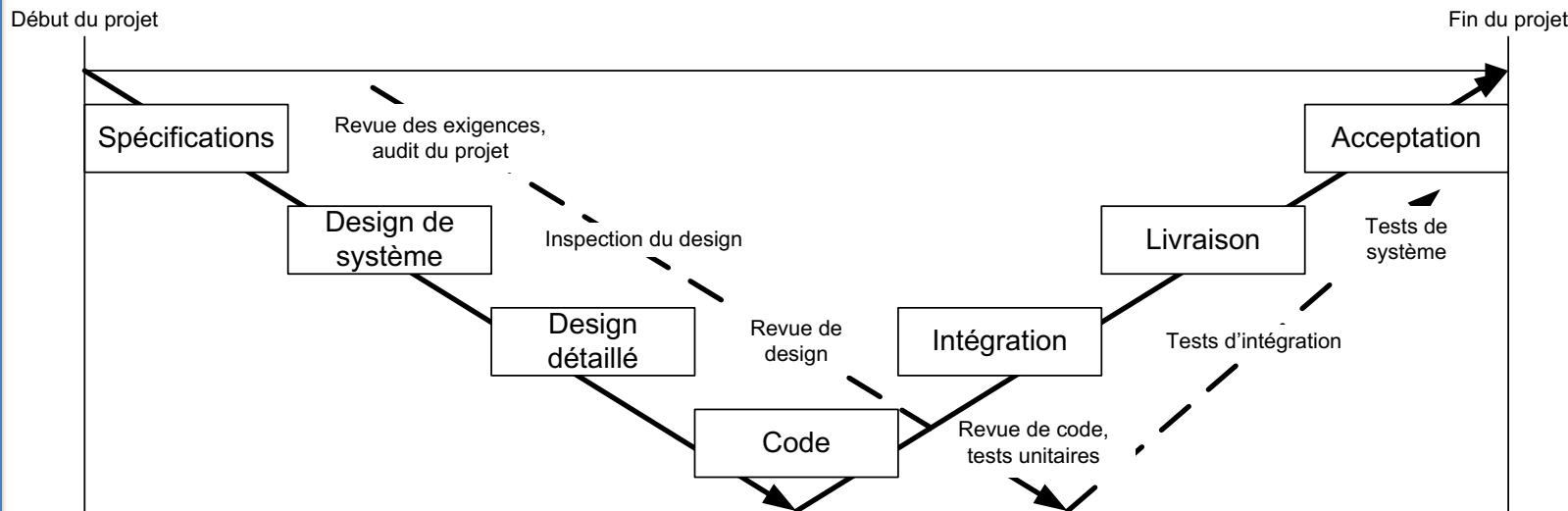
Ce que l'architecture a décrit

Ce qui a été programmé

1. Le travail de l'ingénieur
2. Introduction aux cycles de vie
- 3. Cascade**
4. Incrémental
5. Transformation.
6. Spirale
7. Conclusion sur les cycles de vie

Modèle V, W ou X

- V: Activités de développement.
- W: Ajout des activités d'assurance-qualité.



- Vision essentiellement cascade, avec revues et inspections.
- Les termes cascade, V, W et X sont souvent interchangés dans l'industrie.

1. Le travail de l'ingénieur

2. Introduction aux cycles de vie

3. Cascade

4. incrémental

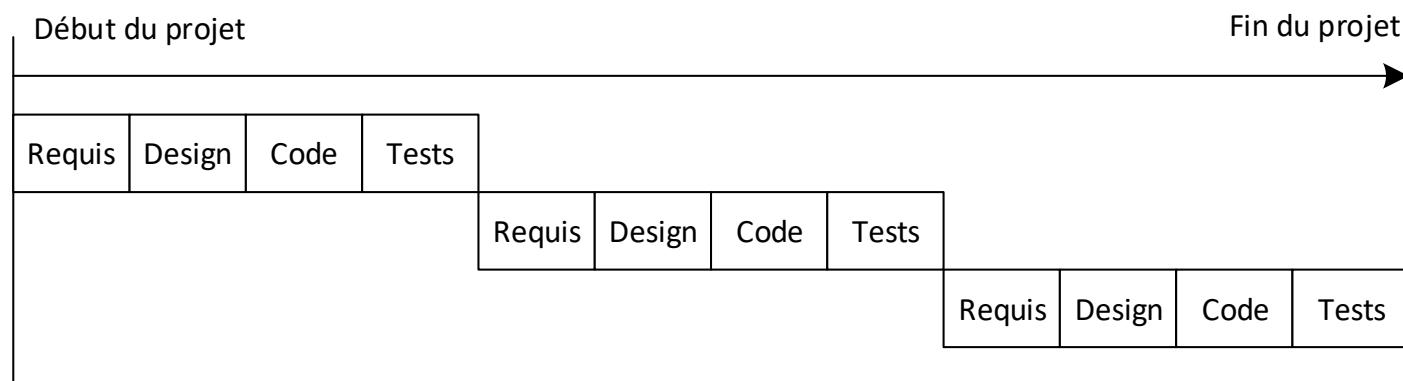
5. Transformation.

6. Spirale

7. Conclusion sur les cycles de vie

Modèle incrémental

- Itérations (cycles rapides) de production d'un produit fonctionnel, **chacune se concentrant sur une fraction des exigences.**
- Les itérations sont indépendantes les unes des autres.
- Itérations typiques de 2 à 6 semaines.
- Besoin d'un projet fractionnable où une base d'exigences est connue au préalable.



Section adaptée d'une présentation de M. Salman, S. Tyagi, B. Simmons et M. Quinn.

Modèle incrémental

- Avantages

- Plus de **flexibilité** : Moins coûteux de faire des changements.
- **Gestion du risque plus facile** : Une itération permet de passer à travers toutes les étapes de développement et d'identifier les problèmes.
- Chaque itération est un jalon (*milestone*) facile à gérer, comme une approche cascade.

- Inconvénients

- Les itérations sont rigides et ne permettent pas de retour en arrière → Sauf en créant de nouvelles exigences.
- Des **problèmes d'architecture** peuvent survenir parce que les requis ne sont pas tous connus au départ.
- Difficile de déterminer la fin du projet (*scope creep*).

1. Le travail de l'ingénieur

2. Introduction aux cycles de vie

3. Cascade

4. Incrémental

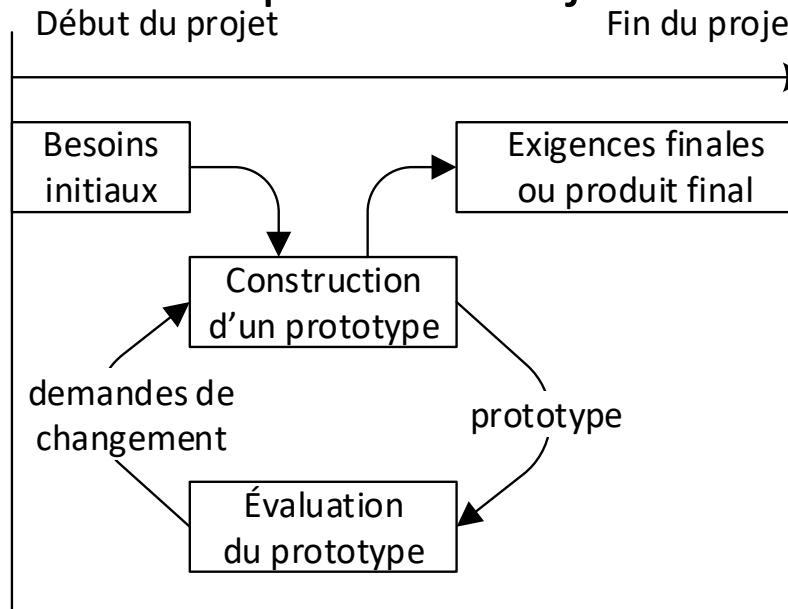
5. Transformation.

6. Spirale

7. Conclusion sur les cycles de vie

Modèle transformationnel/prototypage

- Prototypage évolutif
 - Prototype jetable (utilisé pour déterminer les exigences) ou non (utilisé comme produit final).
 - Toutes les parties sont développées simultanément sans planification initiale : Peut être néfaste sur la modularisation.
 - Itérations très rapides : Une journée à deux semaines.



- Processus populaire pour la conception centrée sur l'utilisateur (UCD).

1. Le travail de l'ingénieur

2. Introduction aux cycles de vie

3. Cascade

4. Incrémental

5. Transformation.

6. Spirale

7. Conclusion sur les cycles de vie

Modèle transformationnel/prototypage

- ## Avantages

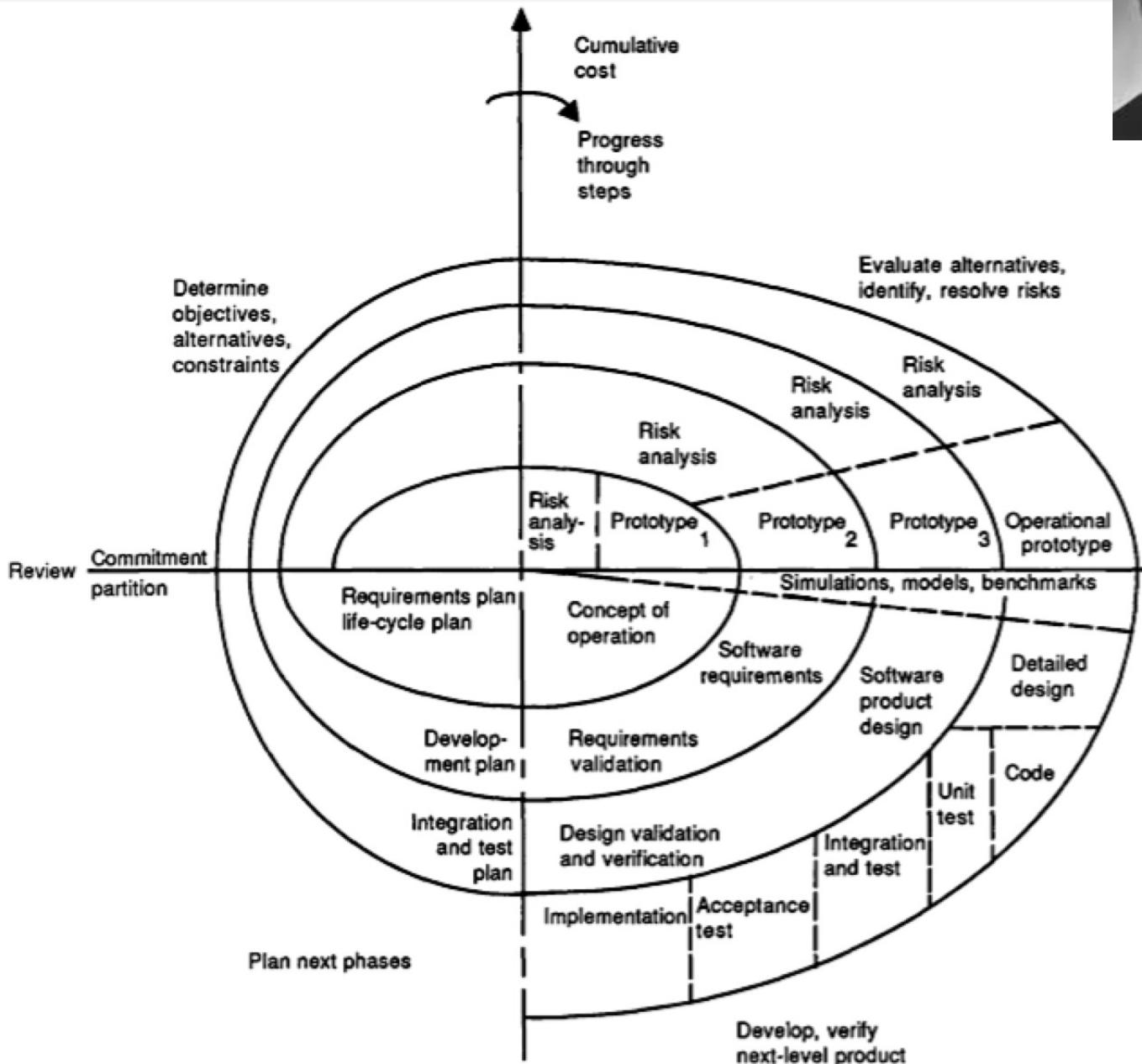
- Cycles très rapides : on peut faire rapidement des corrections.
- Feedback plus étayé : **plus facile d'évaluer un prototype que des documents papier**.

- ## Inconvénients

- Difficile à gérer : dépassement d'échéancier, de budgets, etc.
- *Gold-plating* : On peut détailler très longtemps une fonctionnalité qui finit à la poubelle.
- Réutilisation de prototype : qualité du code souvent déficiente.

1. Le travail de l'ingénieur
2. Introduction aux cycles de vie
3. Cascade
4. Incrémental
5. Transformation.
- 6. Spirale**
7. Conclusion sur les cycles de vie

Modèle spirale



1. Le travail de l'ingénieur

2. Introduction aux cycles de vie

3. Cascade

4. Incrémental

5. Transformation.

6. Spirale

7. Conclusion sur les cycles de vie

Modèle spirale

- Avantages
 - Analyse de **risque** très élaborée pour **prioriser** l'ordre des **cycles**.
 - **Générateur** de processus: peut utiliser n'importe quel cycle de vie (+ mélanger)
- Inconvénients
 - **Difficile à gérer** : dépassement d'échéanciers, de budgets, etc.
 - Processus très proche d'un projet particulier: difficile à réutiliser.

1. Le travail de l'ingénieur

2. Introduction aux cycles de vie

3. Cascade

4. Incrémental

5. Transformation.

6. Spirale

7. Conclusion sur les cycles de vie

Conclusion sur les cycles de vie

- Chaque approche possède des avantages et des inconvénients.
- Le cycle de vie approprié à utiliser dépend toujours du **contexte** du développement logiciel.
- L'essentiel du cours de LOG3000 est d'apprendre quels facteurs du contexte il faut observer, et quelles **solutions** existent pour chaque cas.



1. Le travail de l'ingénieur

2. Introduction aux cycles de vie

3. Cascade

4. incrémental

5. Transformation.

6. Spirale

7. Conclusion sur les cycles de vie

Conclusion sur les cycles de vie

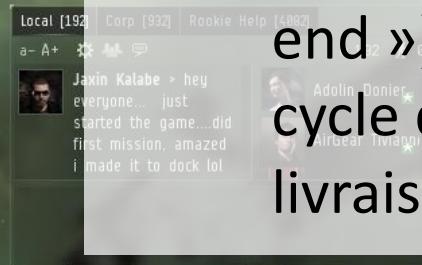
- Chaque approche possède des avantages et des inconvénients.
- L'approche à choisir dépend du contexte du projet.
 - Peu d'incertitude ou tout est connu → Cascade.
 - Projet avec une base d'exigences initiales facile à fractionner → incrémental.
 - Projet avec beaucoup d'incertitudes → Prototypage.
 - Projet complexe à haut risque → Spirale.





Exemple : CCP Games et le jeu en ligne Eve Online.

- Processus 1 : **L'interface** doit être changée rapidement, donc utilisation d'un **cycle de vie rapide** permettant de livrer des mises-à-jour à chaque mois.
- Processus 2 : Changer le **moteur** (« engine », « back-end ») est très **risqué**, alors il vaut mieux prendre un **cycle de vie avec plus d'étapes de vérification** et 1-2 livraisons par année.



1. Valeurs et
principes Agile

2. Exemples de
processus Agile

3. Critique du
débat
discipliné/Agile

LOG3000

Processus du génie logiciel

La philosophie Agile

Historique des processus

- 1968: Conférence de l'OTAN en Allemagne de l'ouest : Le logiciel est en crise!
- Le développement logiciel:
 - Dépasse les budgets et les échéanciers prévus;
 - produit des logiciels qui ne répondent pas aux besoins;
 - produit des logiciels qui tombent toujours en panne;
 - produit des logiciels qui ne sont pas maintenables ...
- Problème: Le développement est chaotique.
- Solution: Processus ! Sauf que ...

... 33 ans plus tard, on est écœurés
d'être traités comme des machines !



Manifeste pour le développement Agile de logiciels

Nous découvrons comment mieux développer des logiciels
par la pratique et en aidant les autres à le faire.

Ces expériences nous ont amenés à valoriser :

- Les individus et leurs interactions plus que les processus et les outils**
- Des logiciels opérationnels plus qu'une documentation exhaustive**
- La collaboration avec les clients plus que la négociation contractuelle**
- L'adaptation au changement plus que le suivi d'un plan**

Nous reconnaissons la valeur des seconds éléments,
mais privilégiions les premiers.

Principes de l'Agile

- Notre plus haute priorité est de satisfaire le client en livrant rapidement et régulièrement des fonctionnalités à **grande valeur ajoutée**.
- **Accueillez positivement les changements** de besoins, même tard dans le projet. Les processus Agiles exploitent le changement pour donner un avantage compétitif au client.
- **Livrez fréquemment** un logiciel opérationnel avec des cycles de quelques semaines à quelques mois et une préférence pour les plus courts.
- Les utilisateurs ou leurs représentants et les développeurs doivent **travailler ensemble quotidiennement** tout au long du projet.

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

Principes de l'Agile

- Réalisez les projets avec des **personnes motivées**. Fournissez-leur l'environnement et le soutien dont ils ont besoin et faites-leur **confiance** pour atteindre les objectifs fixés.
- La méthode la plus simple et la plus efficace pour transmettre de l'information à l'équipe de développement et à l'intérieur de celle-ci est le **dialogue en face à face**.
- Un logiciel opérationnel est la principale mesure d'avancement.
- Les processus Agiles encouragent un rythme de **développement soutenable**. Ensemble, les commanditaires, les développeurs et les utilisateurs devraient être capables de maintenir indéfiniment un rythme constant.

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

Principes de l'Agile

- Une attention continue à l'excellence technique et à une bonne conception renforce l'Agilité.
- La simplicité – c'est-à-dire l'art de **minimiser la quantité de travail inutile** – est essentielle.
- Les meilleures architectures, spécifications et conceptions **émergent** d'équipes **auto-organisées**.
- À intervalles réguliers, l'équipe réfléchit aux moyens de devenir plus efficace, puis règle et modifie son comportement en conséquence.

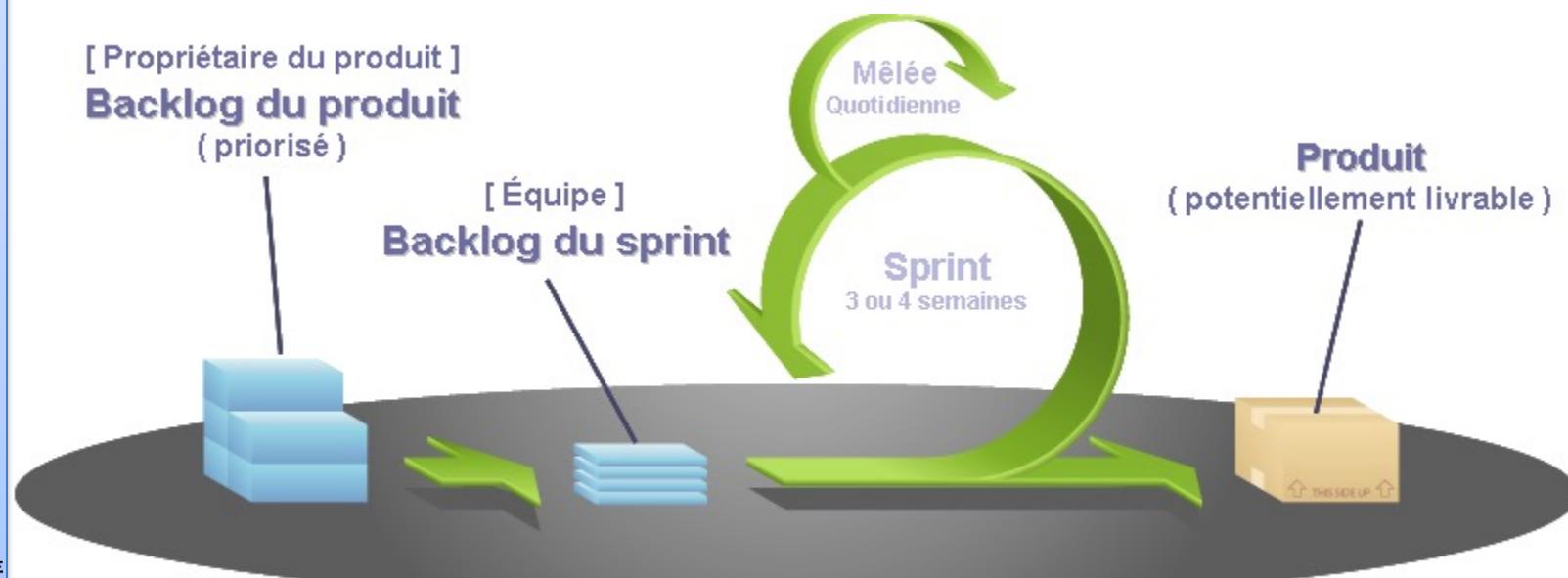
<http://agilemanifesto.org/iso/fr/principles.html>

Exemple: Les principes de l'XP

- L'eXtreme Programming (XP) est une des premières approche Agile essayée.
- Principes :
 - *Test Driven Development (TDD)* : On écrit les tests unitaires en premier.
 - *Pair Programming (PP)* : Tout le code est écrit avec deux développeurs sur un ordinateur.
 - *You Ain't Gonna Need It (YAGNI)* : On écrit juste assez de code pour faire passer les tests.
 - *Refactoring* : Le code écrit est ensuite révisé pour satisfaire aux critères de qualité.
 - Le client doit être assis en permanence avec les développeurs.
 - Itérations courtes (1-2 semaines)
 - Exigences traitées par priorités, changements possibles pendant itération

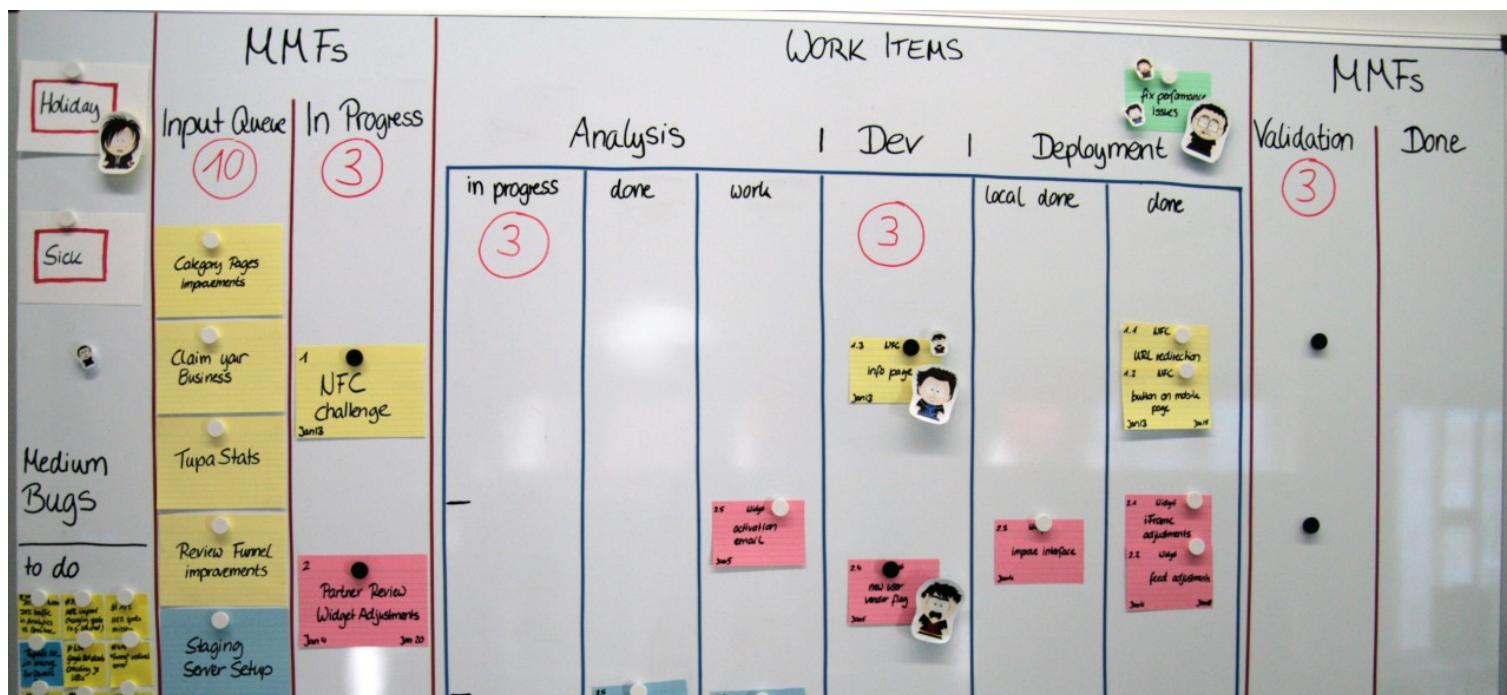
Exemple: Les principes du SCRUM

- Backlog: Ensemble d'exigences, trié par scrum master (« value-driven »).
- Sprint: Itération de 2 à 4 semaines (portée fixe).
- Mêlée quotidienne (« *daily scrum* »): Réunion debout où l'on répond aux trois questions suivantes:
 - Qu'as-tu fait depuis hier?
 - Que penses-tu faire jusqu'à demain?
 - Y a-t-il des obstacles à ton travail?



Exemple: Les principes du Kanban

- Populaire ces dernières années.
- Du japonais pour ‘enseigne’ ou ‘panneau d’affichage’.
- La planification complète du travail est visible pour toute l’équipe.
- Le développeur prend une tâche disponible sur le tableau et la remet une fois complétée.



Exemple: Les principes du Lean

- Inspiré du domaine manufacturier:
Éliminer tout travail inutile.
- Inutile signifie:
 - Tout ce qui n'est pas du code lié à une fonctionnalité désirée par le client;
 - bureaucratie (« decide as late as possible »);
 - mauvaises communications, tests insuffisants, exigences floues ...

Alberta

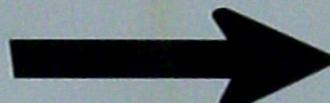
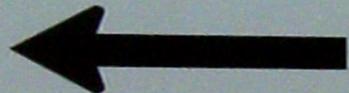
14

Alberta

14

Discipliné

Agile



Problèmes du discipliné

- Problème du « *write-only artefact* »
 - On écrit beaucoup de rapports qui ne sont finalement jamais lus.
- Problème du « *big up-front design* »
 - On crée une structure complexe pour un logiciel qu'on ne comprend pas parfaitement.
 - Dans les faits, beaucoup d'éléments critiques ne deviennent évidents qu'après avoir mis la main à la pâte.



Problèmes du discipliné

- Problème de la communication indirecte,
 - Communiquer à travers un rapport (ou des Powerpoint!) n'est jamais aussi bon que communiquer **face-à-face**.
- Problème de **totalitarisme**,
 - Un besoin absolu de tout contrôler, même quand le risque est minime.
- Problème de **sclérose**.
 - « *All Hail the Almighty Process!* »
 - Impossibilité de déroger au processus, même quand la situation le demande.

Problèmes de l'Agile

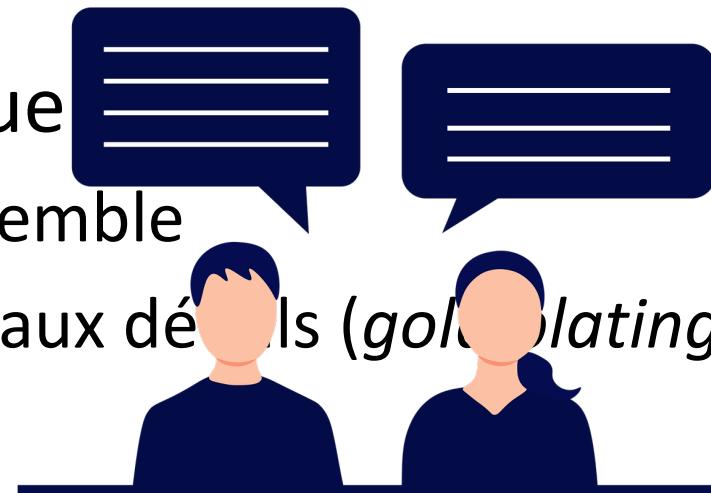
- Itérations trop courtes,
 - Pas capable de faire tout le travail prévu, débordement sur les itérations suivantes,
 - Manque de temps pour tout faire (conception, tests, communications ...),
 - Itérations mal planifiées.
- Dépendances aux rétroactions,
 - Rétroactions arrivent trop tard
 - Décisions souvent basées sur non sur des données concrètes



Problèmes de l'Agile

- Dépendance aux communications,
 - Certaines personnes ne participent pas au dialogue face-à-face,
 - Le client promet mais ne s'implique pas,
 - Besoin d'éduquer l'équipe, la direction et les clients.

- Manque d'une vue
 - Pas de vue d'ensemble
 - Trop d'attention aux détails (*gold plating*).



Qui a raison?

- Il y a beaucoup de vendeurs et vendeuses qui prêchent pour leur église...
- Dans les faits, les deux points de vue sont complémentaires:
 - Choisir l'approche selon le **contexte**;
 - on peut créer des processus **hybrides**!



1. Modélisation
de Processus

2. Du cycles de
vie au processus

LOG3000

Processus du génie logiciel

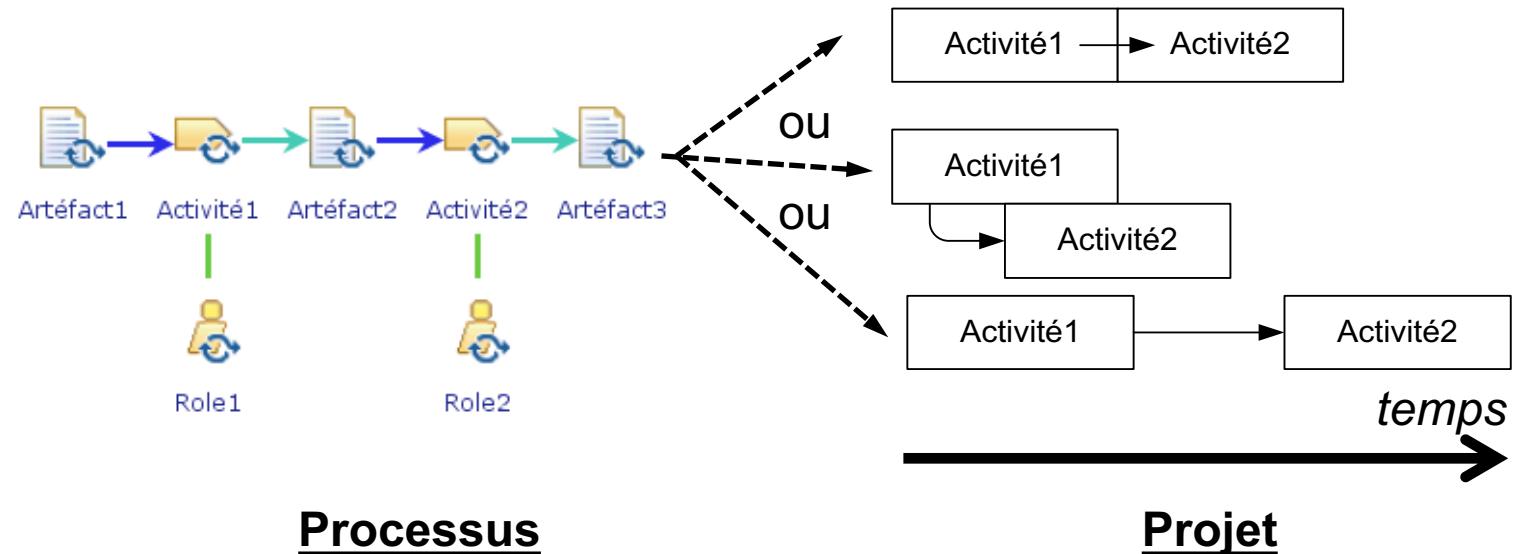
Du cycle de vie au processus

Modélisation de Processus

1. Modélisation de Processus

2. Du cycles de vie au processus

- Éléments du processus:
 - Activité: Élément de travail.
 - Artéfact: Élément d'information.
 - Rôle: Compétences et responsabilités.
- Le processus est un ensemble d'activités ***partiellement ordonnées***, indépendant des notions de temps et de ressources :

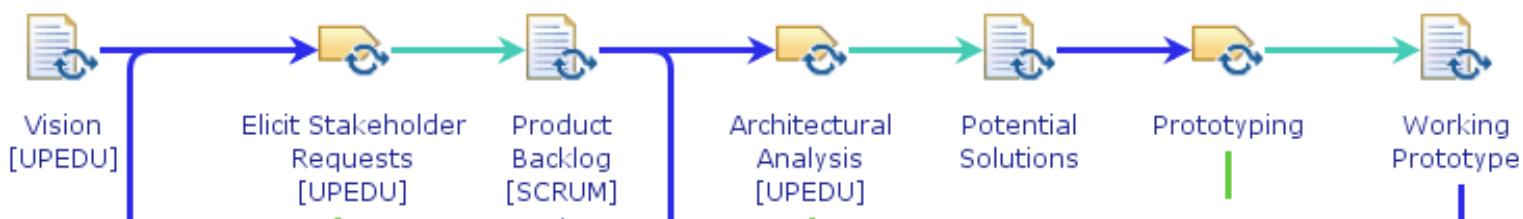


Processus ≠ Projet

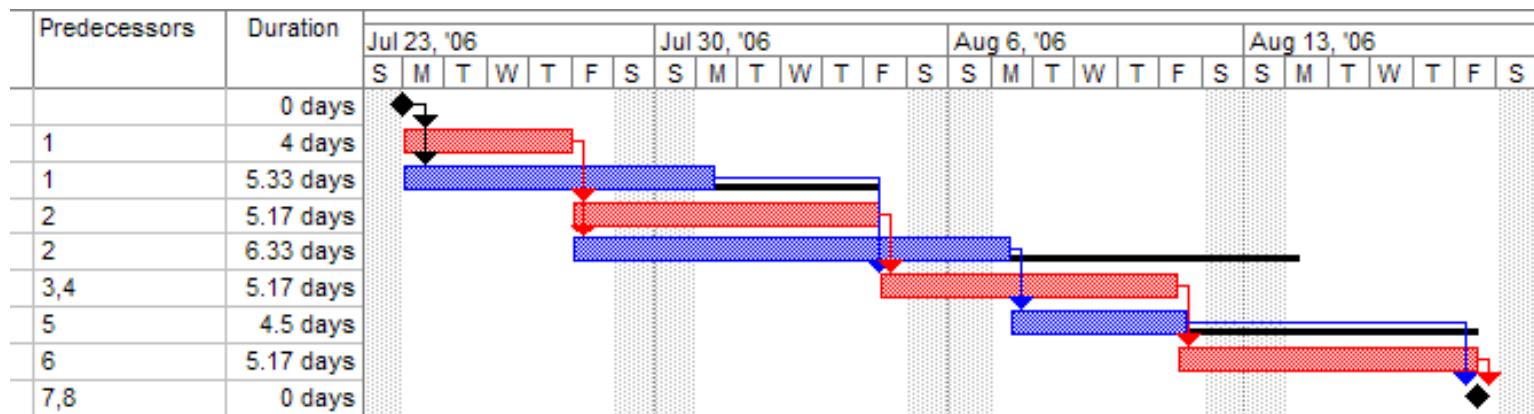
1. Modélisation de Processus

2. Du cycles de vie au processus

- **Processus** : Décrit comment l'information est transformée durant le développement du logiciel.



- **Projet** : Planifie et assigne les ressources durant le développement logiciel.



Vocabulaire du cours LOG3000

- **Processus (process)** : Ensemble partiellement ordonné d'activités, dont le but est de transformer les entrées en sorties selon un objectif défini.
- **Activité** : Élément de travail du processus.
- **Artéfact** : Élément d'information du processus.
- **Rôle** : Ensemble de compétences et de responsabilités nécessaires pour faire une activité.
- **Discipline (discipline)** : Ensemble de pratiques et activités reliées, représentant un domaine.
- **Phase (phase)** : Division d'un cycle de vie, dont la fin représente généralement un jalon important.

Processus

Cycle de
vie

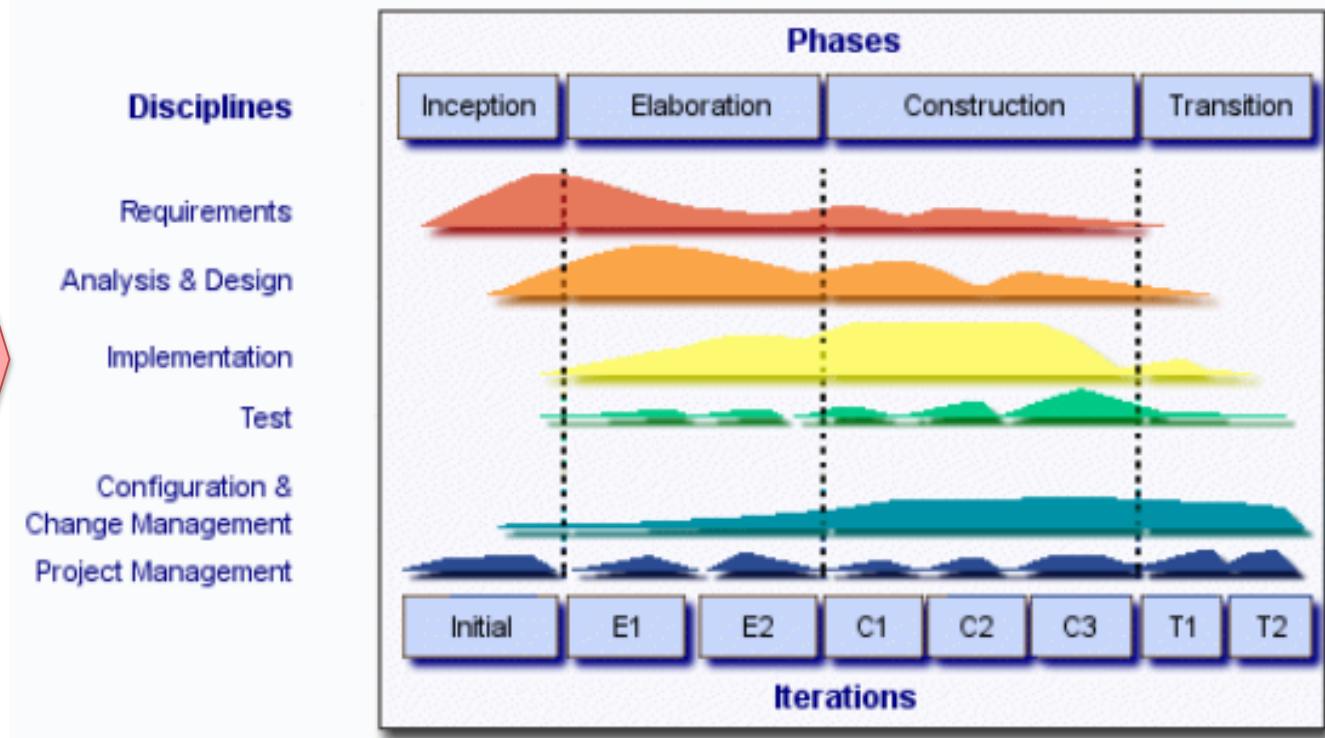
Remarque sur le vocabulaire

- Attention à ne pas mélanger le terme « phase » et le terme « discipline ».

Phases

(lié au cycle de vie)

Disciplines
(lié au contenu)



Vocabulaire du cours LOG3000

- **Processus (process)** : Ensemble partiellement ordonné d'activités, dont le but est de transformer les entrées en sorties selon un objectif défini.
- **Activité** : Élément de travail du processus.
- **Artéfact** : Élément d'information du processus.
- **Rôle** : Ensemble de compétences et de responsabilités nécessaires pour faire une activité.
- **Discipline (discipline)** : Ensemble de pratiques et activités reliées, représentant un domaine.
- **Phase (phase)** : Division d'un cycle de vie, dont la fin représente généralement un jalon important.
- **Jalon (milestone)** : Le jalon marque la complétion d'une étape importante dans le travail. Le jalon est généralement lié à la complétion d'un ou de plusieurs artéfacts.
- **Tâche (task)** : Élément de travail dans un plan de projet ayant des ressources assignées.

Processus

Cycle de
vie

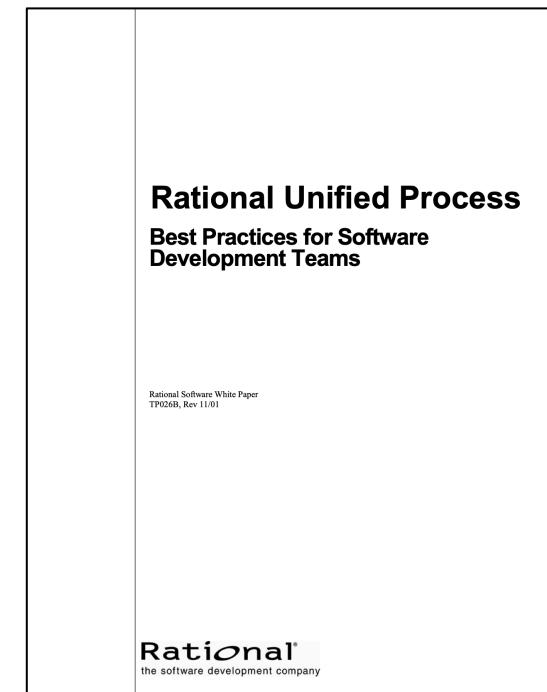
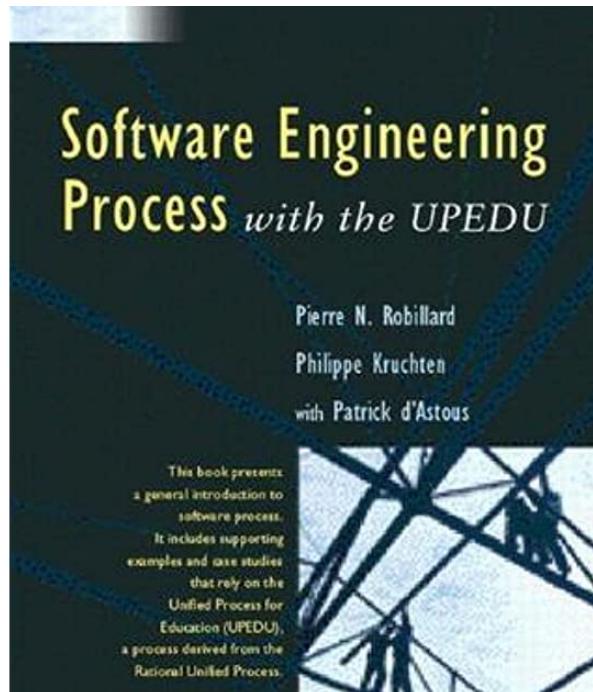
Projet

Unified Process for EDUcation (UPEDU)

1. Modélisation
de Processus

2. Du cycles de
vie au processus

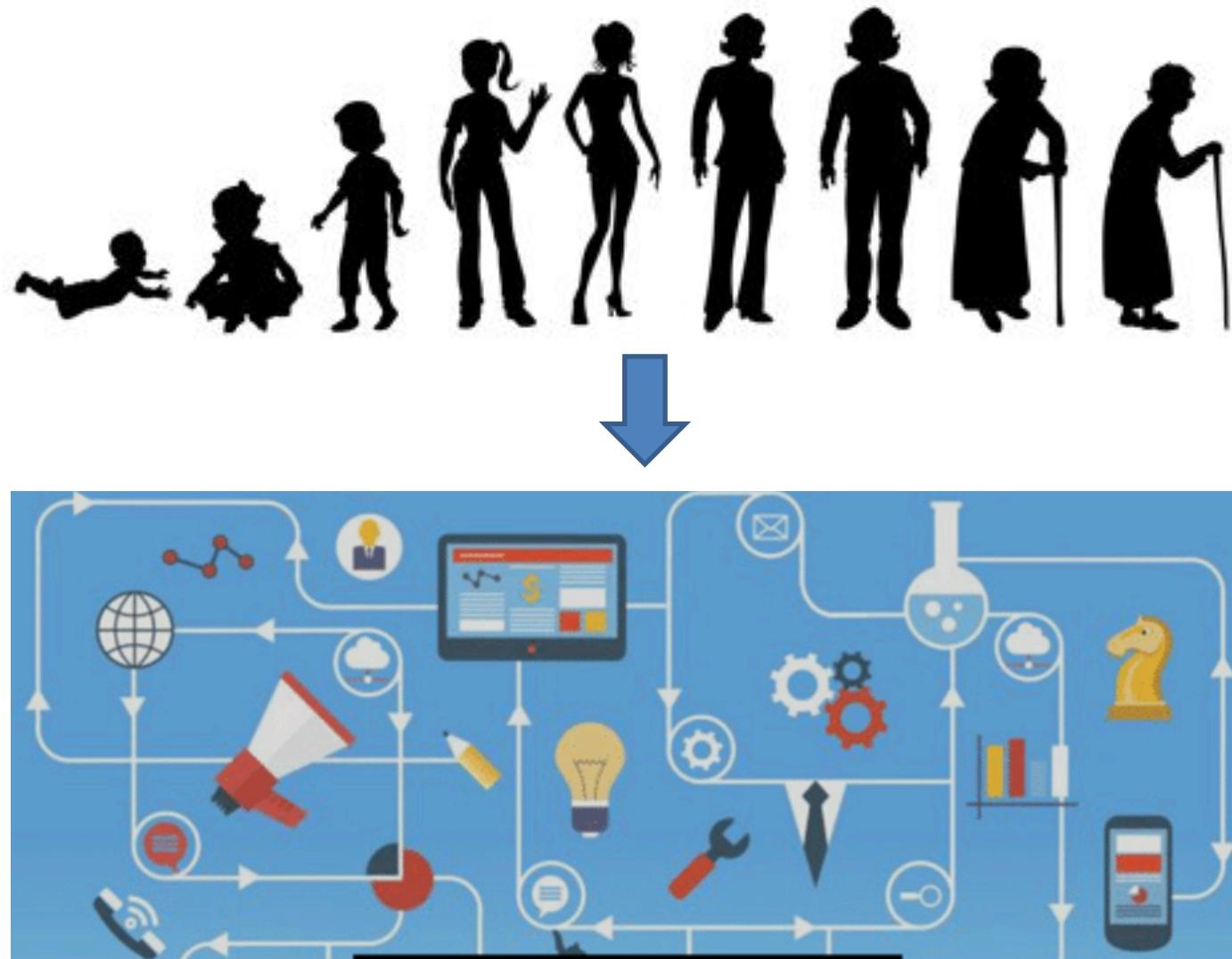
- L'UPEDU est une version simplifiée du RUP (*Rational Unified Process*).
 - <https://web.archive.org/web/20180224032122/http://upedu.org/>



Transformer le cycle de vie en processus

1. Modélisation
de Processus

2. Du cycles de
vie au processus

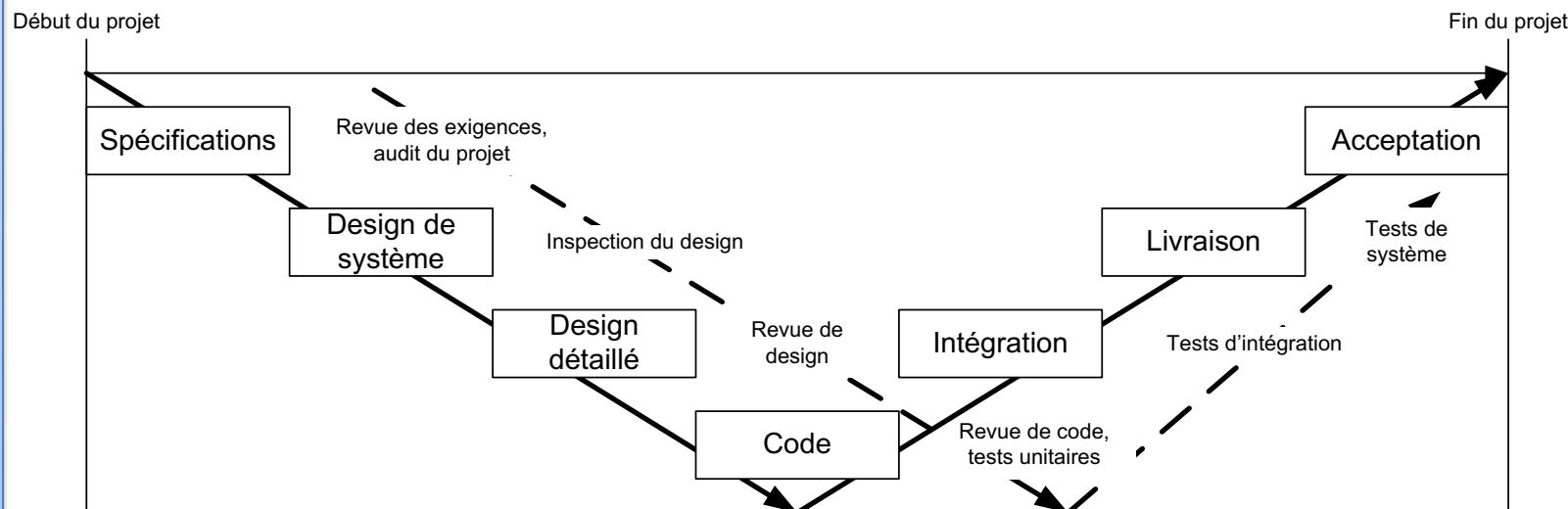


<https://www.softwaretestinghelp.com/software-development-life-cycle-sdlc/>

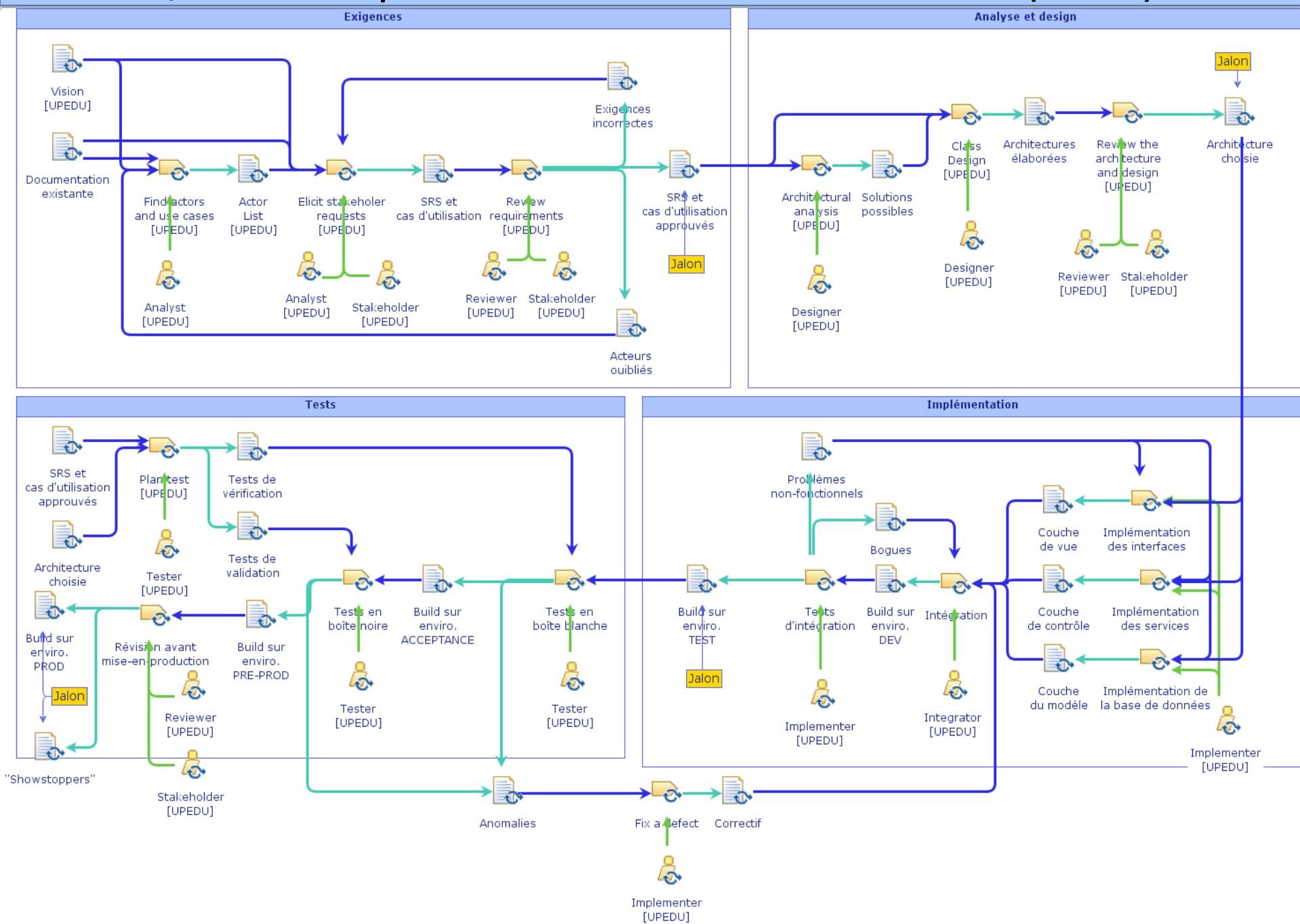
Cycles de vie en V, W ou X

1. Modélisation de Processus
2. Du cycles de vie au processus

- Éléments clés pour le processus:
 - On termine tout le travail sur une phase avant de passer à la suivante (cascade).
 - On ne revient jamais à une phase terminée.
 - On révise le travail fait avant de passer à la phase suivante (V, W ou X).



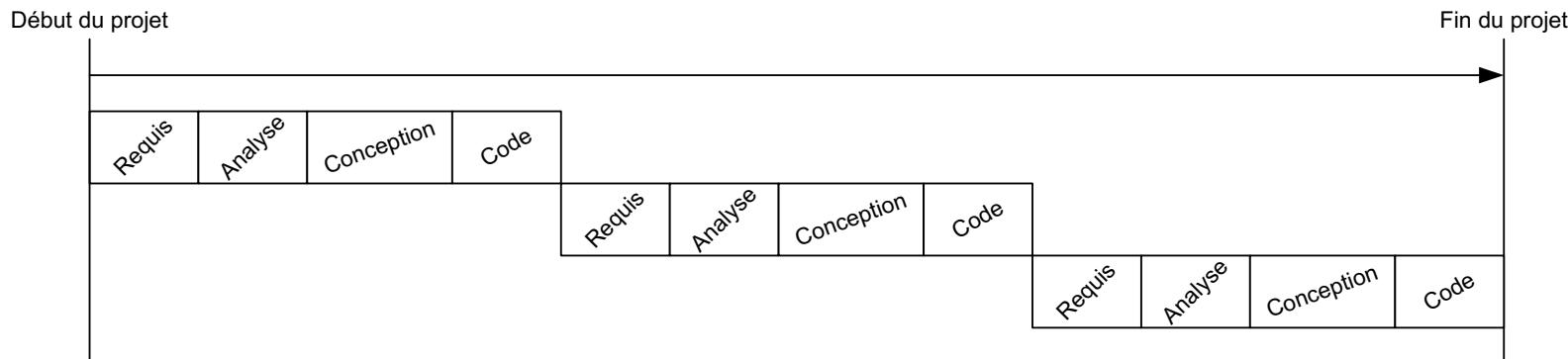
V, W ou X : processus observé en industrie (2014)



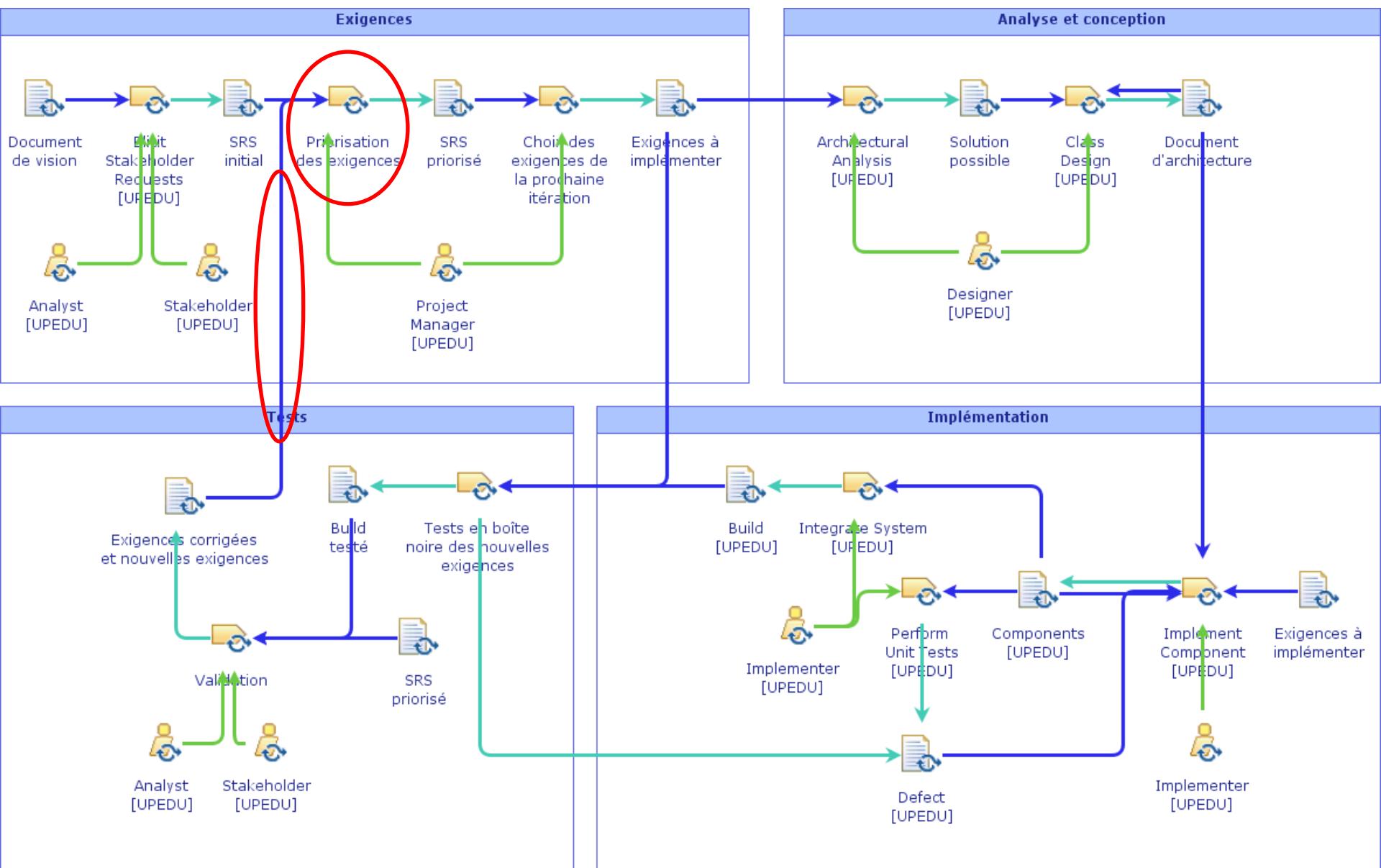
Cycle de vie incrémental

1. Modélisation de Processus
2. Du cycles de vie au processus

- Éléments clés pour le processus:
 - Itérations complètes (design, code, tests, etc.).
 - Chaque itération construit une partie du produit.



Incrémental simple

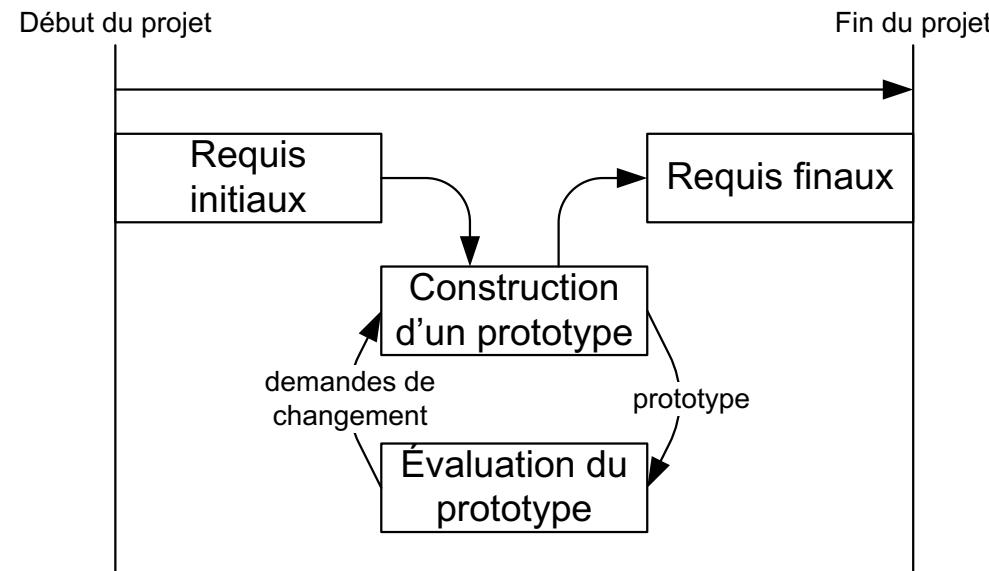


Cycles de vie transformationnels

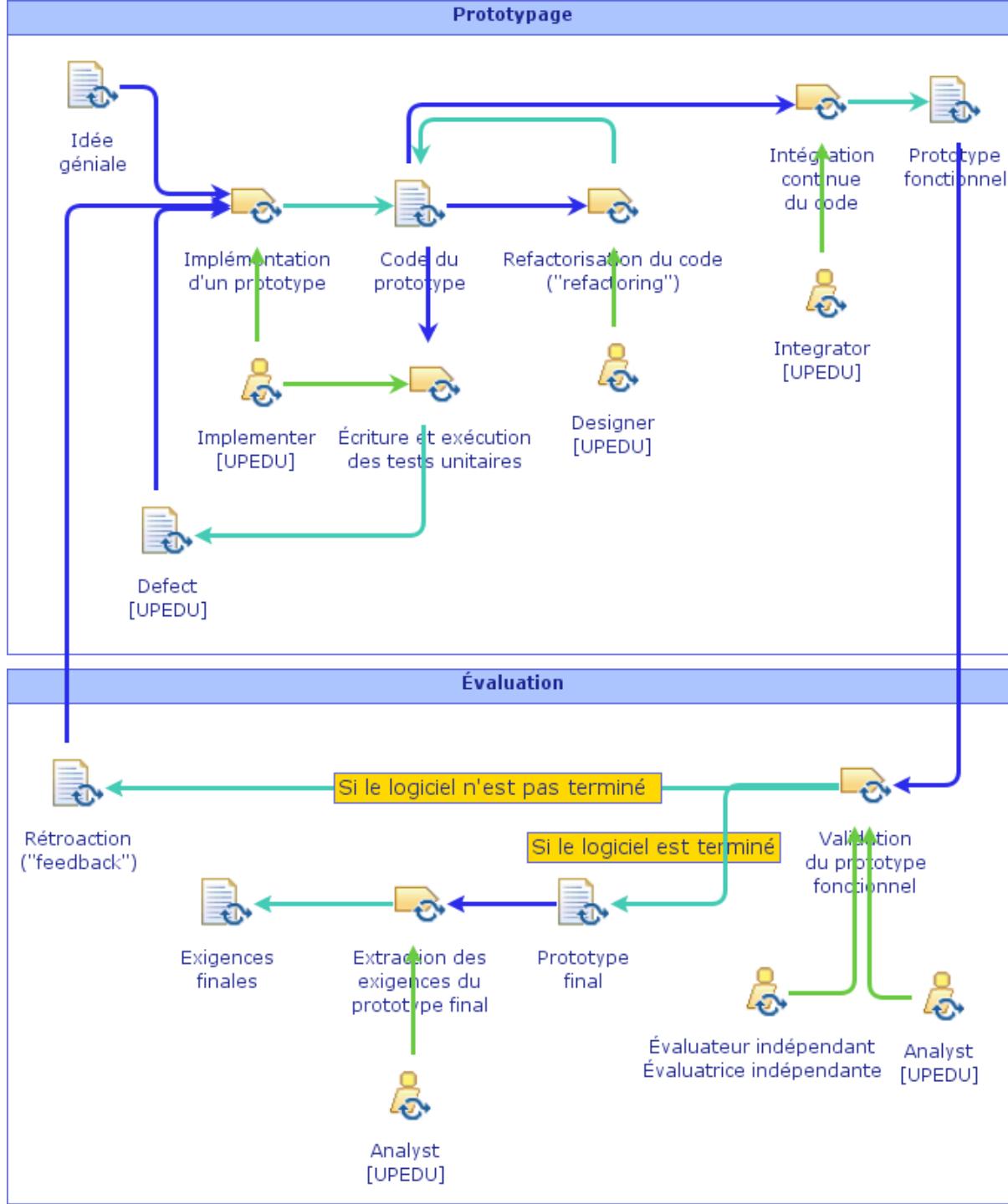
1. Modélisation
de Processus

2. Du cycles de
vie au processus

- Éléments clés pour le processus:
 - Les besoins des client-e-s ne sont pas clairs, les hypothèses ne sont pas vérifiées → On obtient nos réponses à mesure qu'on construit et qu'on évalue les prototypes.
 - Transformation de prototype.



Transformationnel / Prototypage

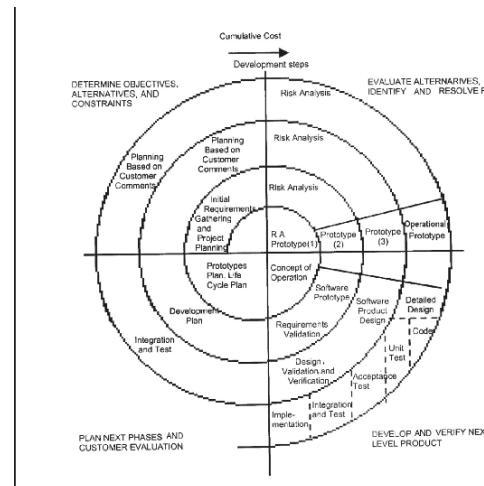


Cycles de vie spirales

1. Modélisation
de Processus

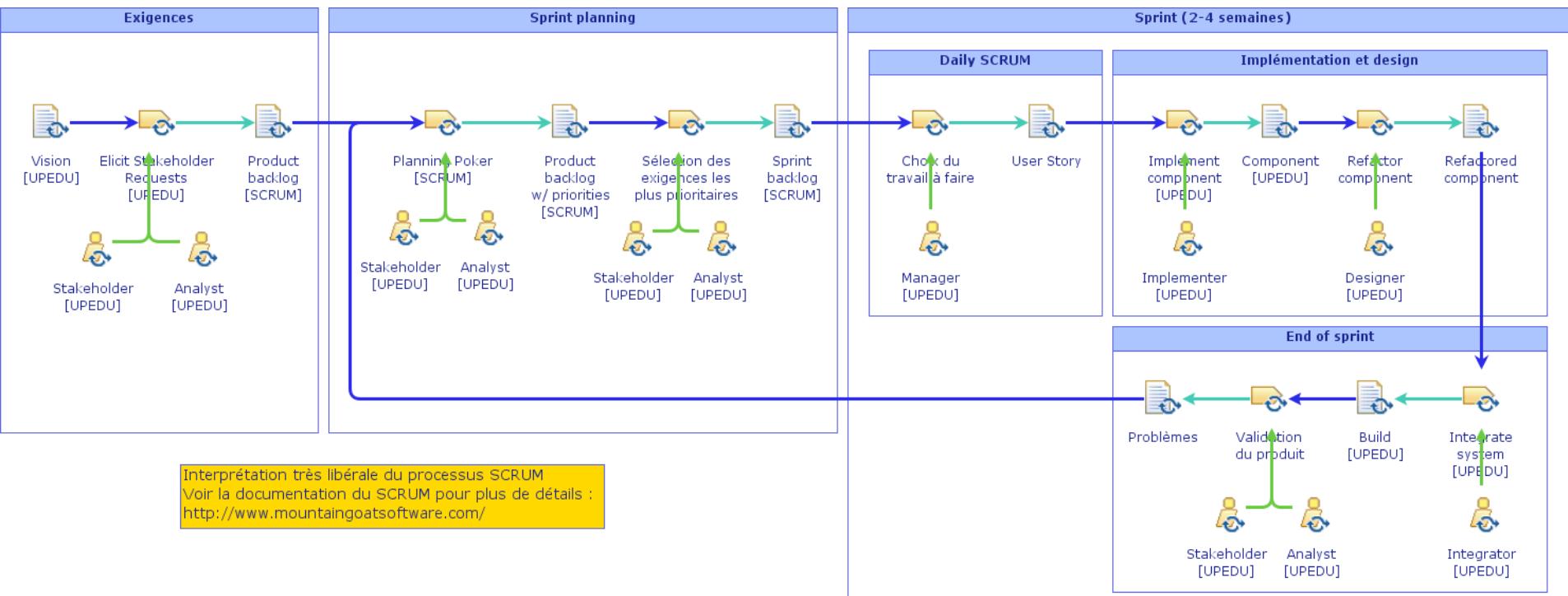
2. Du cycles de
vie au processus

- Éléments clés pour le processus:
 - Mélange de cascade et de transformationnel:
On passe sur toutes les phases du processus, mais sans compléter le travail à chaque fois.
 - Chaque itération amène de nouvelles activités qui raffinent ce qui a été fait précédemment.
 - Crée un processus très spécifique qui s'applique à peu près uniquement pour un seul projet.



Agile??

L'approche du SCRUM

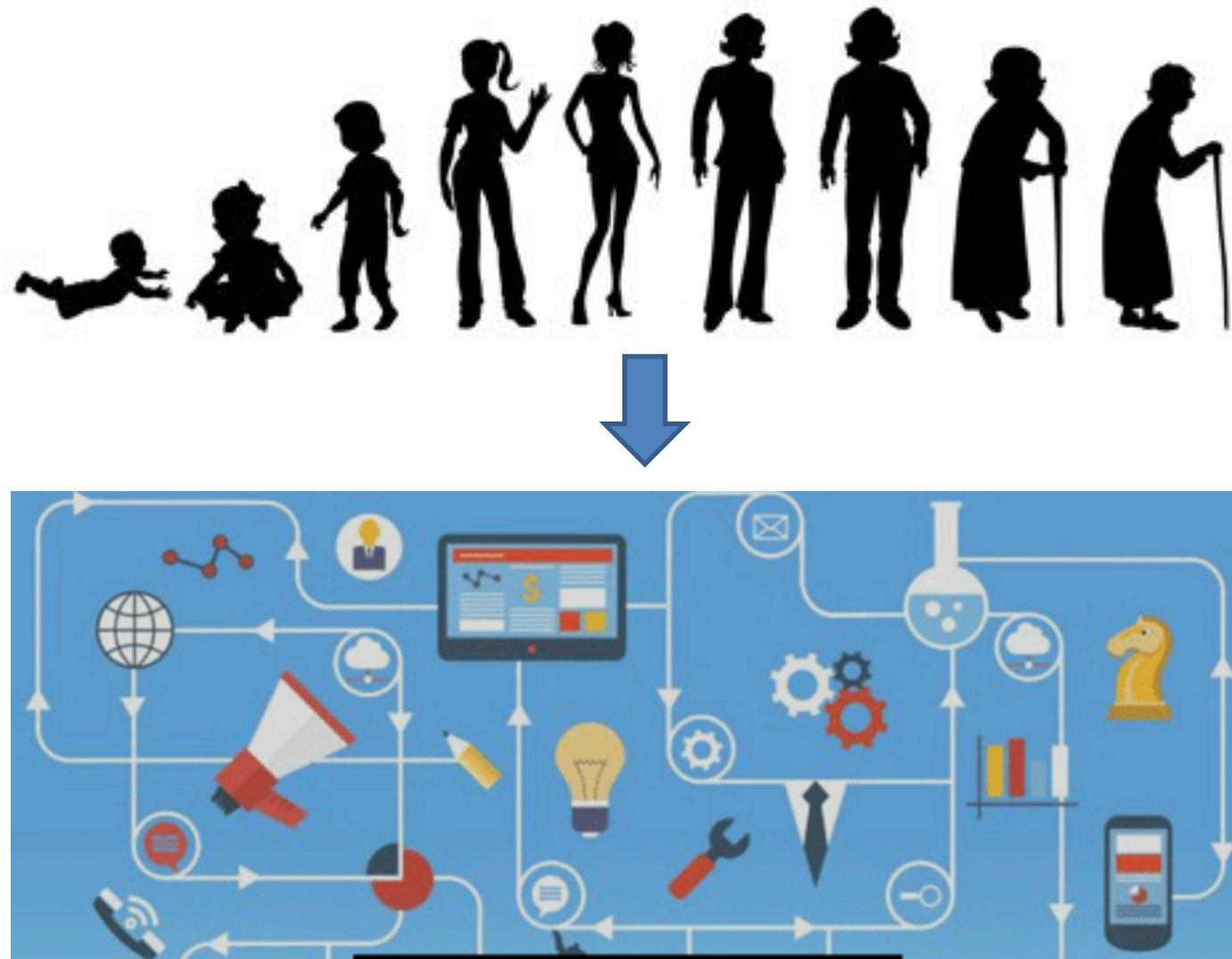


- Daily SCRUM : Réunion debout de 15 minutes sur les trois points suivants :
 - **Qu'est-ce que tu as fait hier ?**
 - **Qu'est-ce que tu vas faire aujourd'hui ?**
 - **Qu'est-ce qui bloque ton avancement ?**
- Sprint : Le SCRUM se limite seulement aux activités de gestion. Il n'y a aucune description de comment le travail doit se faire durant le « Sprint ».

Transformer le cycle de vie en processus

1. Modélisation
de Processus

2. Du cycles de
vie au processus



<https://www.softwaretestinghelp.com/software-development-life-cycle-sdlc/>

LOG3000

Processus du génie logiciel

1. Position du discipline
2. Approche de l'UPEDU
3. Autres approches de saisie d'exigences

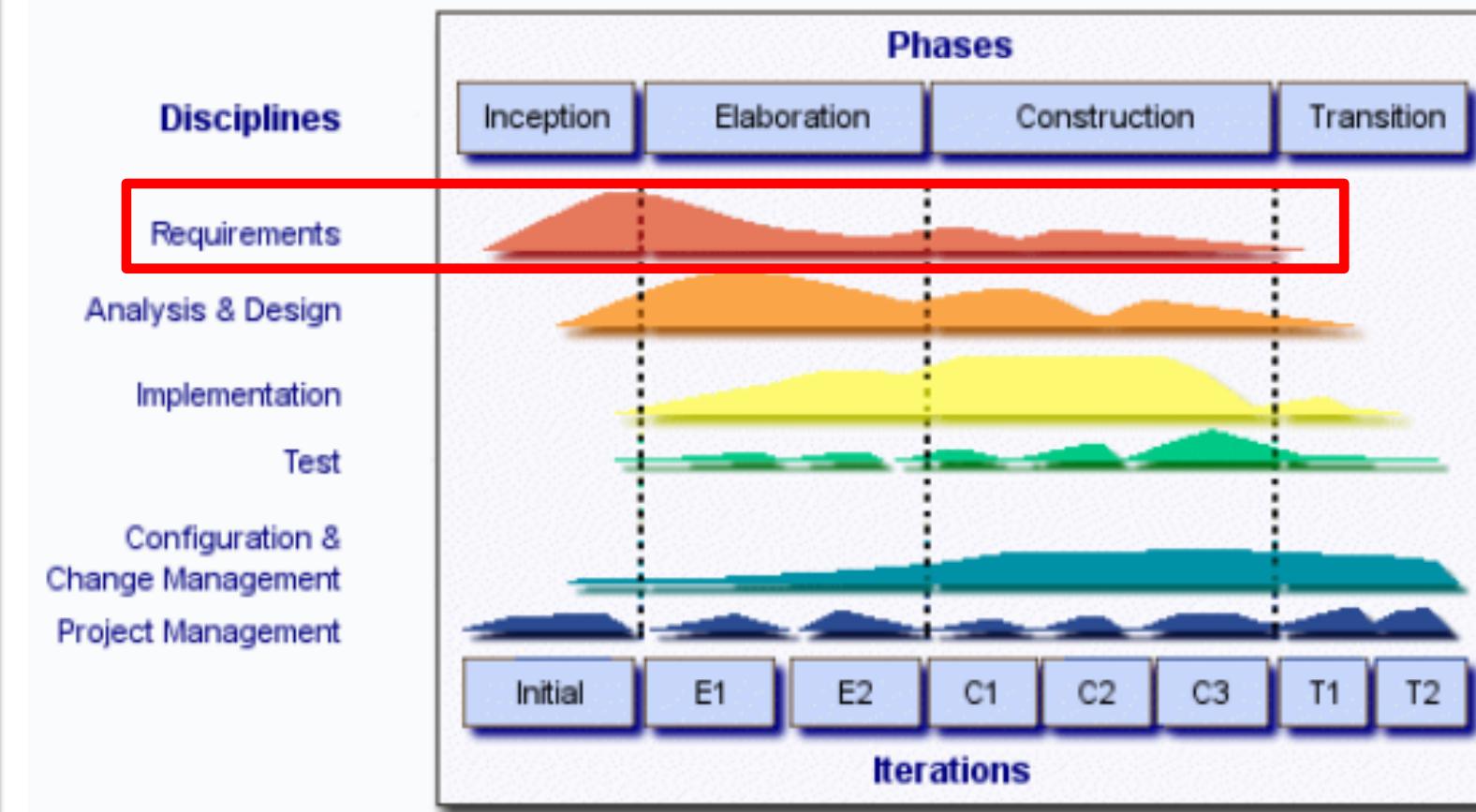
Discipline des requis

Position du discipline des requis

1. Position du discipline

2. Approche de l'UPEDU

3. Autres approches de saisie d'exigences



- La discipline des requis implique la **saisie** des exigences, mais aussi la **mise-à-jour** de celles-ci!

Position du discipline des requis

1. Position du discipline
2. Approche de l'UPEDU
3. Autres approches de saisie d'exigences

	Discipline	Artéfact principal
Ingénierie	Requis	Cas d'utilisation
	Analyse et conception	Architecture et modèle de conception (ex.: diagrammes de l'UML)
	Implémentation	Programmes (ex.: code)
	Tests	Plan de test et cas de test
Gestion	Gestion de projet	Diagramme de Gantt
	Gestion de configuration	Référentiel (ex.: SVN, Git)

Position du discipline des requis

- Objectif du discipline:
Minimiser l'impact des changements
 - Saisir et analyser efficacement les **besoins**,
 - Obtenir un accord (contrat) avec les client-e-s,
 - **Modéliser les interactions** entre utilisateur-e-s et système.
 - Établir un référentiel (ex.: SVN, Git) et un processus de contrôle des modifications.
 - Maintenir la **träçabilité** des exigences en aval et en amont.
 - Effectuer des **revues** des exigences:
Inspection, walkthrough, revue formelle.

1. Position du discipline

2. Approche de l'UPEDU

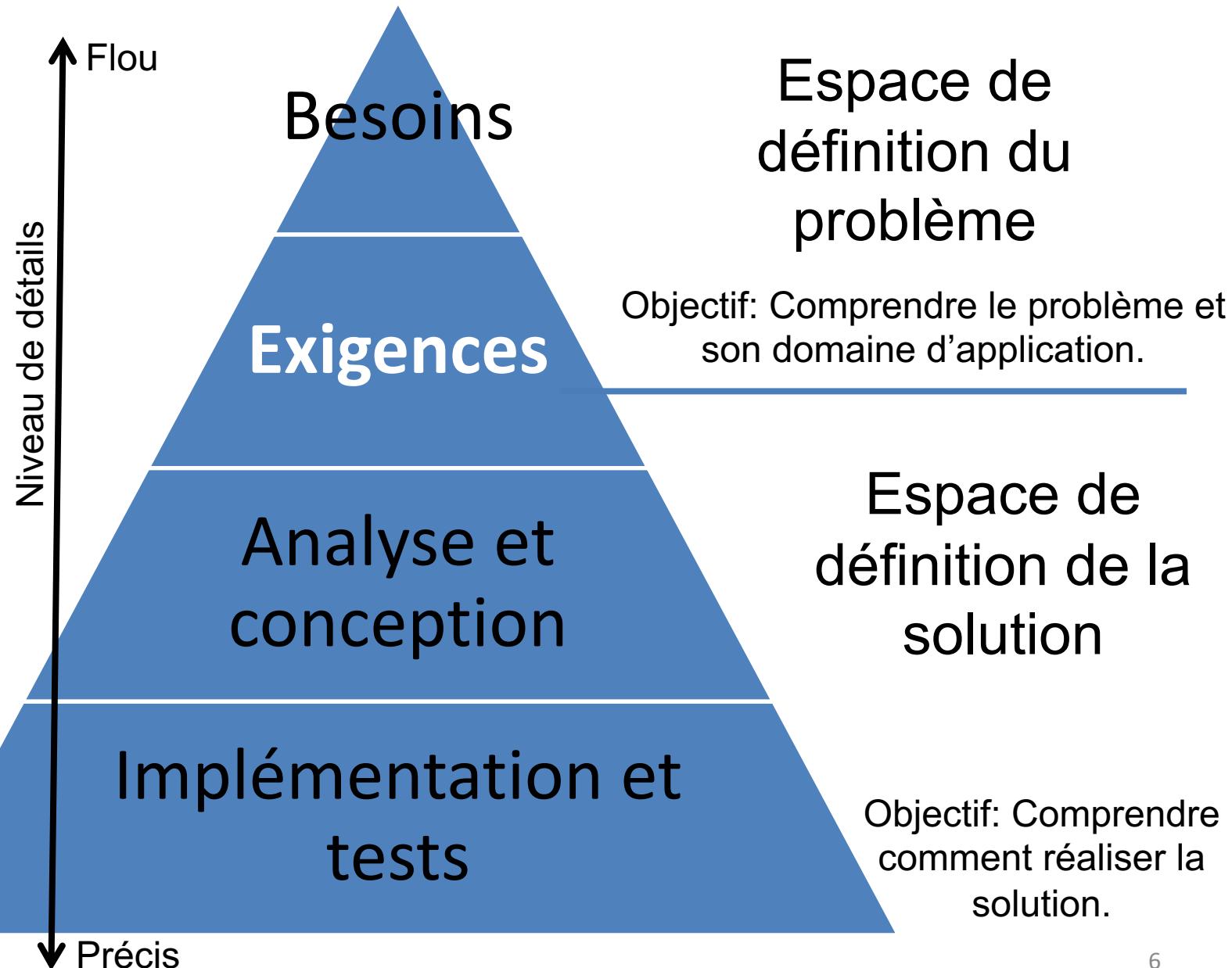
3. Autres approches de saisie d'exigences

Exigence ≠ Besoin

- Exigence : Expression d'un **besoin documenté** sur ce que le produit logiciel devrait être ou devrait faire.
- Le **besoin** est de haut niveau et flou.
 - Ex.: Le système sera facile à utiliser. Document de vision
- **L'exigence** est de plus bas niveau et elle est plus formelle (et vérifiable!).
 - Ex.: Chaque fonctionnalité du système doit avoir une info-bulle (*tooltip*). SRS / cas d'utilisation

Progression de l'information durant le développement

1. Position du discipline
2. Approche de l'UPEDU
3. Autres approches de saisie d'exigences



Relation avec les autres disciplines

1. Position du discipline

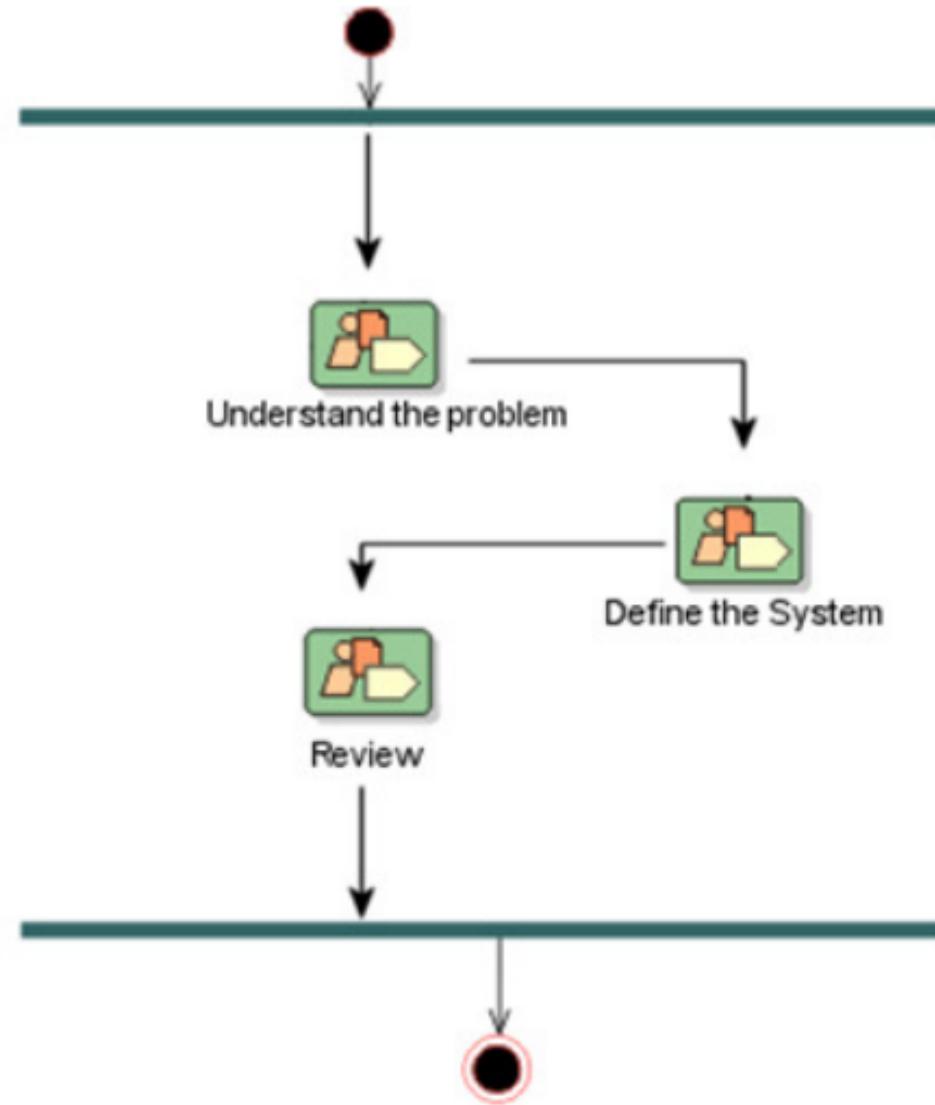
2. Approche de l'UPEDU

3. Autres approches de saisie d'exigences

Discipline	Les exigences servent de base pour ...
Analyse et conception	Cas d'utilisation (analyse) Paquetages et modules (conception)
Implémentation	Dans les applications critiques, chaque ligne de code doit être liée à une exigence
Test	Tests de vérification et validation
Gestion de projet	Planification des itérations et des tâches, définition des jalons, etc.
Gestion de configuration	Structure du référentiel, définition des branches, etc.

Discipline des requis: Approche de l'UPEDU

- Trois ensembles d'activités principales:



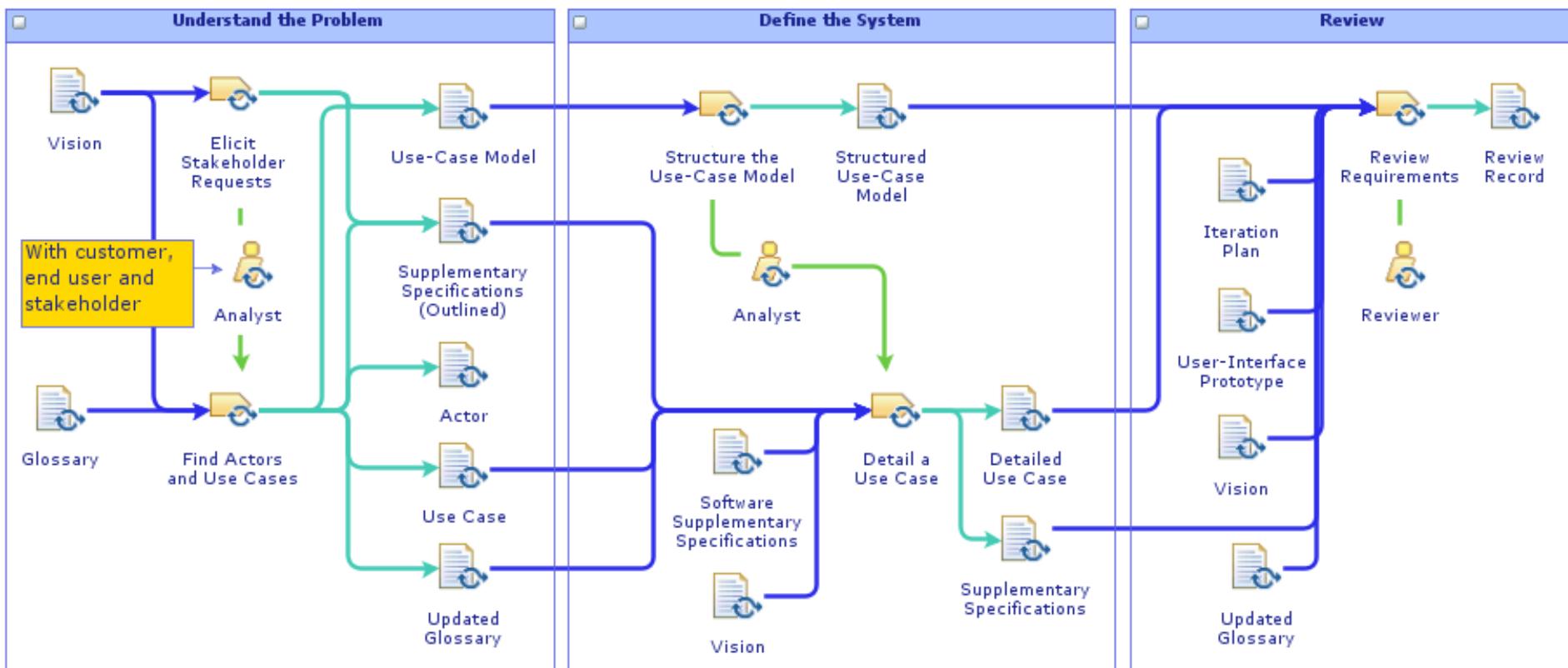
1. Position du discipline

2. Approche de l'UPEDU

3. Autres approches de saisie d'exigences

Discipline des requis: Approche de l'UPEDU

- Cinq activités, deux rôles et beaucoup d'artéfacts

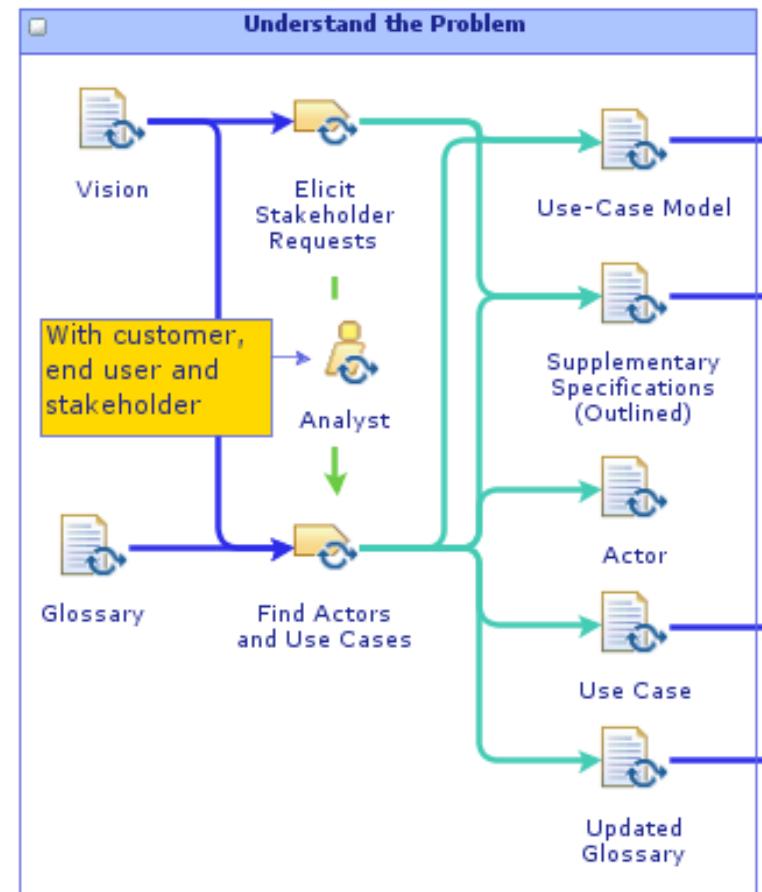


Discipline des requis: Approche de l'UPEDU

1. Position du discipline
2. Approche de l'UPEDU
3. Autres approches de saisie d'exigences

Understand the Problem

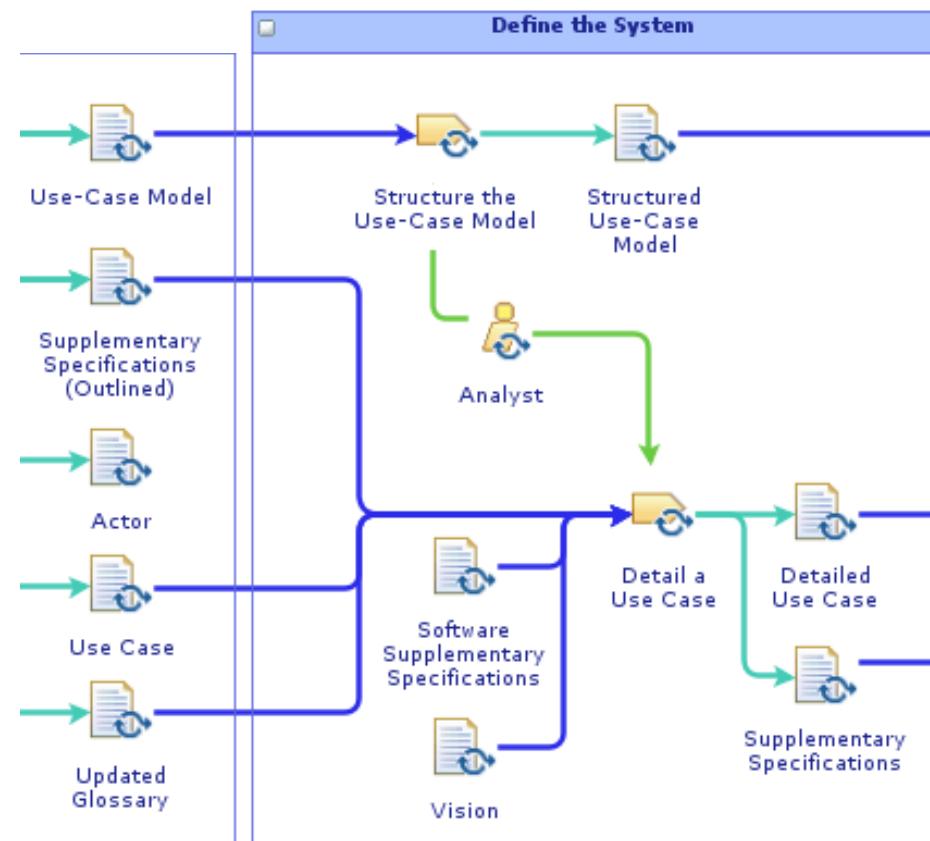
- **Elicit Stakeholder Requests**
Requests: Rencontrer les parties prenantes pour saisir leurs exigences.
- **Find Actors and Use Cases:** Définir les frontières du système (portée, *scope*).



Discipline des requis: Approche de l'UPEDU

Define the System

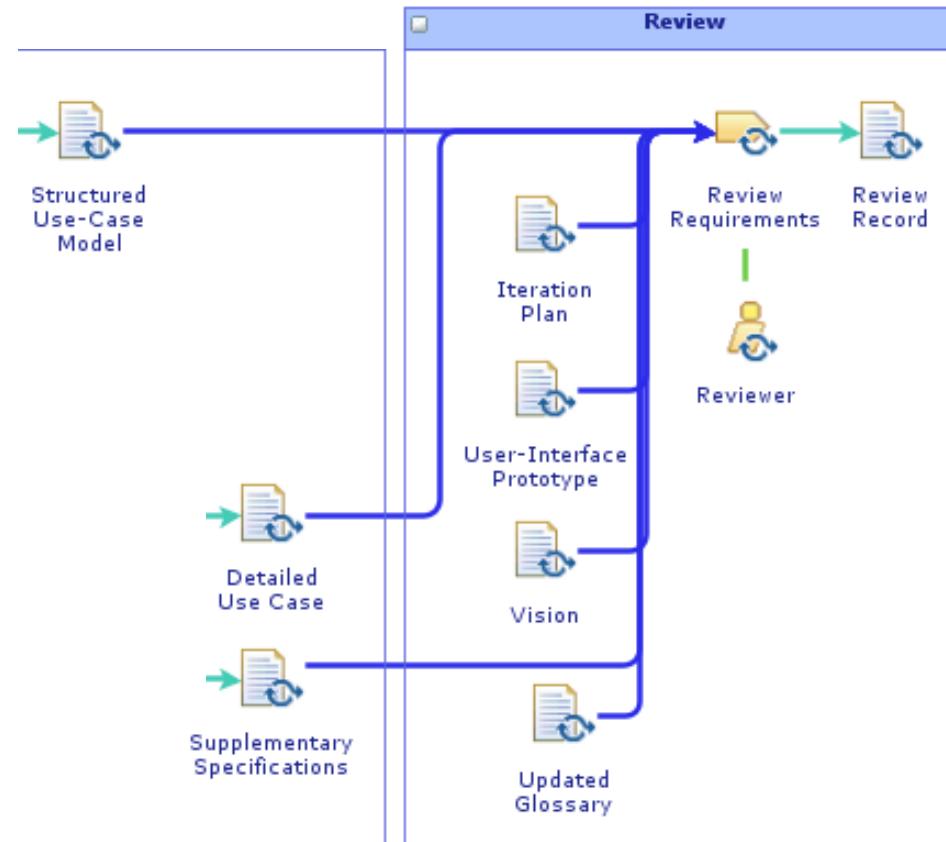
- **Structure the Use Case Model:** Définir les relations entre les cas d'utilisation.
- **Detail a Use Case:** Détailler le flot des cas d'utilisation avec pré-conditions, alternatives, etc.



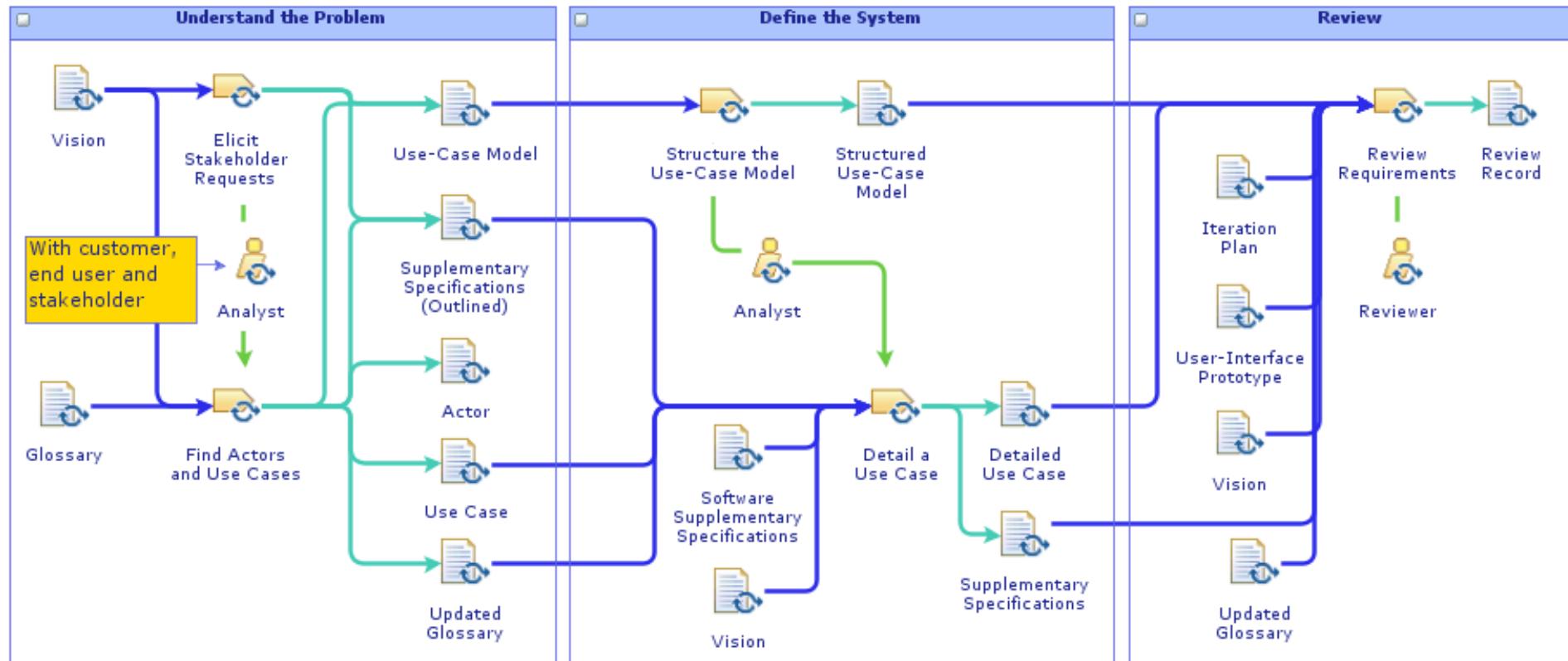
Discipline des requis: Approche de l'UPEDU

Review

– **Review Requirements:** Faire des réunions régulières avec les parties prenantes pour évaluer les documents produits.



Discipline des requis: Approche de l'UPEDU



But wait, there's more !

- C'est l'approche de l'UPEDU; il existe beaucoup d'autres approches!
- Parce que les approches traditionnelles ne sont pas toujours suffisantes...

1. Position du discipline

2. Approche de l'UPEDU

3. Autres approches de saisie d'exigences

The System

www.systemcomic.com

WHEN I SEE POOR DESIGN CHOICES



I used to think
“what a poor designer!”



Now I don't.



“The client must be a dick.”



Approches traditionnelles de saisie d'exigences

- Rencontre avec les parties prenantes
 - Clients, utilisateurs, études de marché, etc.
 - Réunions, interviews, questionnaires et sondages.
- Analyse de documents existants
 - Études de marché existantes, analyse de produits similaires, données de vente, etc.
- Introspection et remue-ménages (*brainstorming*).
 - Souvent inefficace, donc moins populaire.

Certains éléments proviennent d'une présentation de Steve Easterbrook (University of Toronto)

Autres approches de saisie d'exigences

1. Position du discipline

2. Approche de l'UPEDU

3. Autres approches de saisie d'exigences

- Analyse de tâche, du processus d'affaire.
- Observations participatives ou non-participative (*shadowing*) du travail.
- Raffinement de prototype.
- Avec un représentant du utilisateur participant au développement logiciel.
- Recherche de demandes de changement provenant des utilisateurs actuels.
- Etc.

Certains éléments proviennent d'une présentation de Steve Easterbrook (University of Toronto)

LOG3000

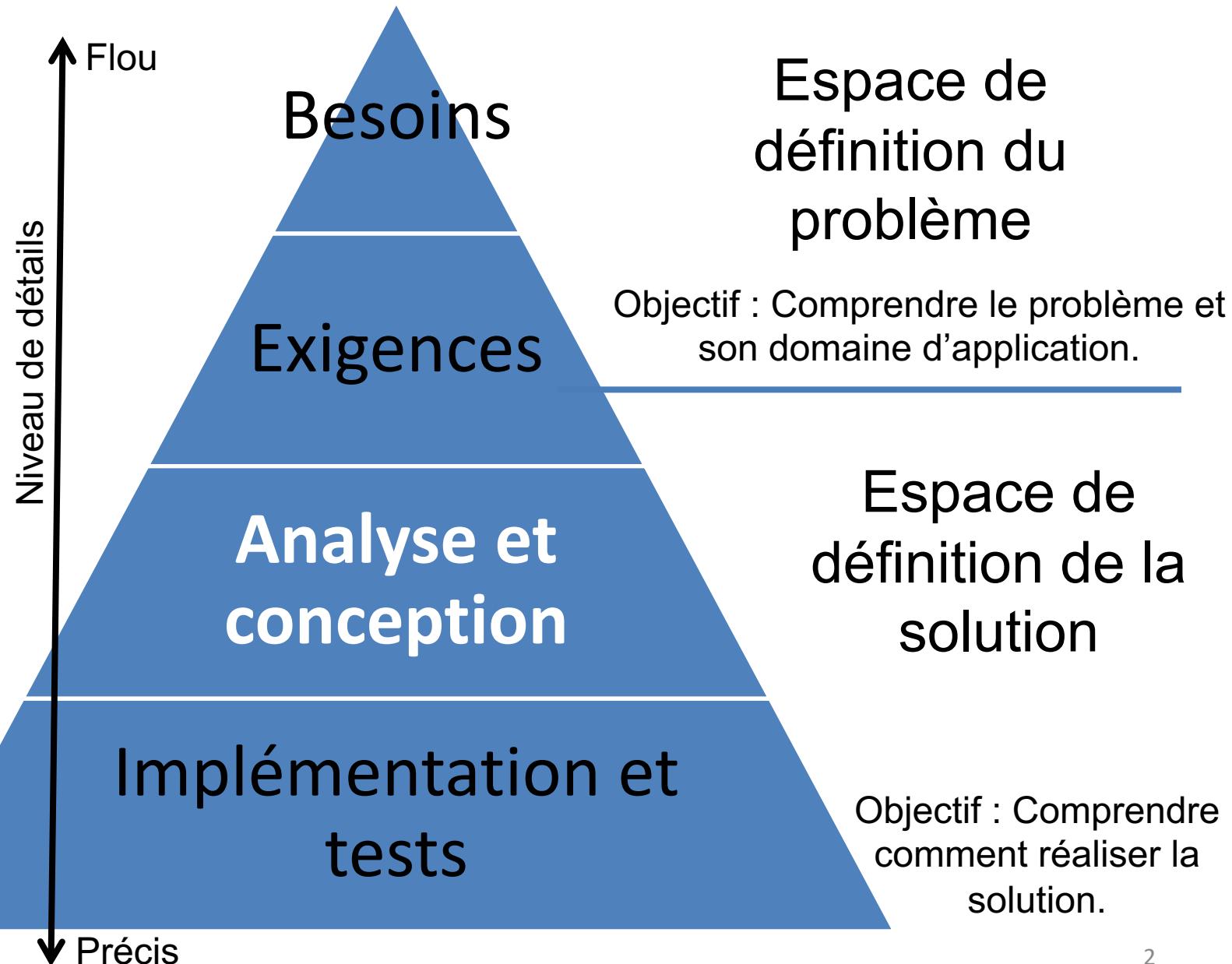
Processus du génie logiciel

1. Position de la discipline
2. L'analyse et la conception selon l'UPEDU
3. Autres approches

***Discipline d'analyse et
conception***

Progression de l'information durant le développement

1. Position de la discipline
2. L'analyse et la conception selon l'UPEDU
3. Autres approches



Génération de solutions

1. Position de la discipline

2. L'analyse et la conception selon l'UPEDU

3. Autres approches

- Le génie logiciel attaque des problèmes « difficiles » (*wicked*).
 - Le problème n'est pas clairement défini.
 - La solution optimale n'est pas clairement définie.
 - Donc, approches de résolution par heuristiques :
 - Par la définition de sous-buts (séparation en parties),
 - Par approximations successives,
 - Par essais-erreurs,
 - Par diagrammes (modèles UML, croquis ...),
 - Par analogie (expérience, règles du pouce, problèmes et solutions similaires ...),
 - Etc.

Définir des activités particulières
peut aider → Processus !

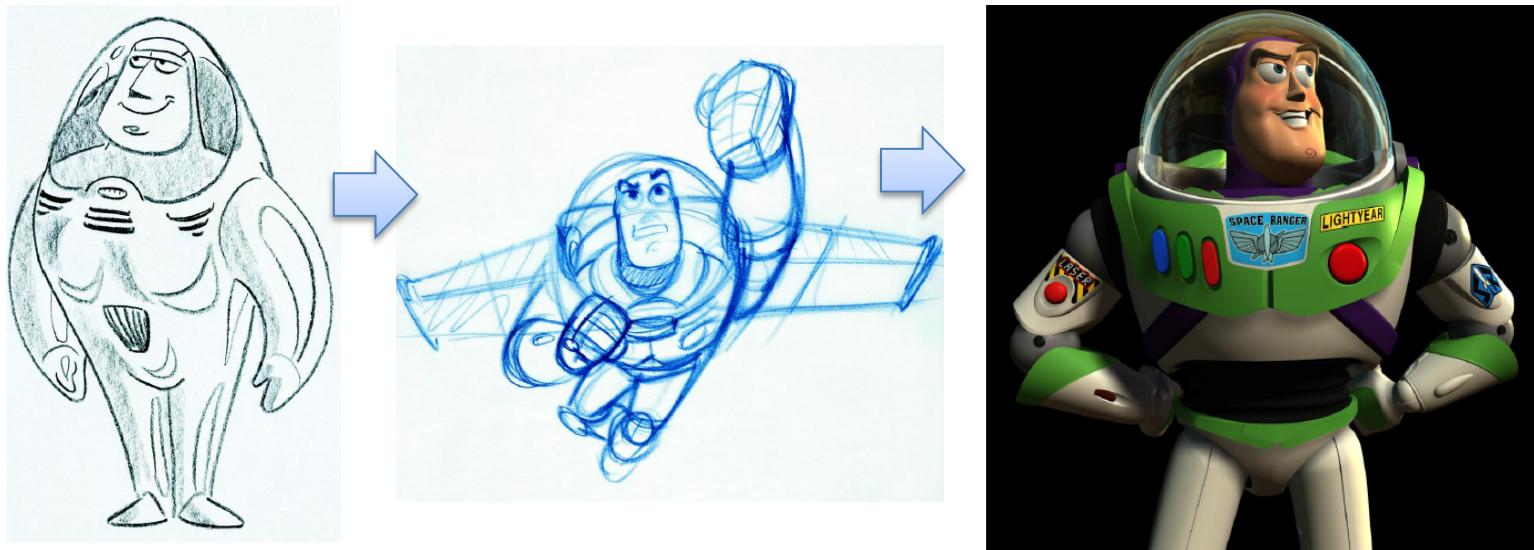
Génération de solutions

Il est difficile de comprendre un problème immédiatement dans son entièreté. Il est plus facile de fonctionner par étapes et par itérations.

1. Position de la discipline

2. L'analyse et la conception selon l'UPEDU

3. Autres approches



Exemple de *storyboard* utilisé pour les dessins animés



Activités cognitives du concepteur

1. Position de la discipline

2. L'analyse et la conception selon l'UPEDU

3. Autres approches

- 1. Acquisition:** Obtenir de l'information.
 - Que se passe-t-il quand le design se base sur des informations incomplètes ?
- 2. Cristalliser:** Produire des solution basé sur les informations.
 - Que se passe-t-il quand le design est mal décrit ?
- 3. Synchroniser:** Communiquer l'information et la solution.
 - Que se passe-t-il quand les concepteurs obtiennent des détails conflictuels ?
- 4. Valider:** Évaluer la qualité de la solution.
 - Que se passe-t-il quand on ne valide pas l'information reçue ?

Difficultés de la génération de solutions

1. Position de la discipline

2. L'analyse et la conception selon l'UPEDU

3. Autres approches

- **Fixation:** Avoir déjà son idée en tête et être convaincu que c'est la meilleure.
 - Le plus longtemps que l'idée est utilisée, le plus difficile c'est à reconsidérer.
 - Complexifier la solution pour s'attacher à certains éléments du problème.
 - Essayer d'arriver immédiatement à une **solution parfaite** : Il vaut mieux faire plusieurs esquisses sommaires.
- Manque de **créativité**: Incapacité de générer des solutions alternatives.
 - S'améliore avec l'expérience, en voyant des solutions ailleurs.
 - Ex.: Vol d'oiseau qui inspire l'avion.
 - Ex.: Biologie et l'algorithme de colonie de fourmis.
- Se perdre dans la recherche d'information, dans les aspects structurels.
 - Constamment **repousser des décisions** sous prétexte qu'on attend plus d'informations.

Approche « satisficing »

1. Position de la discipline

2. L'analyse et la conception selon l'UPEDU

3. Autres approches

- Approche rationnelle: ne pas prendre de décision avant d'avoir toutes les informations pertinentes.
- Approche naturaliste: sauter sur la première solution qui semble marcher.
- Point milieu: rechercher des informations jusqu'à obtenir une gamme de solutions, parmi laquelle s'en trouve une **suffisamment satisfaisante**.
 - *Satisficing* : mélange de *satisfy* (satisfaire) et *suffice* (suffisant).
 - Demander d'identifier les **bénéfices** principaux que la solution doit avoir et les **défauts** qu'on peut accepter.

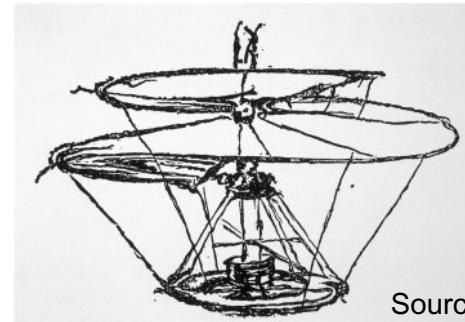
Génération de solutions

- Pourquoi nous faut-il un processus pour l'analyse et la conception?
 1. Parce que nous avons besoin d'**une stratégie** pour découvrir les aspects importants du problème, absorber ses détails, synthétiser une solution, et évaluer la solution.



Génération de solutions

- Pourquoi nous faut-il un processus pour l'analyse et la conception?
2. Parce que concevoir est **opportuniste**:
 - Le chemin d'exploration idéal ne peut pas être prédit d'avance.
 3. Parce que la conception est **réfléctrice**:
 - Elle bénéficie de l'utilisation de représentations (diagrammes) afin de réfléchir sur des idées partiellement formées.
 - Elle bénéficie de la réflexion sur plusieurs niveaux d'abstraction.



Source : Léonard de Vinci !

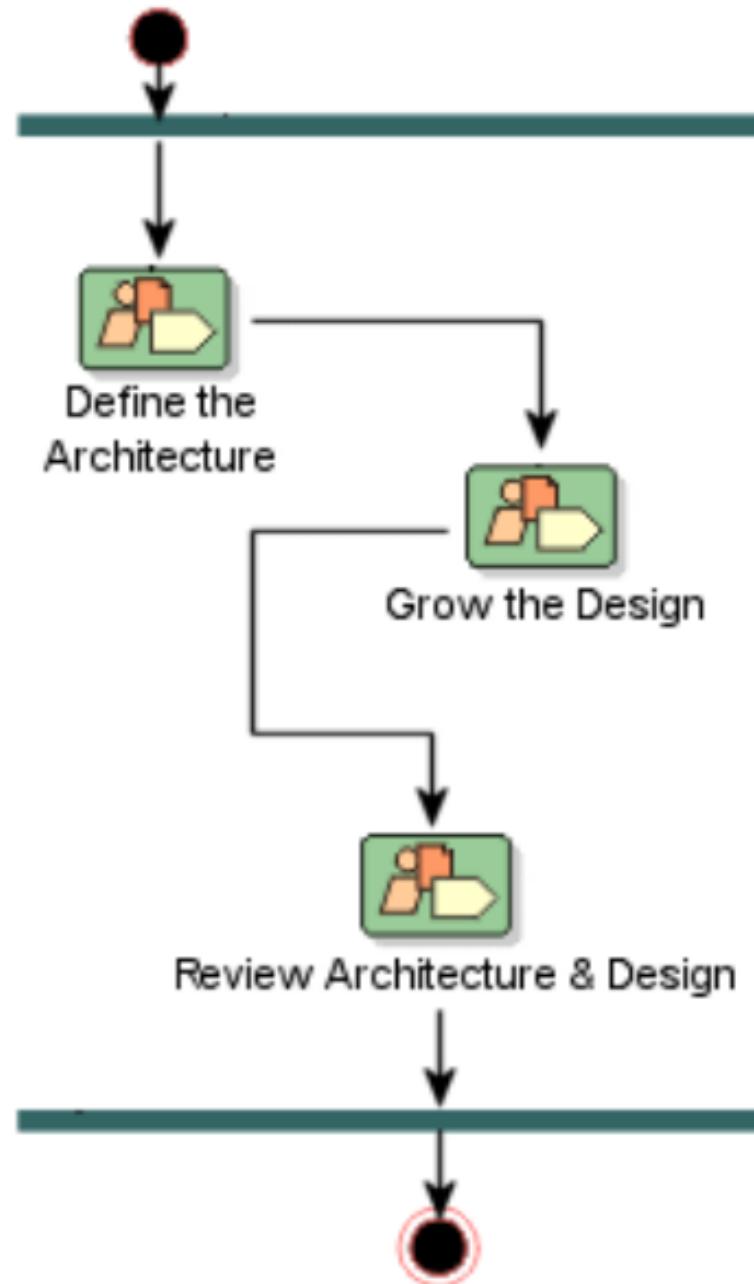
Position de la discipline dans l'UPEDU

1. Position de la discipline
2. L'analyse et la conception selon l'UPEDU
3. Autres approches

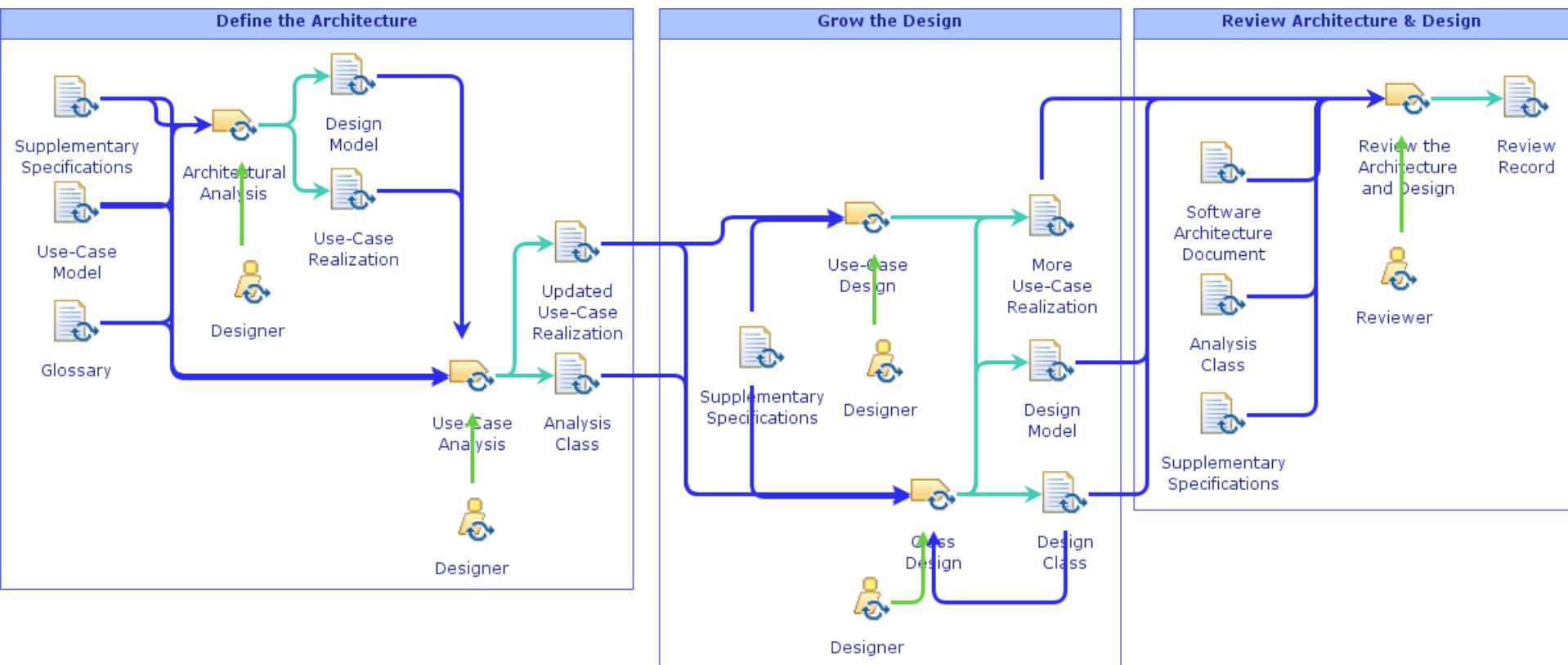
	Discipline	Artéfact principal
Ingénierie	Requis	Cas d'utilisation
	Analyse et conception	Cas d'utilisation détaillés Diagrammes UML
	Implémentation	Programmes (ex.: code)
	Tests	Plan de test et cas de test
Gestion	Gestion de projet	Diagramme de Gantt
	Gestion de configuration	Référentiel (ex.: SVN, Git)

Analyse et conception selon l'UPEDU

1. Position de la discipline
2. L'analyse et la conception selon l'UPEDU
3. Autres approches



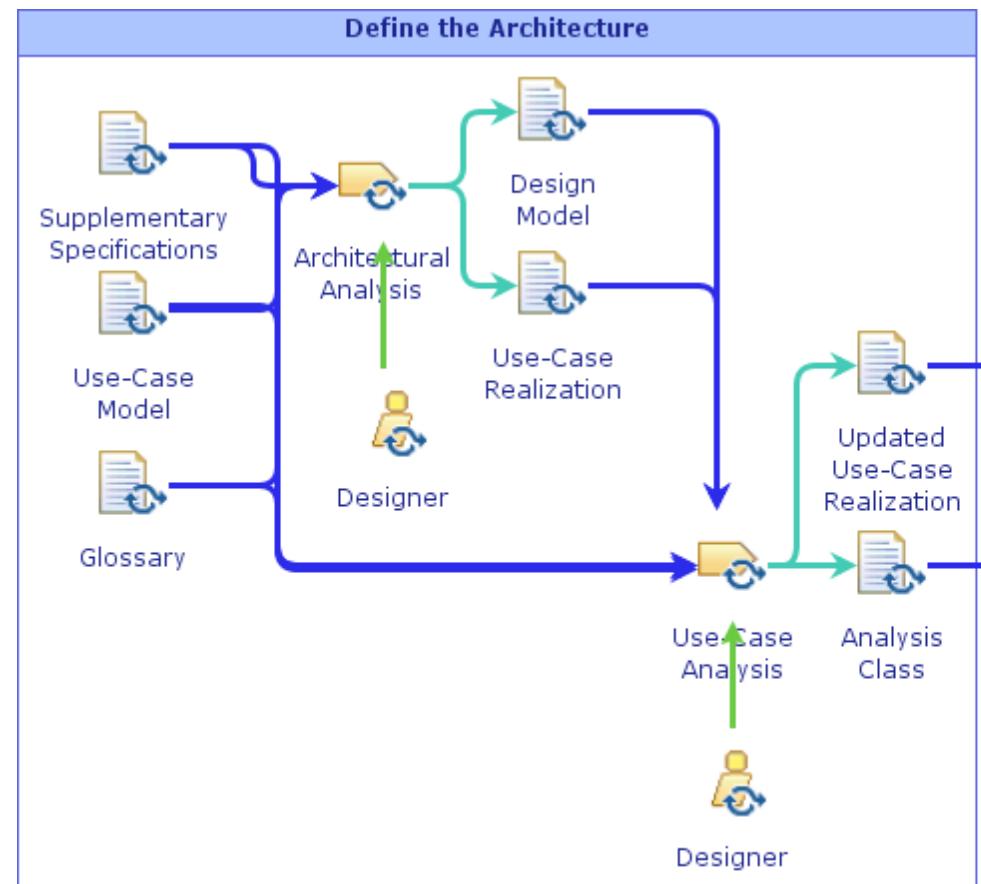
Analyse et conception selon l'UPEDU



Analyse et conception selon l'UPEDU

Define the Architecture

- **Architectural Analysis:** Générer des solutions de base.
- **Use-Case Analysis:** Trouver les objets principaux à partir des cas d'utilisation.



1. Position de la discipline

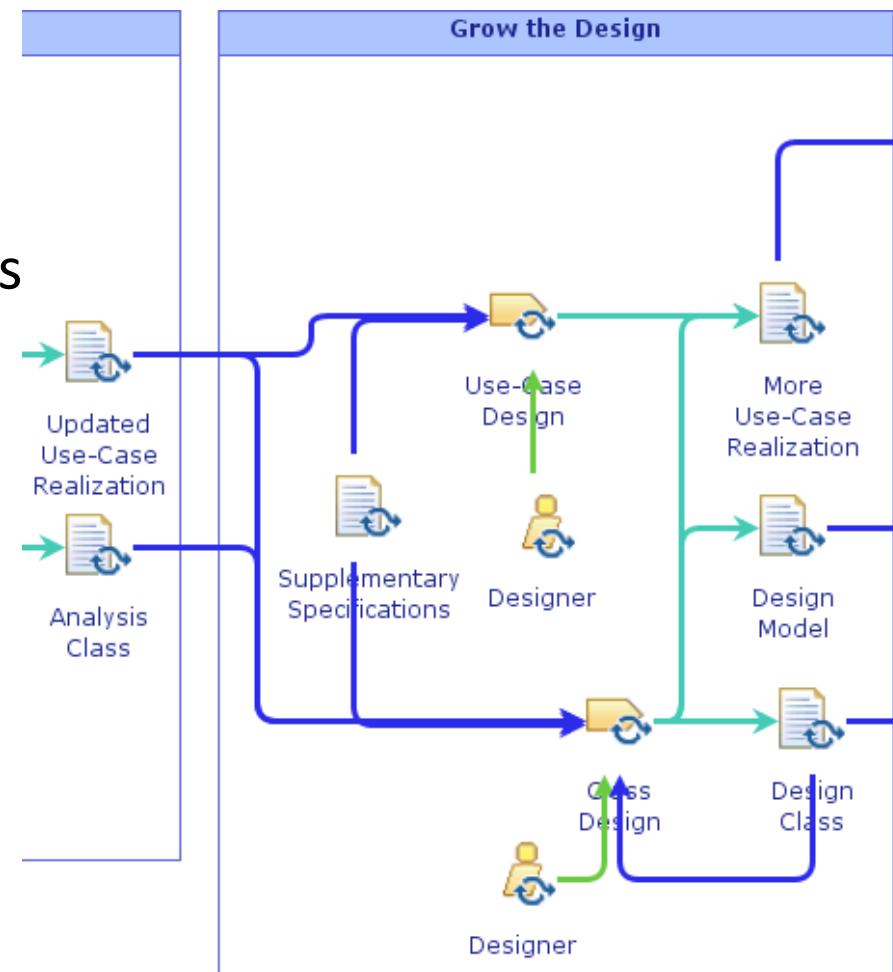
2. L'analyse et la conception selon l'UPEDU

3. Autres approches

Analyse et conception selon l'UPEDU

Grow the Design

- **Use-Case Design:** Spécifier les interactions entre les classes définies (héritage, association, public, privé, etc.).
- **Class Design:** Ajouter les détails pour que chaque classe puisse être implémentée sans ambiguïté.



1. Position de la discipline

2. L'analyse et la conception selon l'UPEDU

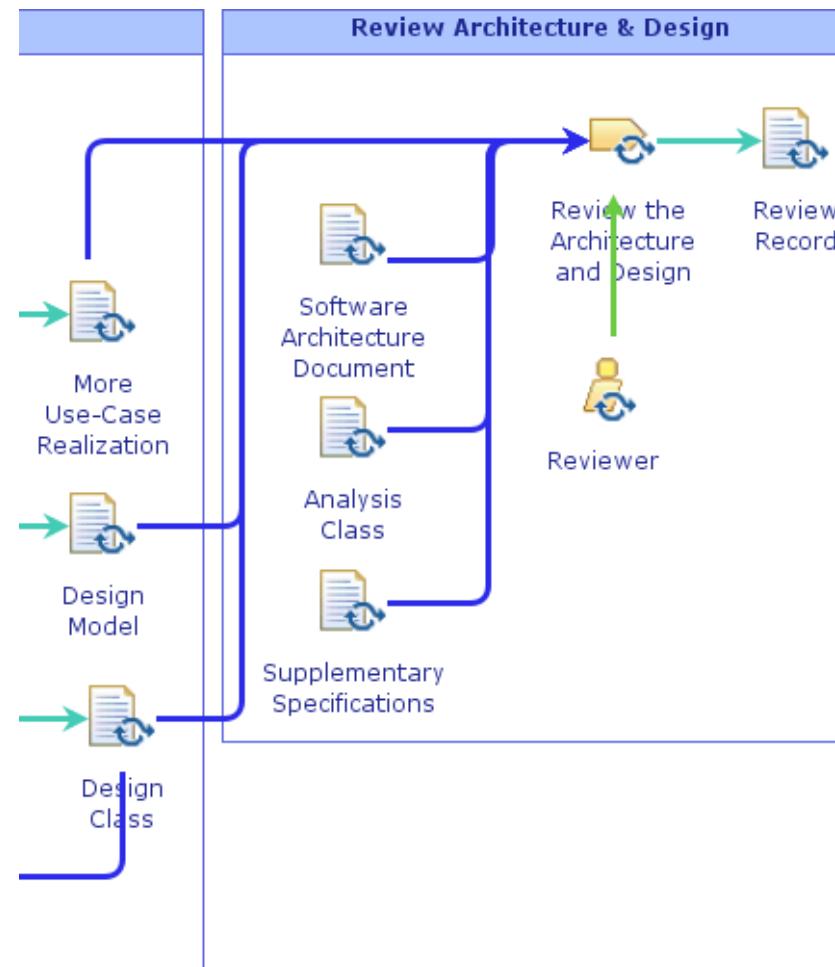
3. Autres approches

Analyse et conception selon l'UPEDU

Review the Architecture and Design

- **Review the Architecture and Design:** Révision des artefacts produits.

1. Position de la discipline
2. L'analyse et la conception selon l'UPEDU
3. Autres approches



Autres approches d'analyse et conception

1. Position de la discipline

2. L'analyse et la conception selon l'UPEDU

3. Autres approches

- Émergence de l'architecture
 - En Agile, il n'y a pas de document d'architecture et on espère que la structure émerge adéquatement.
 - La refactorisation assure que la qualité de l'architecture demeure passable → Mais pas de vue globale.
- Processus ATAM
 - *Architecture Tradeoff Analysis Method* développé par le Software Engineering Institute (avec le CMMI, le PSP, le TSP, etc.).
 - Identification des facteurs de qualité importants (ex.: performance) et construction d'une architecture qui maximise ces facteurs au détriment d'autres moins importants (*tradeoff*).
- Approches heuristiques
 - Essai-erreur, approximations successives, règle du pouce, définition de sous-buts, analogie, diagrammes, etc.
 - Prototypage d'architectures, tests d'architectures.
- Etc.

Il ne faut pas grand-chose pour rendre un produit inutile ...

1. Position de la discipline
2. L'analyse et la conception selon l'UPEDU
3. Autres approches



Sandales de pluie



Chaudron avec poignées accessibles



Bol à soupe facile à vider

Source : Exposition « les inconfortables » de Katerina Kamprani <http://www.kkstudio.gr/projects/the-uncomfortable>



Parapluie solide (en béton)

LOG3000

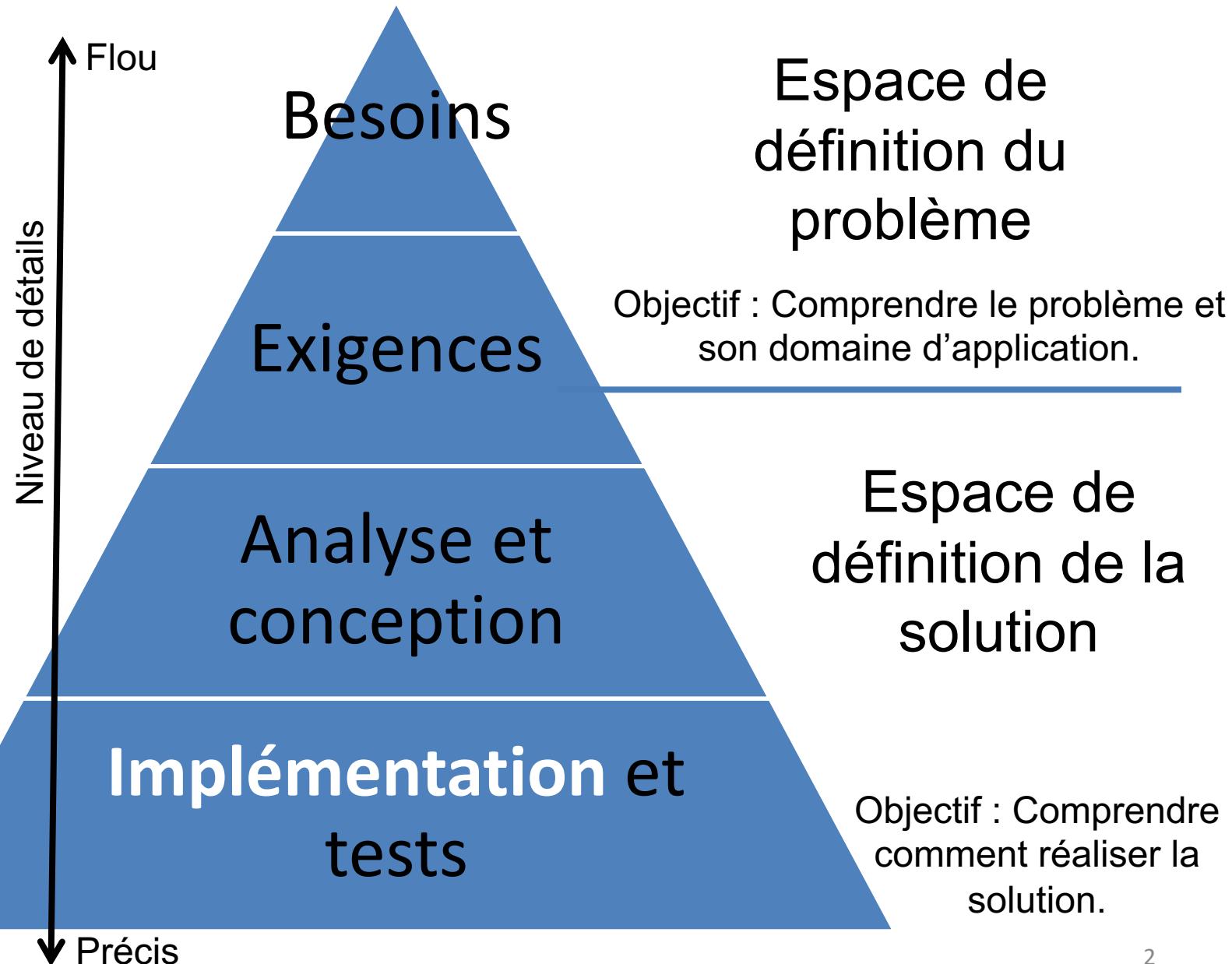
Processus du génie logiciel

1. Position de l'implémentation
2. UPEDU et implémentation
3. Bonnes pratiques de programmation
4. Tester unitaire
5. Intégrer
6. Autres approches

Discipline d'implémentation

Progression de l'information durant le développement

1. Position de l'implémentation
2. UPEDU et implémentation
3. Bonnes pratiques de programmation
4. Tester unitaire
5. Intégrer
6. Autres approches



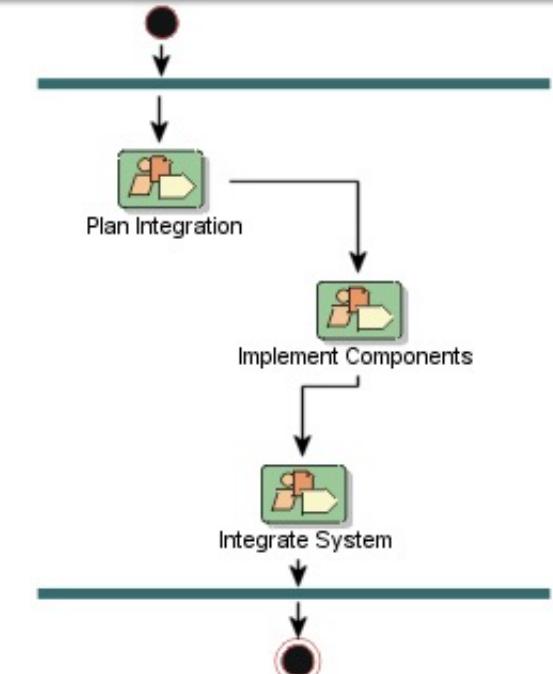
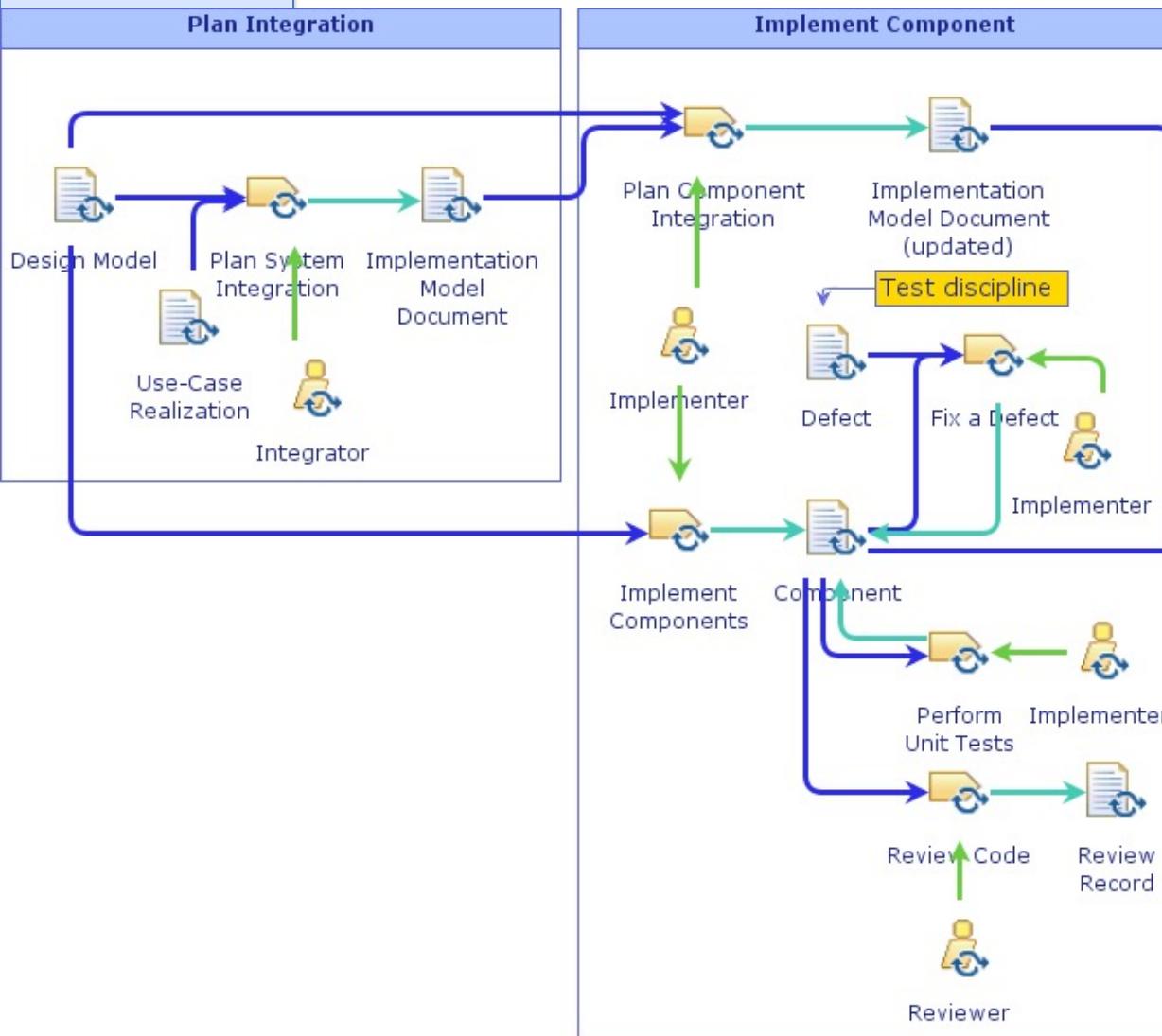
Aspects importants de l'implémentation

1. Position de l'implémentation
2. UPEDU et implémentation
3. Bonnes pratiques de programmation
4. Tester unitaire
5. Intégrer
6. Autres approches

- **Programmer**
 - **Traduire** la spécification en code.
 - Compétences : Compréhension de la conception, du langage de programmation et des API.
- **Tester unitaire**
 - Enlever les **fautes de traduction**.
 - Compétences : Compréhension de la conception.
- **Déboguer**
 - Enlever les **fautes de programmation**.
 - Compétences : Langage de programmation.
- **Intégrer**
 - Préparer l'identification des **fautes de système** (préparer les tests !).
 - Compétences : Outils et API (ex.: Tinderbox, Jenkins ...).



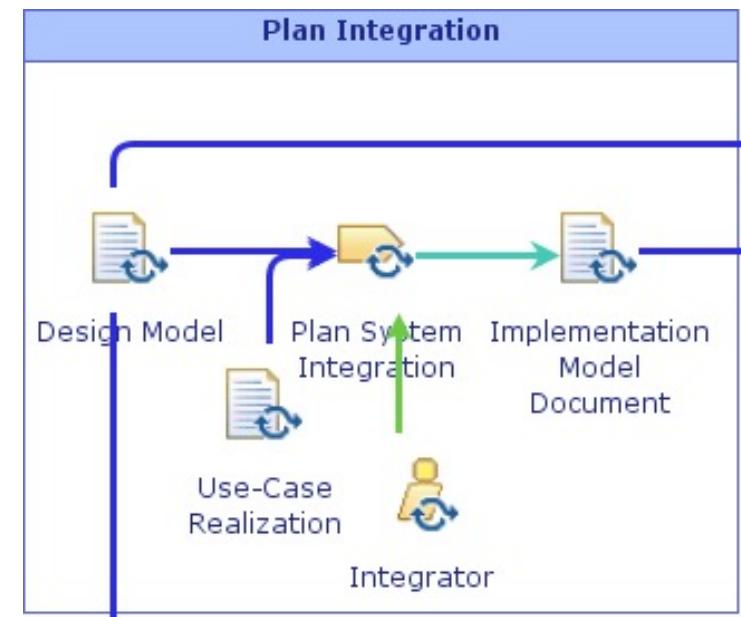
Implémentation selon l'UPEDU



1. Position de l'implémentation
2. UPEDU et implémentation
3. Bonnes pratiques de programmation
4. Tester unitaire
5. Intégrer
6. Autres approches

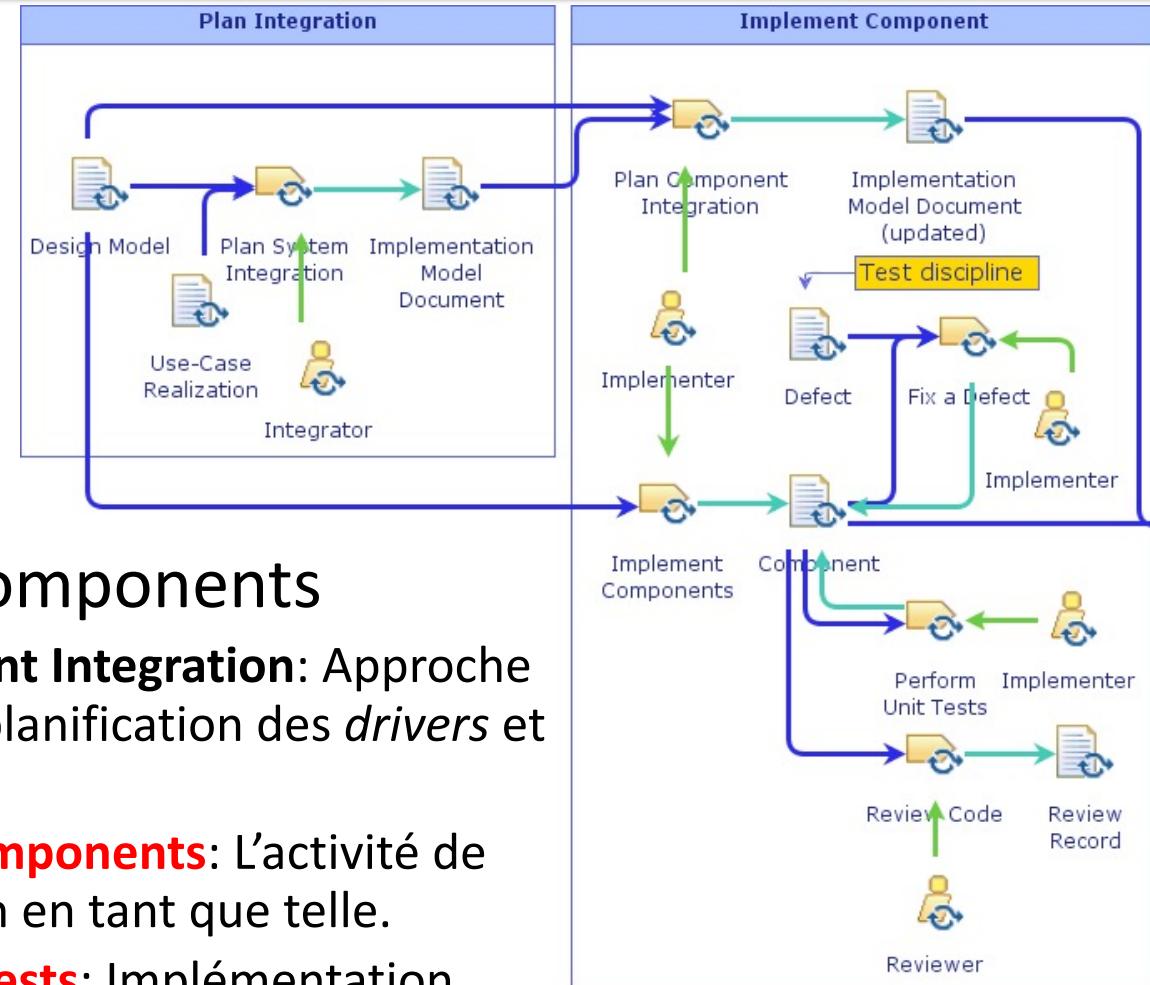
Plan Integration

- **Plan System Integration:** Ordre d'intégration des sous-systèmes en *builds*, et comment ces *builds* sont intégrés dans un système fonctionnel.



Activités de l'UPEDU

1. Position de l'implémentation
2. UPEDU et implémentation
3. Bonnes pratiques de programmation
4. Tester unitaire
5. Intégrer
6. Autres approches



Implement Components

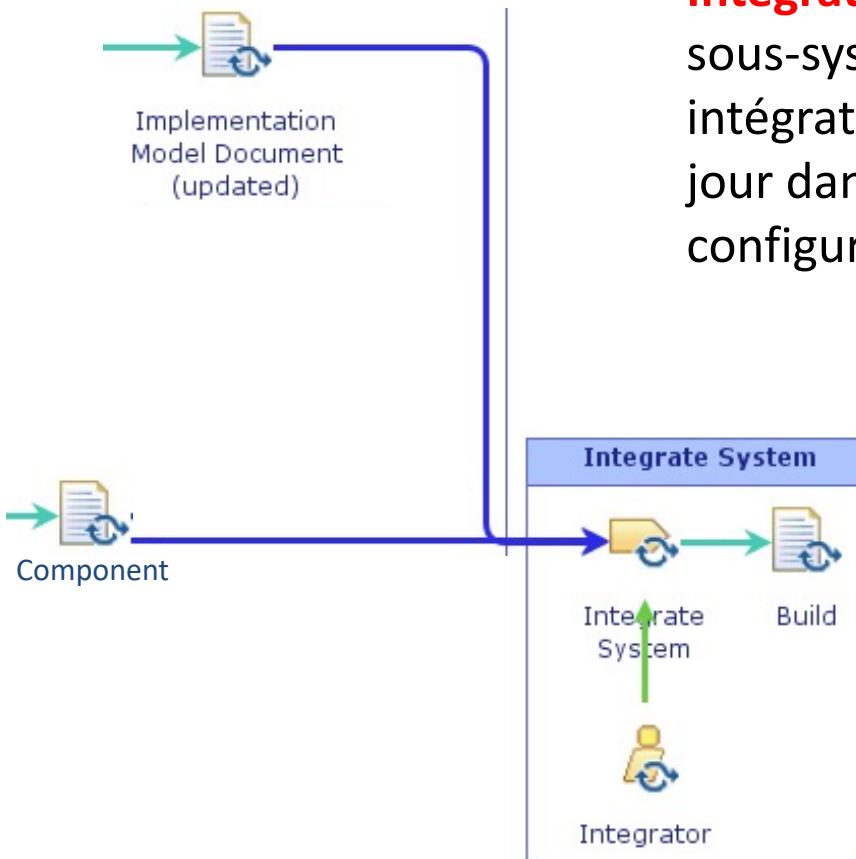
- **Plan Component Integration:** Approche d'intégration, planification des *drivers* et *stubs*.
- **Implement Components:** L'activité de programmation en tant que telle.
- **Perform Unit Tests:** Implémentation, exécution et évaluation des tests unitaires.
- **Fix a Defect:** Répliquer le défaut, trouver la cause et la corriger.
- **Review Code:** Revue manuelle ou automatique selon certains critères (ex.: guide de programmation).

Activités de l'UPEDU

1. Position de l'implémentation
2. UPEDU et implémentation
3. Bonnes pratiques de programmation
4. Tester unitaire
5. Intégrer
6. Autres approches

Integrate System

— **Integrate System:** Intégration des sous-systèmes en *builds*, intégration du système et mise-à-jour dans la gestion de configuration (ex.: SVN, git).



Bonnes pratiques de programmation

1. Position de l'implémentation

2. UPEDU et implémentation

3. Bonnes pratiques de programmation

4. Tester unitaire

5. Intégrer

6. Autres approches

- Principes populaires :
 - DRY: *Don't repeat yourself.* Éviter les redondances.
 - KISS: *Keep it simple, stupid!* Aller vers la solution la plus simple.
 - YAGNI: *You ain't gonna need it.* Éviter de planifier des structures trop complexes.
 - SOLID: Voir
[https://en.wikipedia.org/wiki/SOLID_\(object-oriented_design\)](https://en.wikipedia.org/wiki/SOLID_(object-oriented_design))

Le guide de programmation

1. Position de l'implémentation

2. UPEDU et implémentation

3. Bonnes pratiques de programmation

4. Tester unitaire

5. Intégrer

6. Autres approches

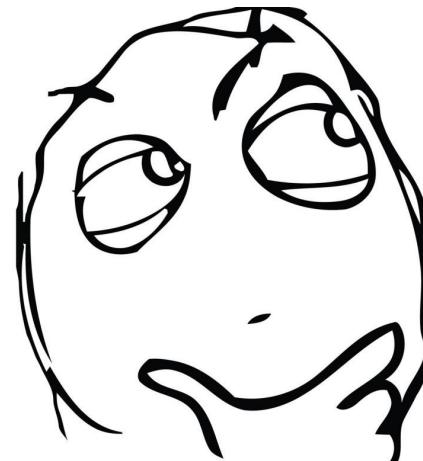
- Questions à se poser sur le guide:
 - Qui est responsable de son application?
 - Comment est-il choisi, rédigé, modifié?
 - Que contient-il?
- Contenu du guide:
 - **Niveau de documentation du code:** Qu'est-ce qui doit être commenté et comment.
 - Ex.: Javadocs, doxygen, entêtes de fichiers, entêtes de fonctions, etc.
 - **Restrictions de programmation:** Y a-t-il certains mécanismes du langage qu'on n'utilise pas.
 - Ex.: Pas de « Goto », minimiser la réflexion, pas de variables globales, etc.
 - **Conventions de nommage:** Comment rédiger les noms de variables, de fonctions, de classes, etc.
 - Constantes en majuscules, classes en *CamelCase*, notation hongroise, etc.
- Approche de création du guide :
 - Généralement sous la responsabilité de l'organisation;
 - créé itérativement selon les besoins des équipes;
 - adapté aux besoins de l'organisation, du projet.

Tests unitaires

1. Position de l'implémentation
2. UPEDU et implémentation
3. Bonnes pratiques de programmation
- 4. Tester unitaire**
5. Intégrer
6. Autres approches

Citation fréquente:

« Mes tests unitaires couvrent 100% de mes lignes de code et passent tous avec succès.
Donc je n'ai plus besoin de tester? »



Que vérifient les tests unitaires ?

1. Position de l'implémentation

2. UPEDU et implémentation

3. Bonnes pratiques de programmation

4. Tester unitaire

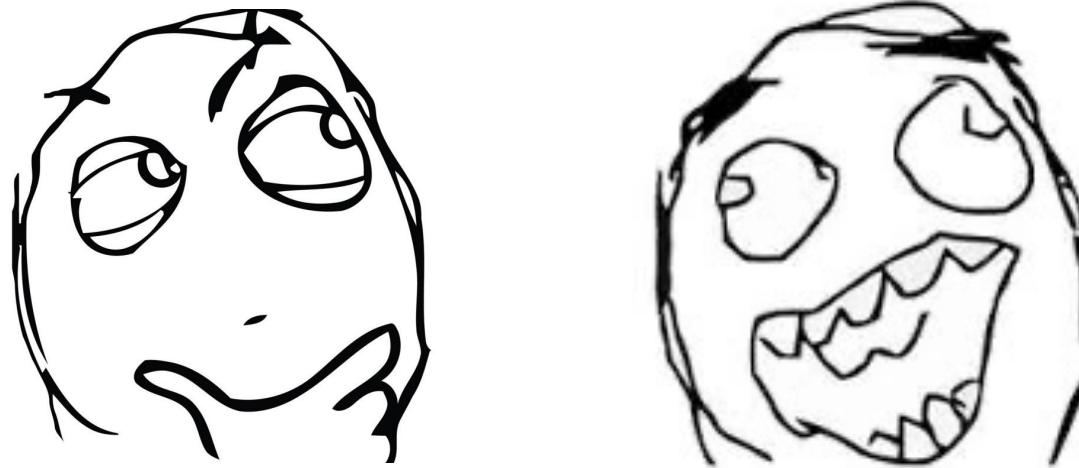
5. Intégrer

6. Autres approches

- Ce qui est testé:
 - Paramètres corrects et incorrects;
 - retour de fonction corrects et incorrects;
 - erreurs et exceptions possibles;
 - préconditions et postconditions;
 - invariants de classe.
- Ce qui n'est pas testé:
 - Est-ce que ça interagit correctement avec une autre fonction ? → **Intégration**
 - Est-ce que ça couvre correctement les spécifications et les exigences? → **Vérification**
 - Est-ce que ça correspond aux besoins du client ? → **Validation**

Citation fréquente

« Mes tests unitaires couvrent 100% de mes lignes de code et passent tous avec succès.
Donc je n'ai plus besoin de tester? »



- **FAUX !**
 - Qualité des tests unitaires eux-mêmes,
 - Tests d'intégration, de validation, de charge, de sécurité, de déploiement, etc.

Intégration

1. Position de l'implémentation

2. UPEDU et implémentation

3. Bonnes pratiques de programmation

4. Tester unitaire

5. Intégrer

6. Autres approches

- Approche selon l'architecture:
 - Bottom-up: Quand les aspects matériels sont importants.
 - Top-down: Quand les interfaces sont importantes.
 - Par silo: Par paquetage, par module relativement indépendant (ex.: client lourd, client léger, serveur ...).
 - En étoile: À partir d'un élément central critique (ex.: plugiciel, *plugin*).
 - Par bus: À travers un canal commun de communication, un portail commun, etc.
 - Pas microservices: Des petits logiciels qui fonctionnent indépendamment les uns des autres.
 - Etc.
- Gestion des « *builds* »
 - Chaque modification à chaque composante peut produire un « *build* » différent → Ça devient lourd !
 - Quels « *builds* » sont produits ? Quand ? Pourquoi ?

Intégration continue (*continuous integration*)

1. Position de l'implémentation

2. UPEDU et implémentation

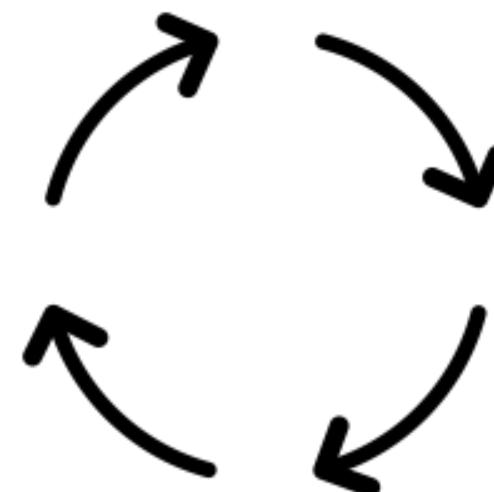
3. Bonnes pratiques de programmation

4. Tester unitaire

5. Intégrer

6. Autres approches

- Que se passe-t-il après un *commit* ?
 - Lancement des tests unitaires;
 - intégration automatique;
 - lancement des tests de système sur le « *build* »;
 - autres tests (performance, sécurité, métriques de qualité du code, etc.).



Autres approches d'implémentation

1. Position de l'implémentation

2. UPEDU et implémentation

3. Bonnes pratiques de programmation

4. Tester unitaire

5. Intégrer

6. Autres approches

- **Pair programming (PP):** Deux programmeur-e-s en face d'un seul ordinateur, avec un clavier et une souris.
 - Coûteux, efficacité controversée. Utile pour les éléments critiques de l'application.
 - Beaucoup de variantes, comme paire programmeur-e/testeur-e.
- **Test-Driven Development (TDD):** Les tests unitaires sont écrits avant le code, ce qui assure que tout le code est testé unitairement.
 - Coûte cher, surtout si le code est finalement jeté.
- **Refactoring:** Coder le minimum possible et nettoyer par la suite le code produit.
 - Suit le principe YAGNI (You Ain't Gonna Need It) qui demande de coder le minimum possible pour passer les tests unitaires.
- **Double Commit:** Le code est soumis dans une branche et est révisé avant d'être fusionné avec le reste.
 - Git est conçu notamment pour cette approche.

LOG3000

Processus du génie logiciel

1. Position de la discipline
2. Les tests selon l'UPEDU
3. Les artefacts des tests
4. Autres approches de test

Discipline des tests

Position de la discipline des tests

- 1. Position de la discipline
- 2. Les tests selon l'UPEDU
- 3. Les artefacts des tests
- 4. Autres approches de test

- Pourquoi tester? → LOG3430
 - Évaluer l'état d'un logiciel dans le but de l'améliorer.
- Qu'est-ce que tester? → LOG3430
 - Examen détaillé d'un logiciel à partir d'information spécifique.
 - Cela exclut: Débogage, tests unitaires.
- Comment rédiger des cas de tests? → LOG3430
 - Compétences techniques requises.
- Quels facteurs de qualité peuvent être évalués par les tests?
→ LOG8371
 - Typiquement: Qualité fonctionnelle, performance, etc.
 - Difficile: Maintenabilité, sécurité, etc.
- **Comment tester?** → LOG3000
 - Processus de tests.
- **Que faire lorsqu'on trouve une erreur?** → LOG3000
 - Processus de gestion de configuration et du changement.

Test ≠ Assurance Qualité

	Assurance Qualité	Test
Concentration	S'assurer que les bonnes choses sont faites.	Évaluer les résultats des choses faites.
Orientation	Orienté sur le processus	Orienté sur le produit
Type d'activités	Préventive/proactive	Corrective/réactive

On ne teste pas seulement qu'à la fin!

Disciplines d'ingénierie

Requis

Analyse & Design

Implementation

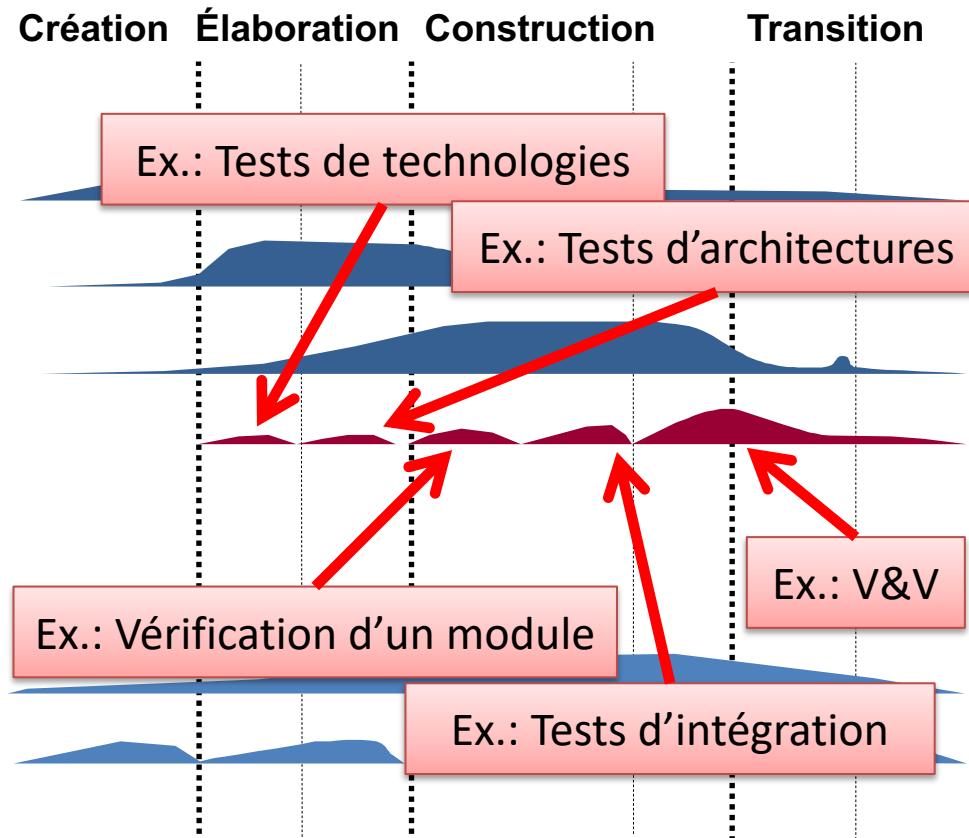
Test

Pratiques de support

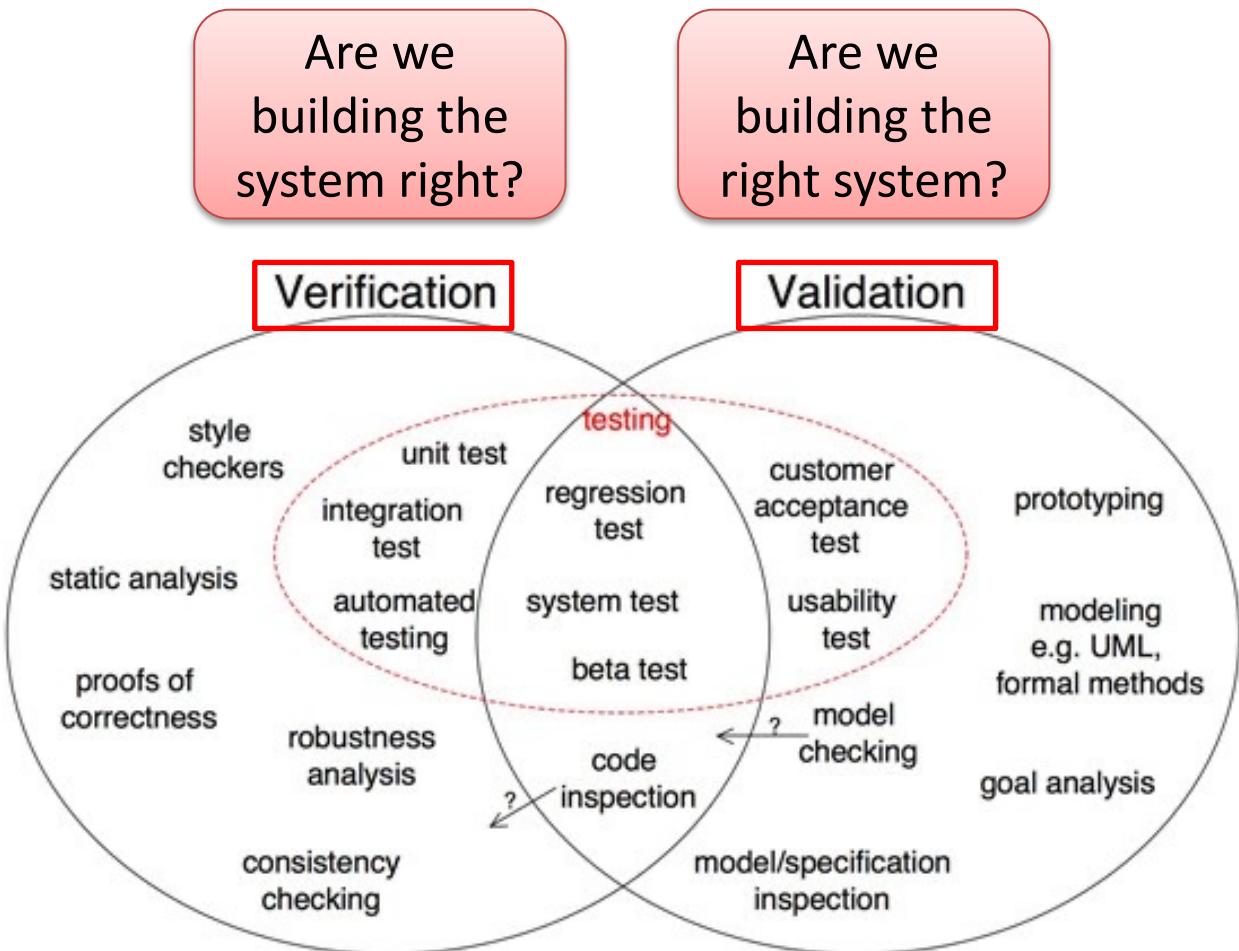
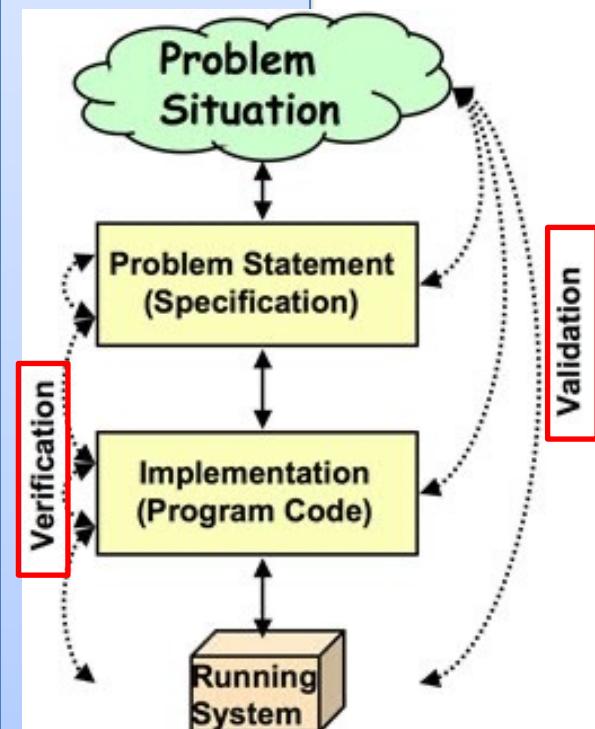
Gestion de la Configuration & Changement

Gestion de projet

Phases du cycle de développement

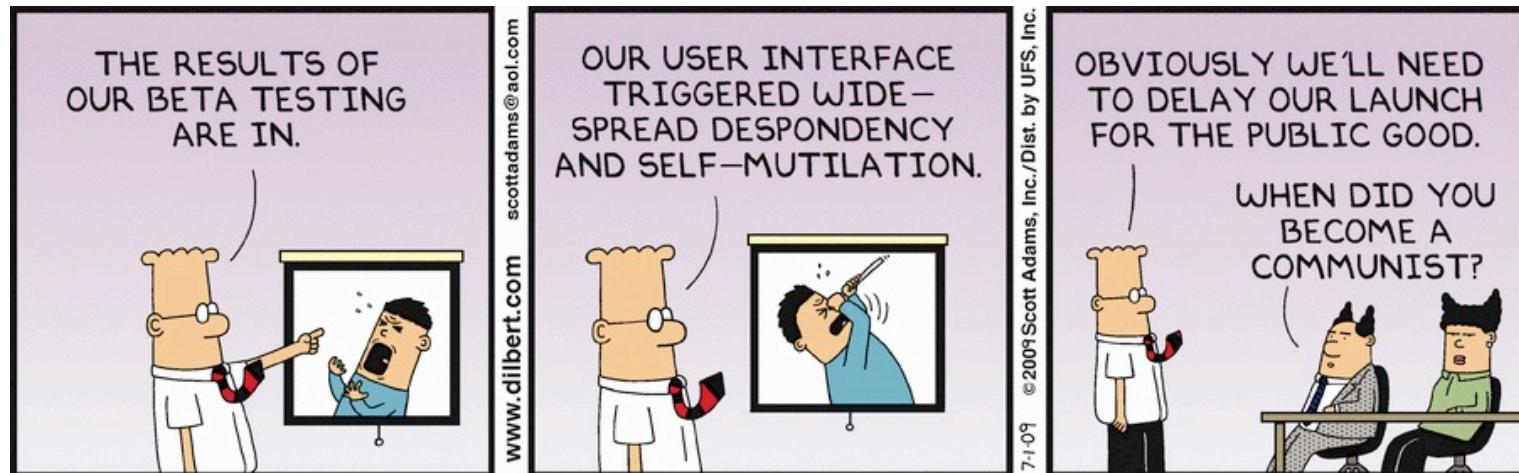


Tests vs. vérification vs. validation

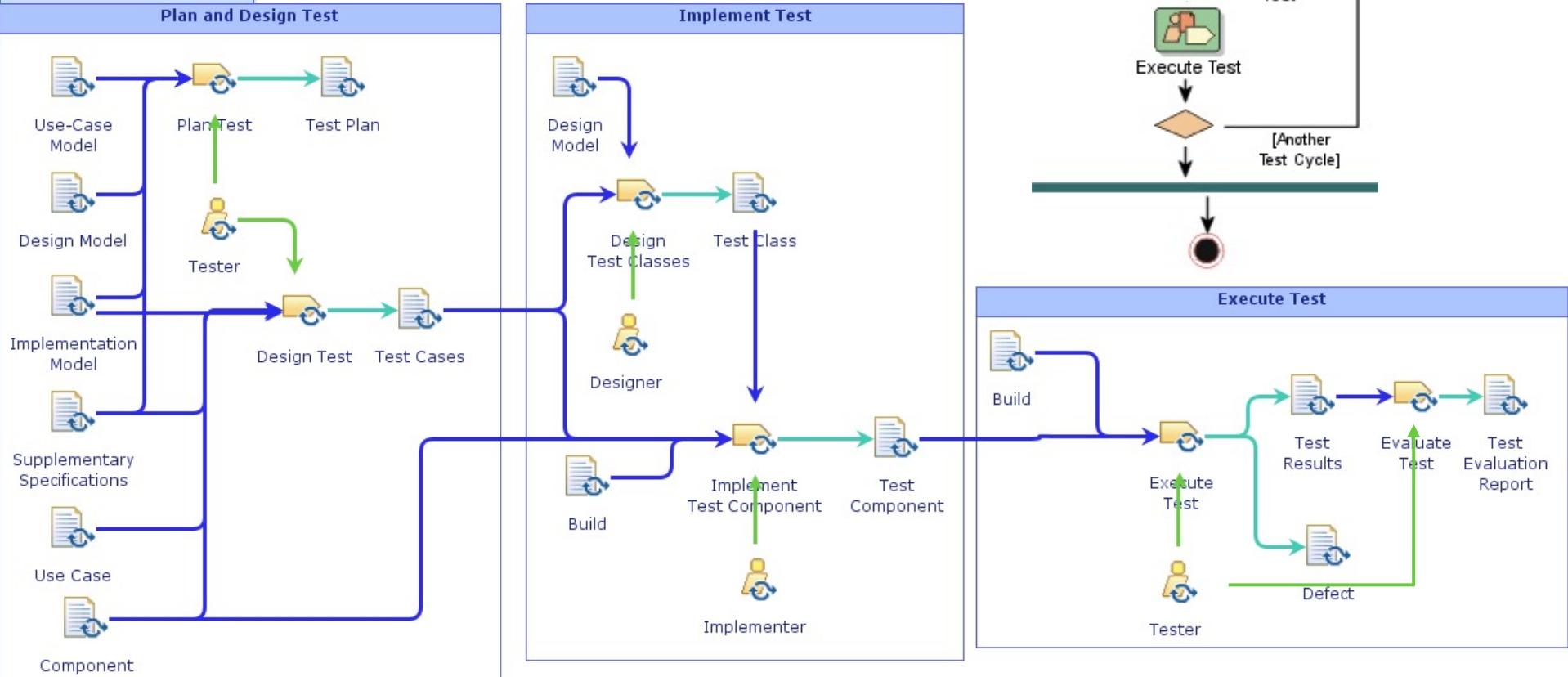


À quoi peuvent servir la discipline des tests?

- **Vérifier** le fonctionnement interne du logiciel et sa cohérence avec la conception et les exigences.
- **Valider** que le système répondra aux besoins réels du client.
- **Trouver et documenter les défauts** dans la qualité du logiciel.
- **Générer des conseils** généraux sur la qualité logicielle perçue.
→ Ex.: recommandations aux gestionnaires.
- **Évaluer les hypothèses** lors des requis et de la conception →
Ex.: Tests de technologies.
- Etc.



Les tests selon l'UPEDU



Activités de test de l'UPEDU

1. Position de la discipline

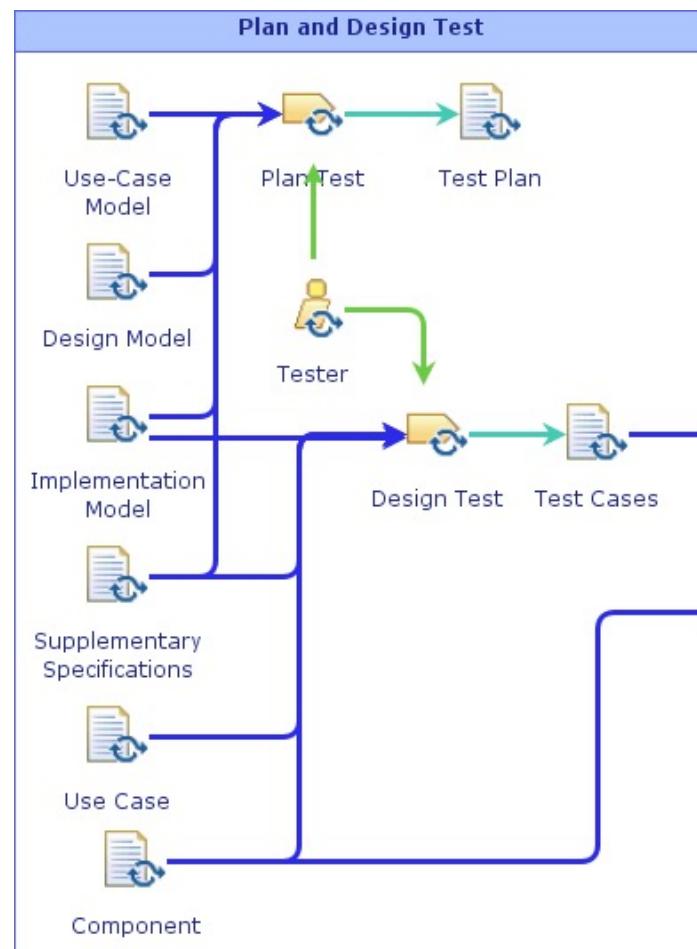
2. Les tests selon l'UPEDU

3. Les artefacts des tests

4. Autres approches de test

Plan and Design Test

- **Plan Test:** Créer un plan de projet (ressources, risques, etc.) pour les tests.
- **Design Test:** Description des procédures et cas de tests à faire.



Activités de test de l'UPEDU

1. Position de la discipline

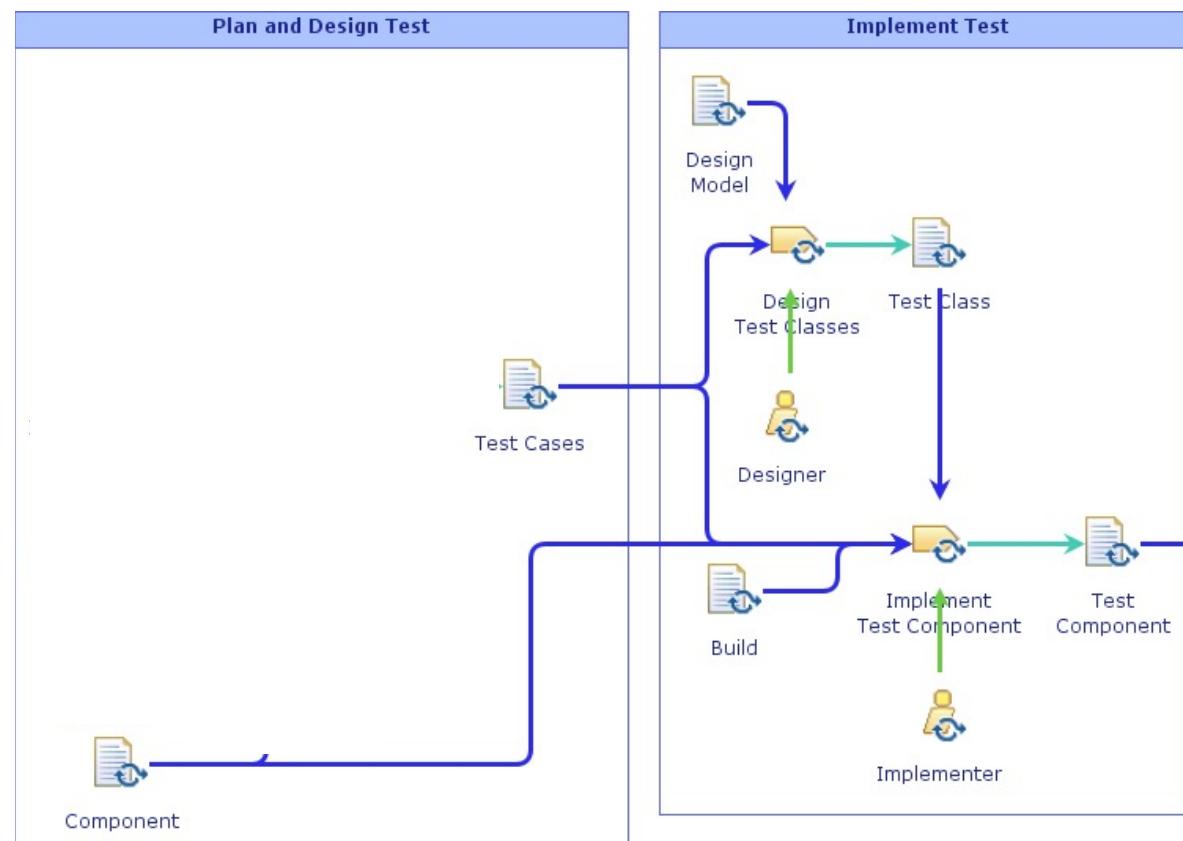
2. Les tests selon l'UPEDU

3. Les artefacts des tests

4. Autres approches de test

Implement Test

- **Design Test Classes:** Concevoir des fonctionnalités pour les tests (classes, interfaces et procédures pour automatiser des tests).
- **Implement Test Components:** Implémenter et tester les composants (*drivers, stubs, scripts, etc.*) nécessaire pour l'automatisation des tests.

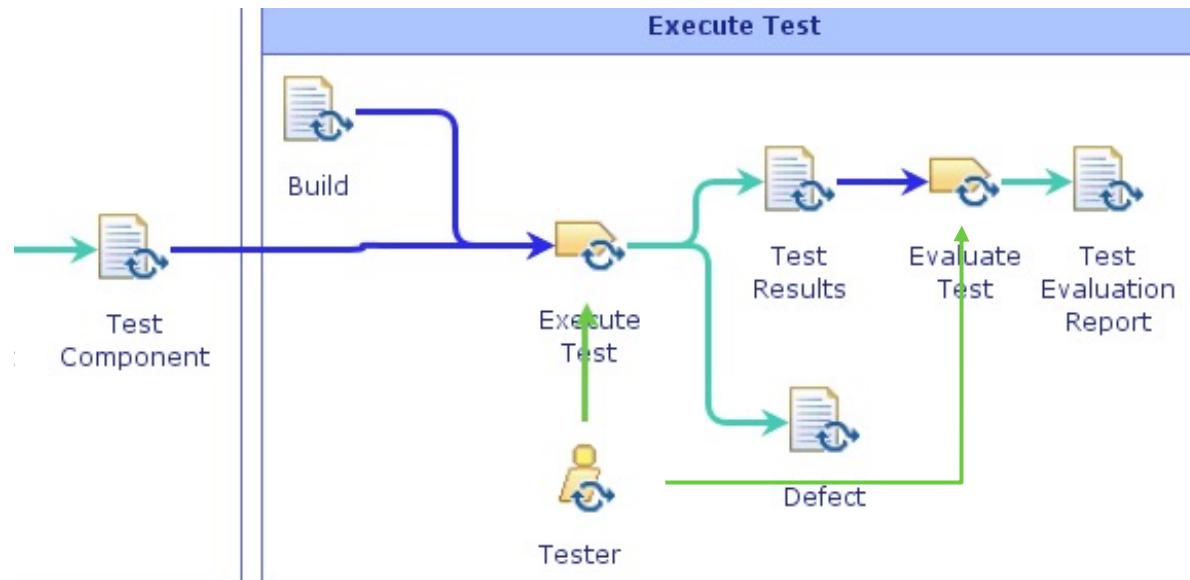


Activités de test de l'UPEDU

Execute Test

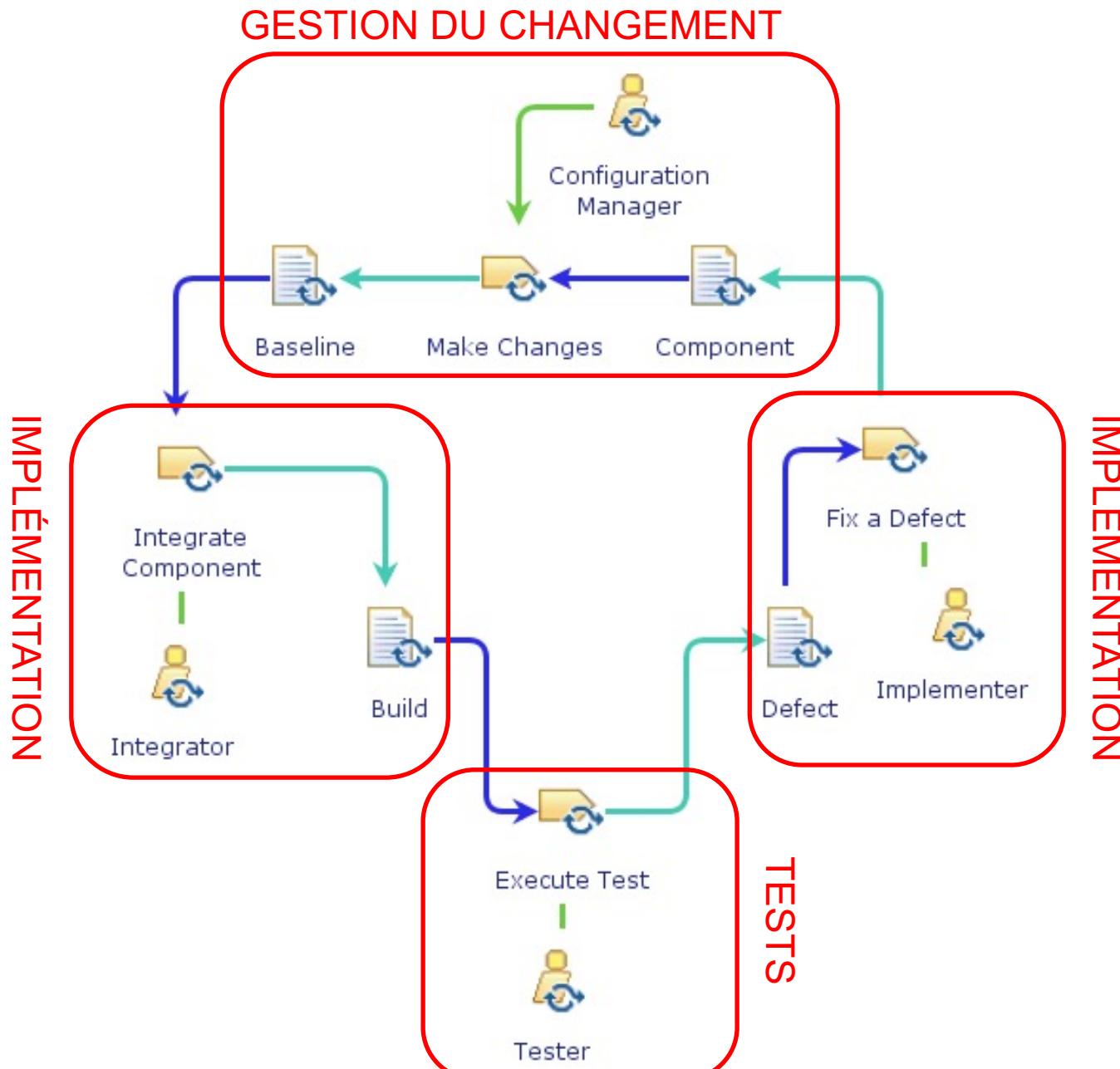
- **Execute Test:** Exécuter les tests et capturer les résultats.
- **Evaluate Test:** Noter les demandes de changements et produire un rapport résumé pour les gestionnaires.

1. Position de la discipline
2. Les tests selon l'UPEDU
3. Les artefacts des tests
4. Autres approches de test



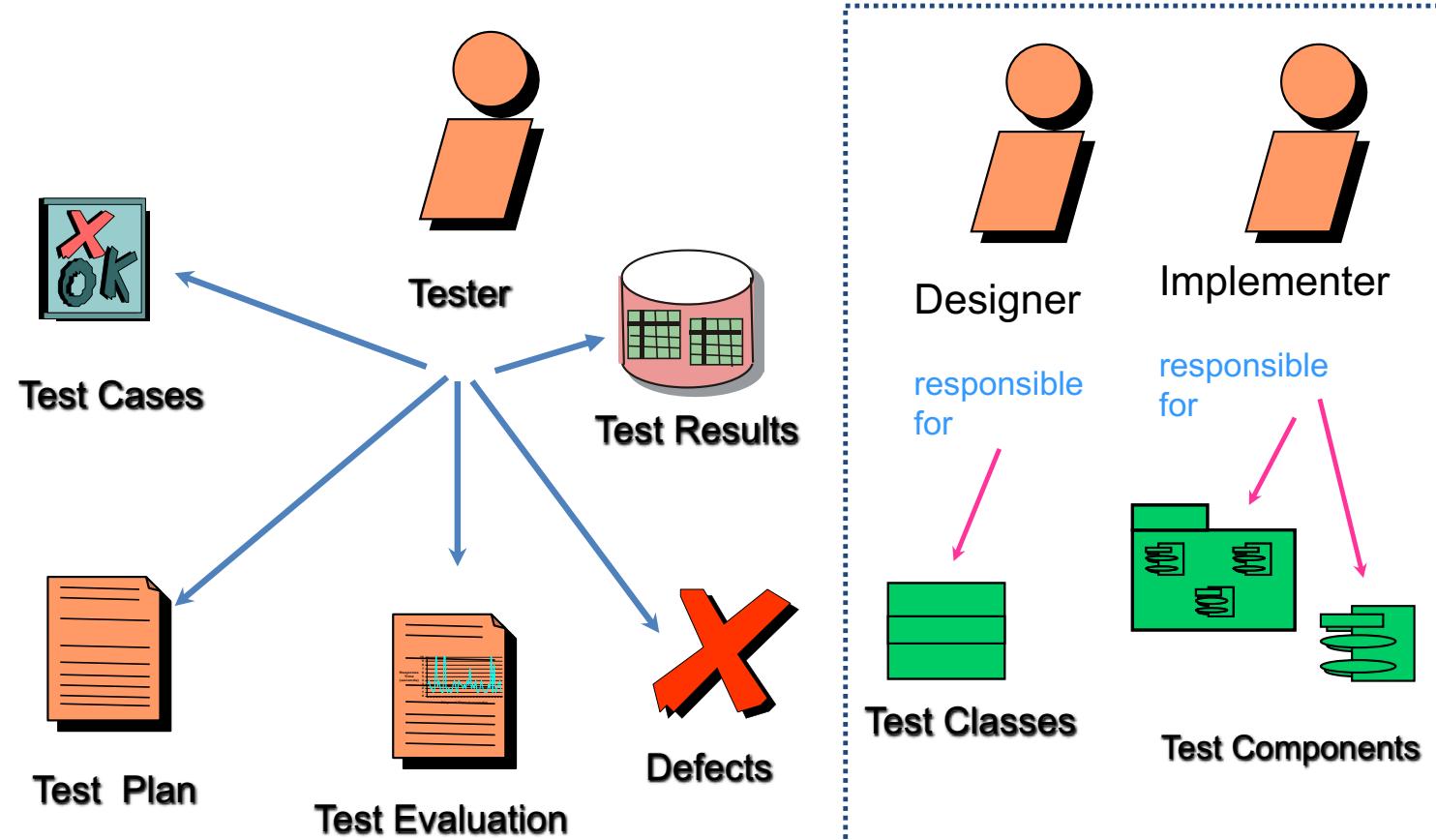
Les tests : Une danse compliquée !

1. Position de la discipline
2. Les tests selon l'UPEDU
3. Les artefacts des tests
4. Autres approches de test



Et surtout: Beaucoup d'artéfacts à gérer !

1. Position de la discipline
2. Les tests selon l'UPEDU
3. Les artéfacts des tests
4. Autres approches de test



1. Position de la discipline
2. Les tests selon l'UPEDU
3. Les artefacts des tests
4. Autres approches de test

- **Plan de test:** Planification des ressources pour les tests.
- **Design du test:** Description des cas de tests, des résultats attendus et des seuils de passage.
- **Cas de test:** Description des données requises pour faire le test.
- **Procédure de test:** Description de la manière dont chaque test doit être exécuté, incluant les préconditions.
- **Rapport de test:** Description chronologique de l'exécution du test.
- **Rapport d'anomalie:** Tout élément apparu durant les tests qui doit être étudié avec plus de profondeur.
- **Rapport complet:** Résumé des résultats de l'ensemble des tests.

Standards utiles pour la discipline des tests

1. Position de la discipline
2. Les tests selon l'UPEDU
3. Les artefacts des tests
4. Autres approches de test

- Standards de l'IEEE:
 - IEEE 829 : Standard for software test documentation.
 - IEEE 1008 : Standard for Software Unit Testing.
 - IEEE 1012 : Standard for Software Verification and Validation.
 - IEEE 1044 : Standard Classification for Software Anomalies.



1. Position de la discipline

2. Les tests selon l'UPEDU

3. Les artefacts des tests

4. Autres approches de test

Contenus:

- **Les objectifs des tests**
 - Dépend du contexte et du phase du projet; se baser sur les exigences
 - Ex.: intégration, sécurité, flux de données, acceptation des utilisateurs ...
- **Les stratégies à utiliser pour les différents types de tests.**
 - Ex.: tests automatisés, tests manuels, tests avec utilisateur présent, tests sur simulateur ...
- **Planification des ressources nécessaires pour les tests.**
 - Matériel, logiciel, personnes, échéancier, etc.
 - Ex.: diagramme de Gantt juste pour les tests.

Document évolutif, raffiné à chaque itération.

- Au début du projet, le *Master Test Plan (MTP)* décrit grossièrement les activités de tests.
- À chaque itération, on ajoute les détails reliés aux tests qui seront faits durant l'itération.

Autres approches de test

1. Position de la discipline

2. Les tests selon l'UPEDU

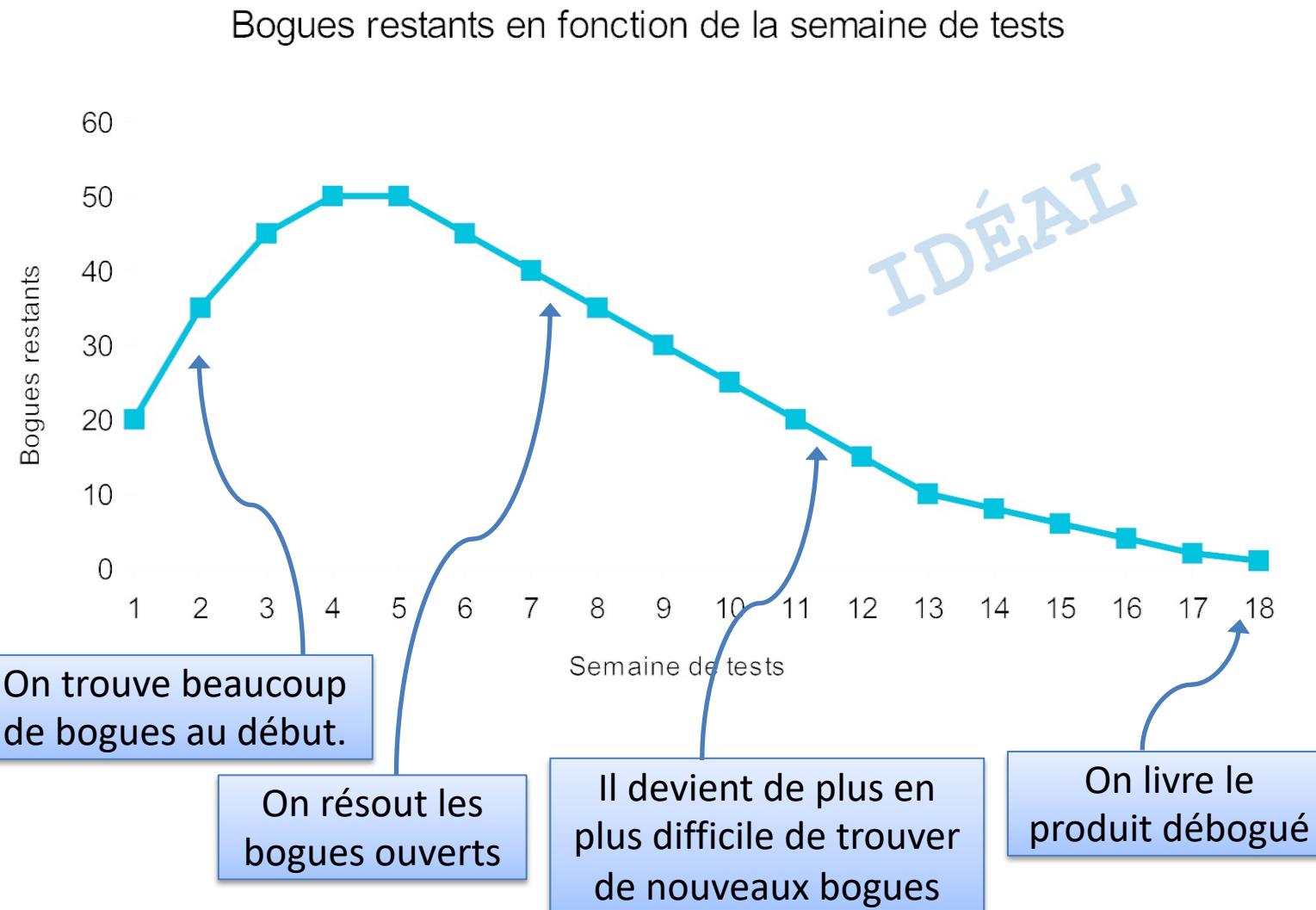
3. Les artéfacts des tests

4. Autres approches de test

- Tests faits à l'interne vs. **équipe externe indépendante**.
 - Souvent un mélange des deux : Interne pour la vérification, et externe pour la validation.
- « **Good enough quality** »: La perfection n'est jamais atteinte, définissons un seuil d'acceptabilité.
 - Généralement absence de *show-stoppers*, mais présence d'irritants qu'on sait qu'on pourra corriger après livraison.
 - Souvent besoin de données d'utilisation, fréquence d'apparition du bogue, etc.
- Revues et inspections
 - Évaluation **statique** des artéfacts produits (code, SRS, architecture, etc.). Le code n'est pas exécuté.
- Tests d'utilisabilité, d'accessibilité, etc.
 - Tests qui évaluent des **facteurs humains** d'utilisation du logiciel.
- Etc.

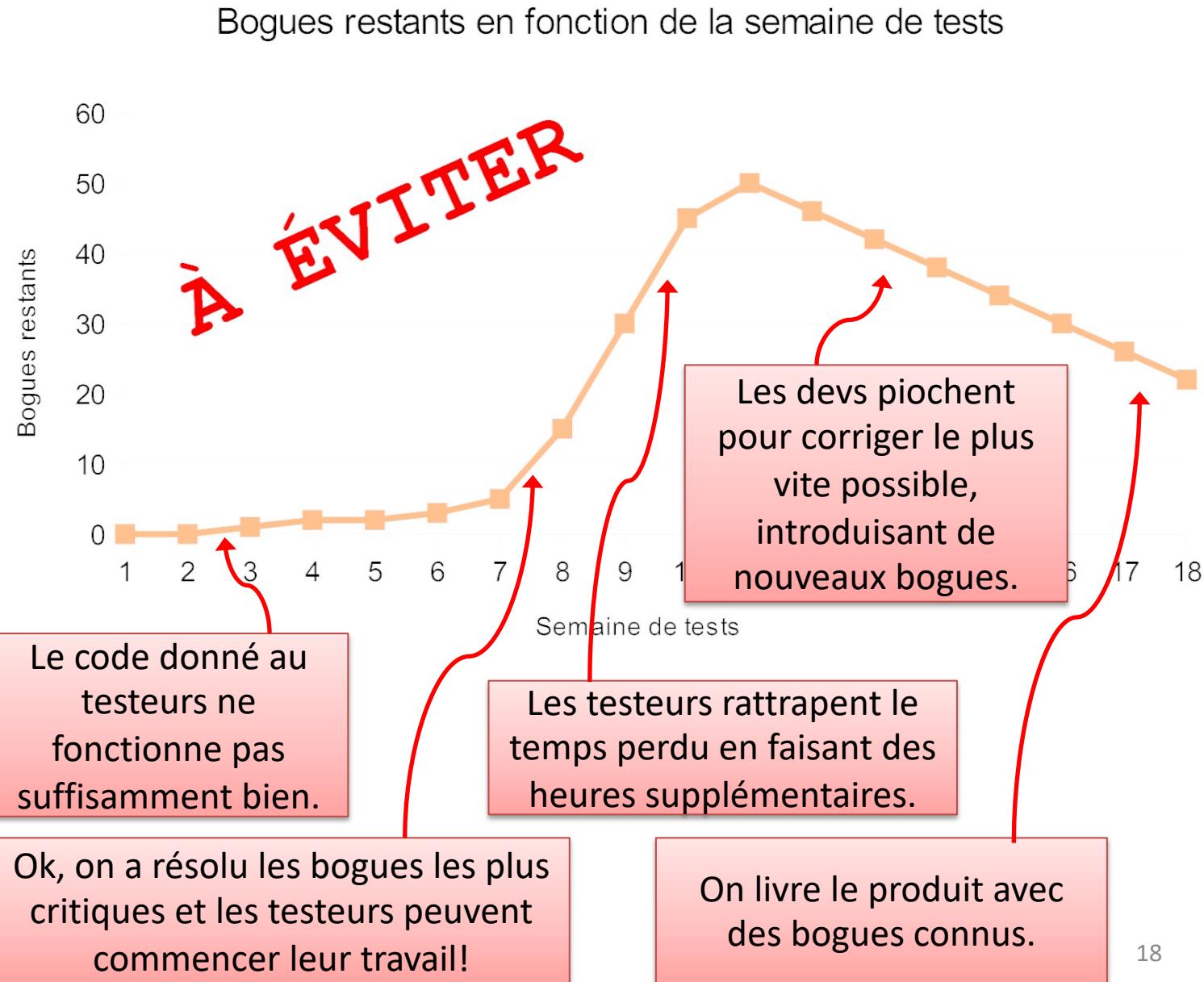
Courbes de détection de bogues

1. Position de la discipline
2. Les tests selon l'UPEDU
3. Les artefacts des tests
4. Autres approches de test



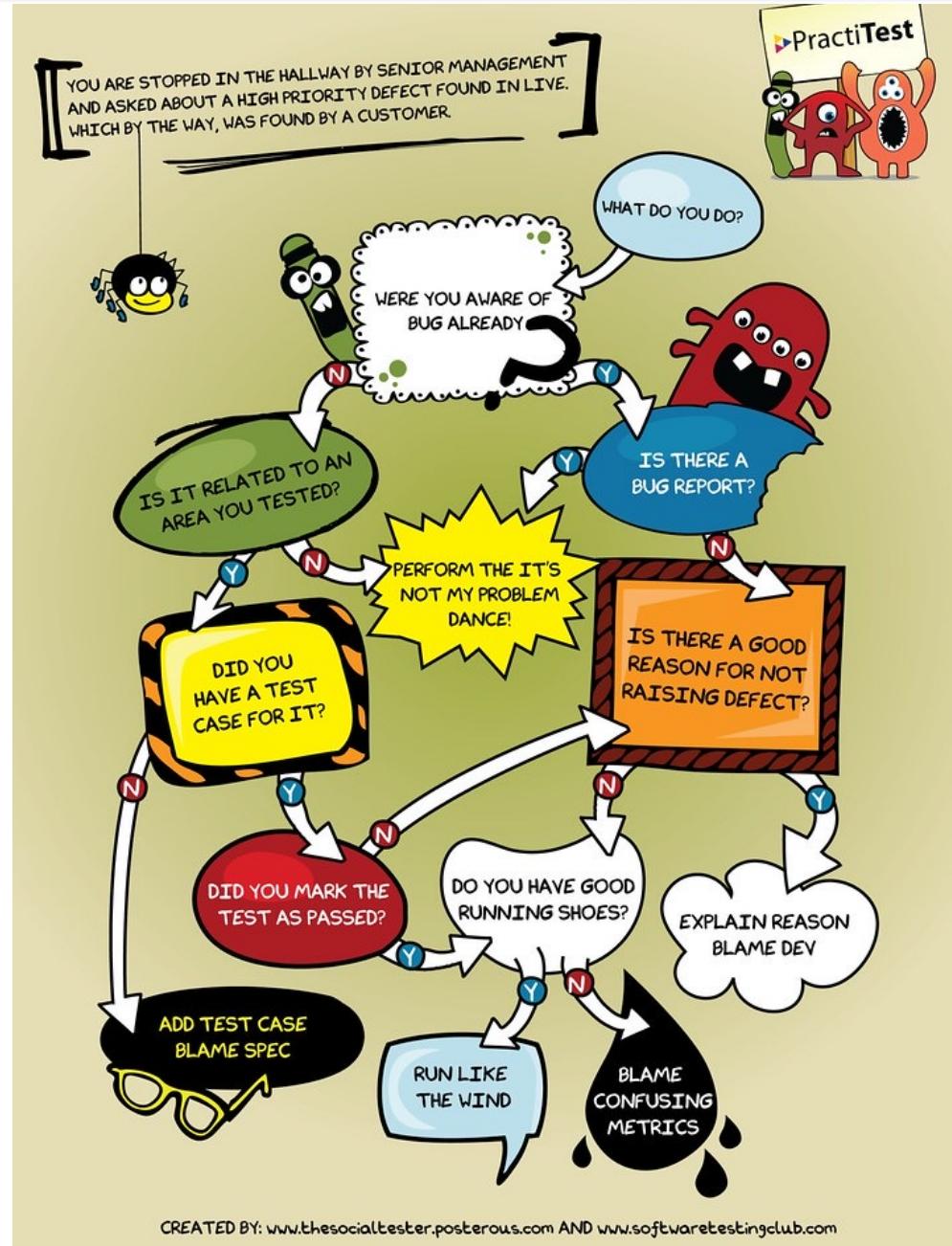
Courbes de détection de bogues

1. Position de la discipline
2. Les tests selon l'UPEDU
3. Les artefacts des tests
4. Autres approches de test



Discipline des tests

1. Position de la discipline
2. Les tests selon l'UPEDU
3. Les artefacts des tests
4. Autres approches de test



LOG3000

Processus du génie logiciel

1. Position de la gestion de projet
2. Gestion de projet selon l'UPEDU
3. Processus vs. Projet
4. Outils

Discipline de gestion de projet

Phases du cycle de développement

1. Position de la gestion de projet

2. Gestion de projet selon l'UPEDU

3. Processus vs. Projet

4. Outils

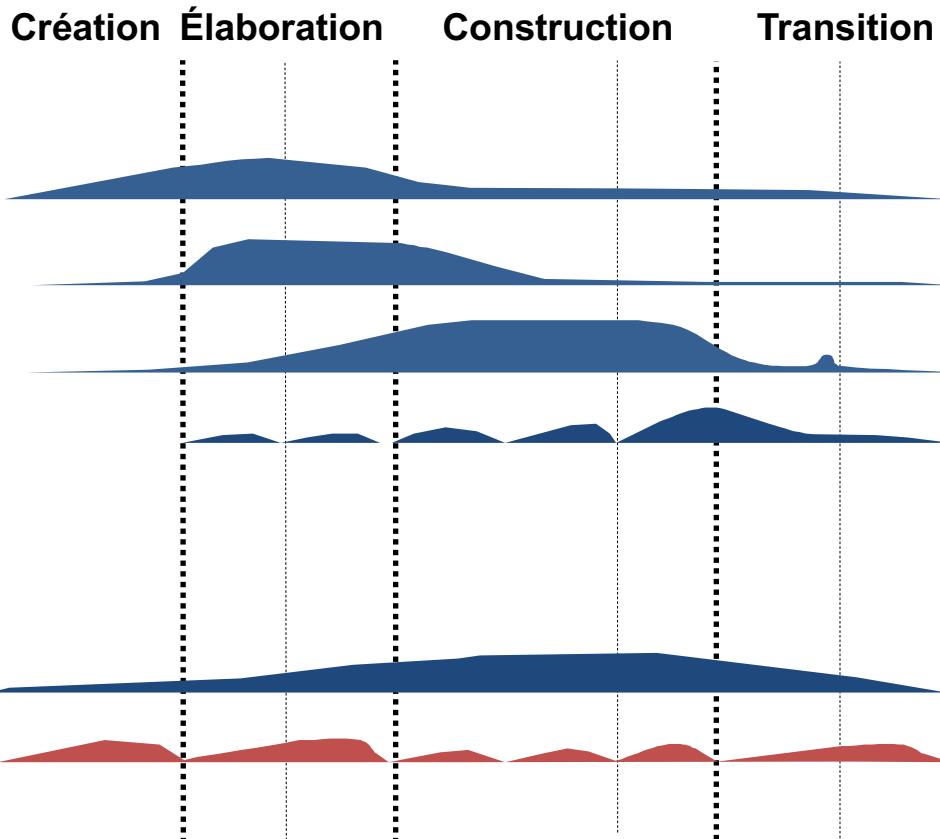
Disciplines d'ingénierie

Requis
Analyse & Design
Implementation
Test

Disciplines de support

G. de la Config. & Changement

Gestion de projet



Aspects importants de la gestion de projet

1. Position de la gestion de projet

2. Gestion de projet selon l'UPEDU

3. Processus vs. Projet

4. Outils

- **Planification des itérations:** On fait quoi ? Qui fait quoi ? Combien de temps faut-il pour chaque tâche ?
- **Gestion des risques:** Quels sont les écueils majeurs du projet ? Comment puis-je les éviter ? Qu'est-ce que je dois surveiller pour m'assurer qu'ils n'apparaissent pas ?
- **Suivi de l'avancement:** Qu'est-ce qui est complété ? Que reste-t-il à faire ? Quand est-ce que tout sera terminé ?

1. Position de la gestion de projet

2. Gestion de projet selon l'UPEDU

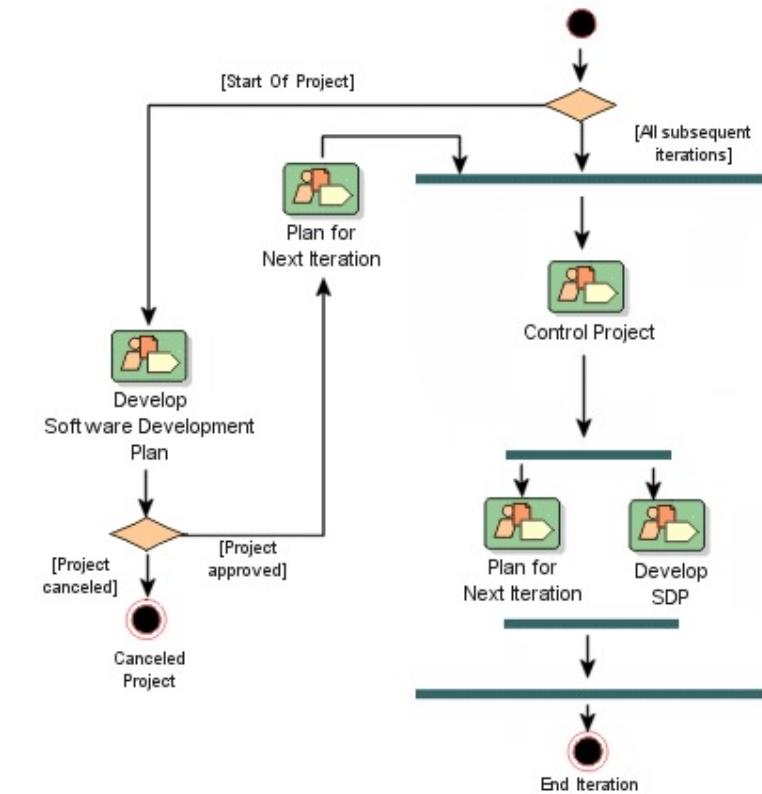
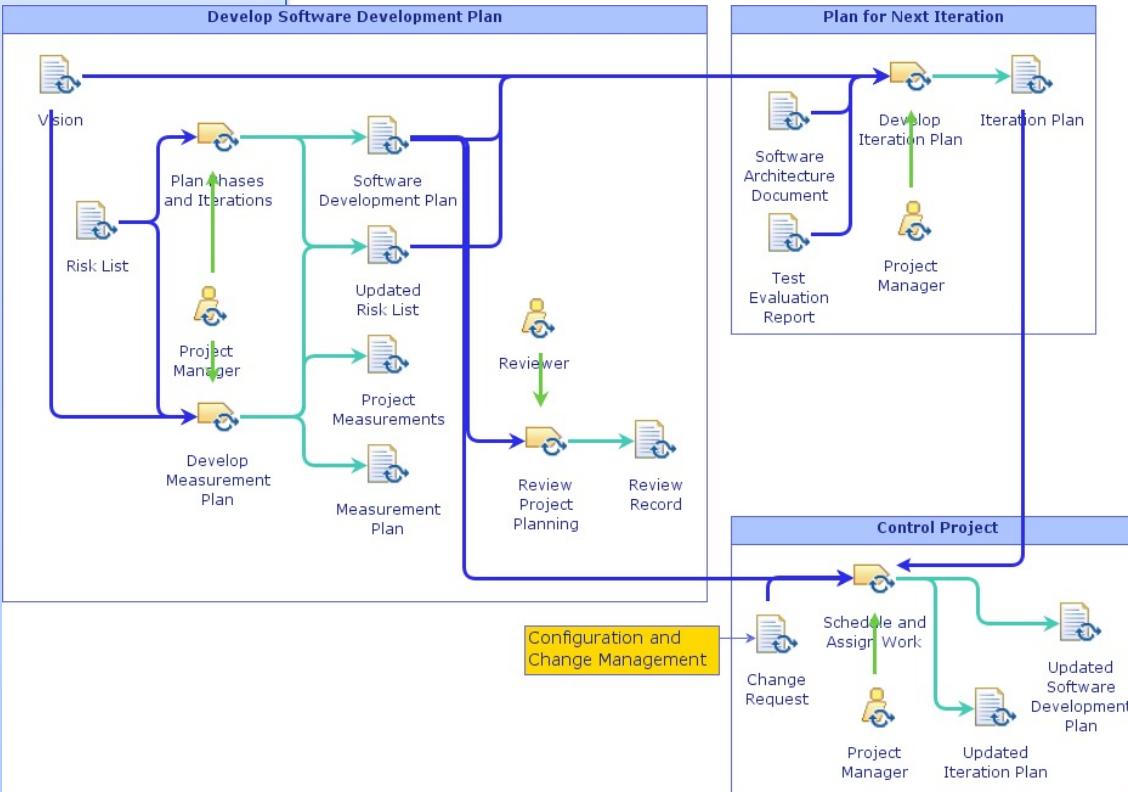
3. Processus vs. Projet

4. Outils

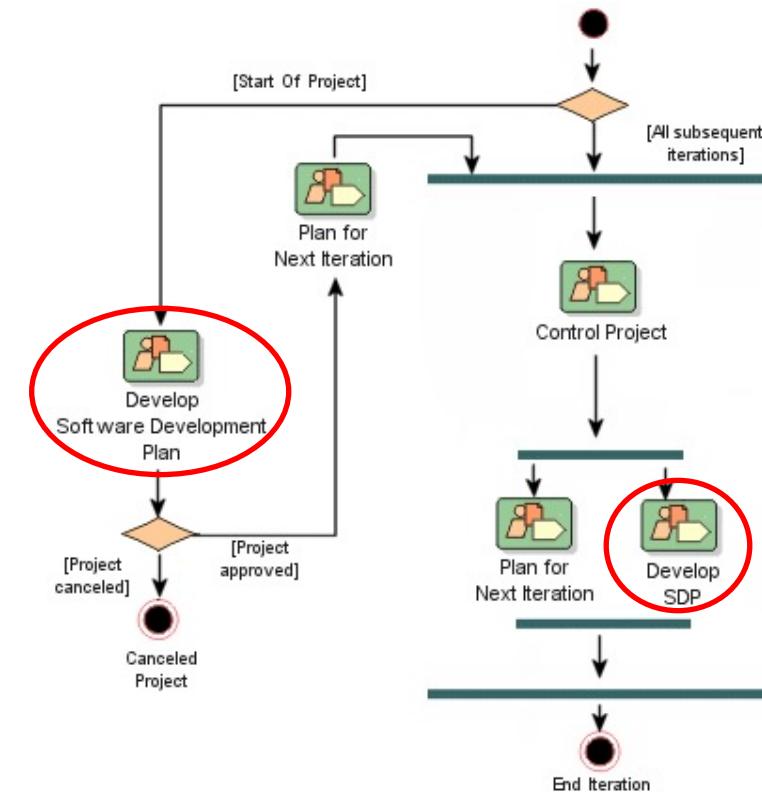
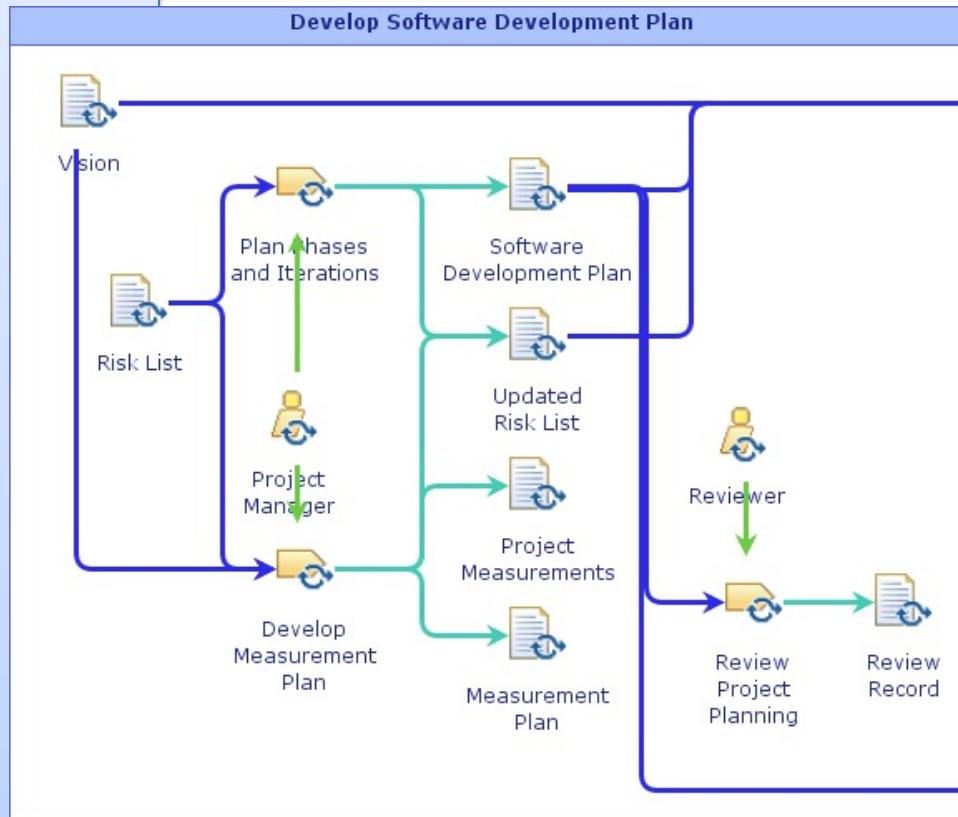
- **Relation client:** Pourquoi fait-on le projet ?
- **Budget:** Comment les ressources seront-t-elles dépensées ?
- **Collaboration de projet:** Comment partager des documents? Comment communiquer avec les différentes équipes?
- **Etc.**



Gestion de projet selon l'UPEDU



Gestion de projet selon l'UPEDU



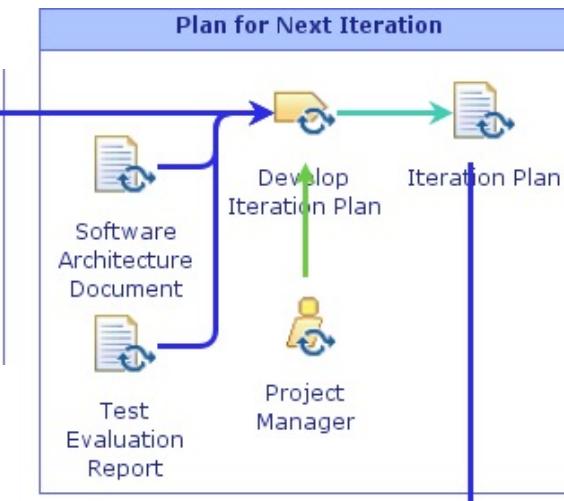
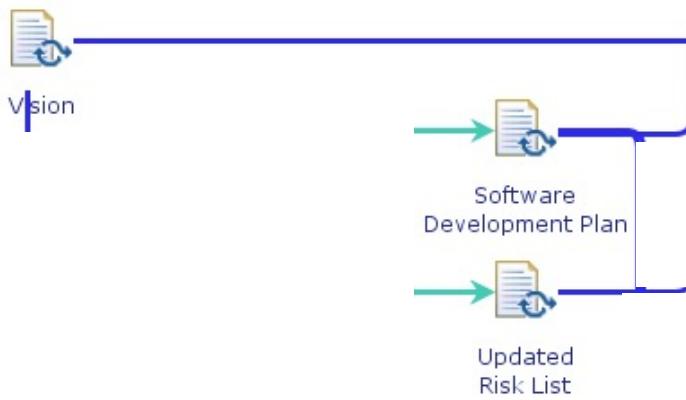
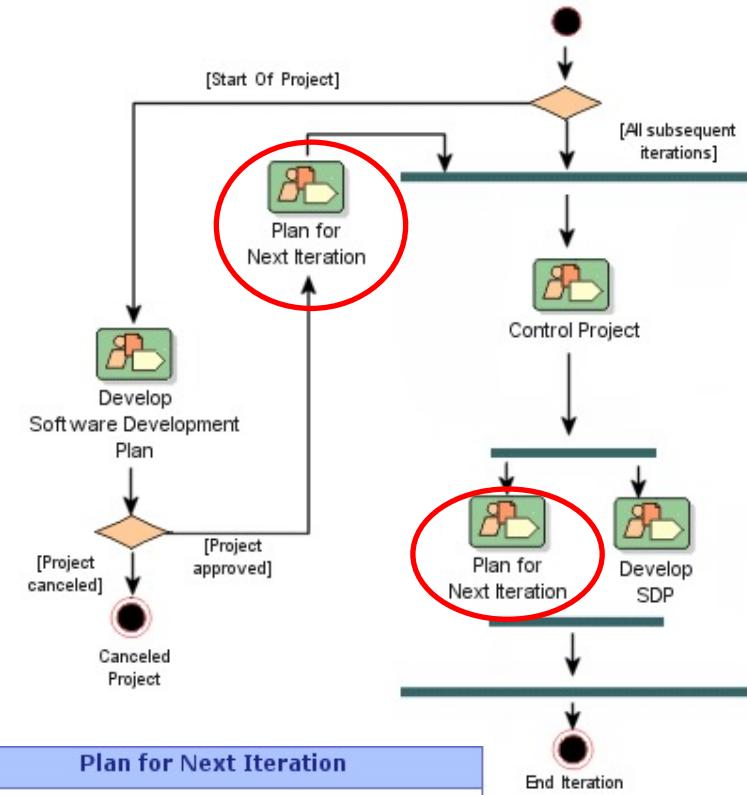
Develop Software Development Plan

- Plan Phases and Iterations:** Définition des ressources, jalons et durée des itérations. Planification de haut niveau.
- Develop Measurement Plan:** Définition des buts du projet et des mesures qui évaluent si ces buts sont atteints.
- Review Project Planning:** Révision des artefacts de planification de projet.

Gestion de projet selon l'UPEDU

Plan for Next Iteration

- **Develop Iteration Plan:** Définir les tâches précises de la prochaine itération. Planification de bas niveau.

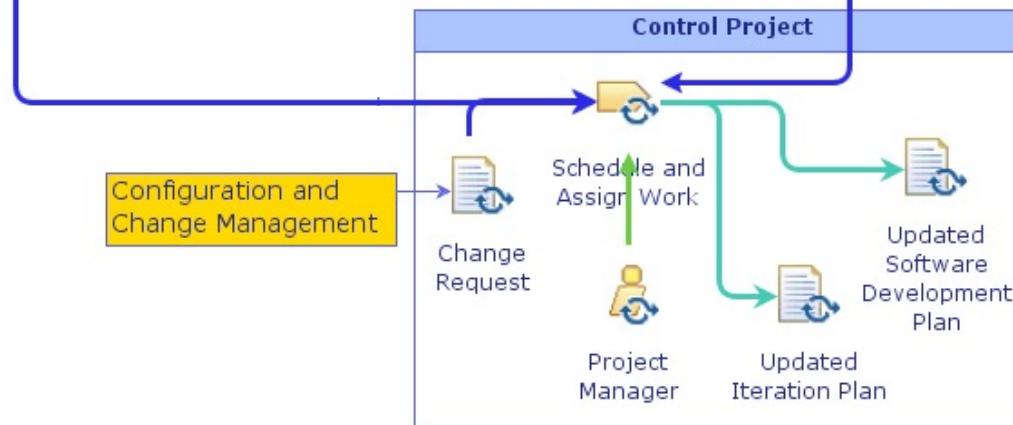
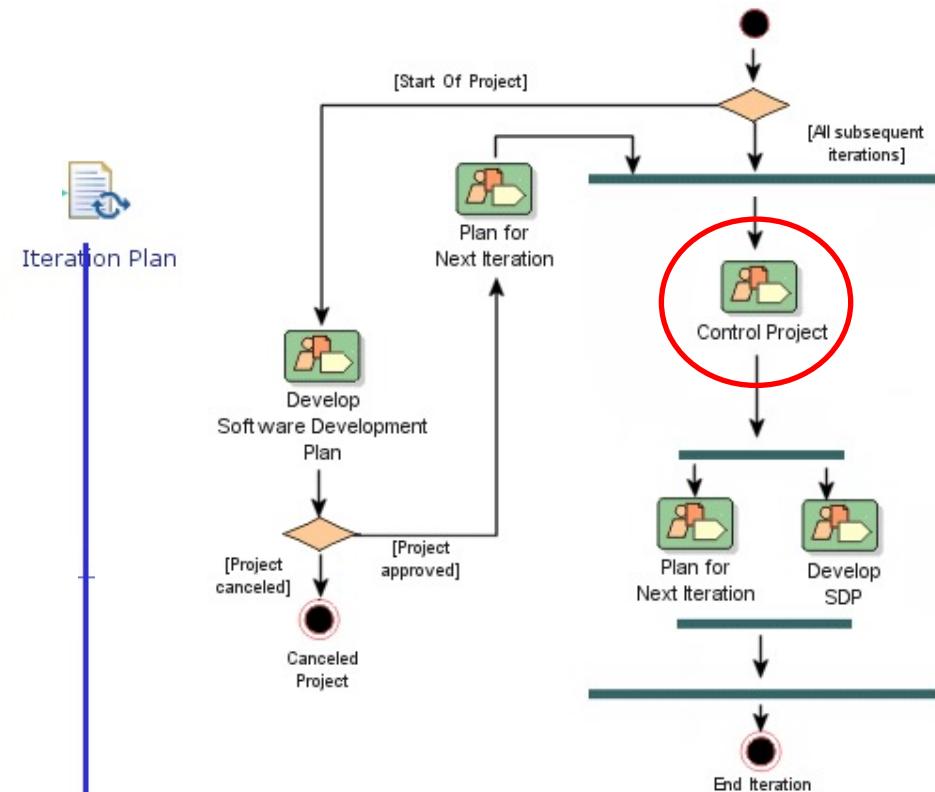


Gestion de projet selon l'UPEDU



Control Project

- Schedule and Assign Work:** Assigner les tâches et ressources aux personnes.



Artéfacts de la gestion de projet avec l'UPEDU

1. Position de la gestion de projet

2. Gestion de projet selon l'UPEDU

3. Processus vs. Projet

4. Outils

Plan de mesure

- Objectifs de la mesure;
- Métriques à saisir;
- Suivi des données

Plan d'itérations

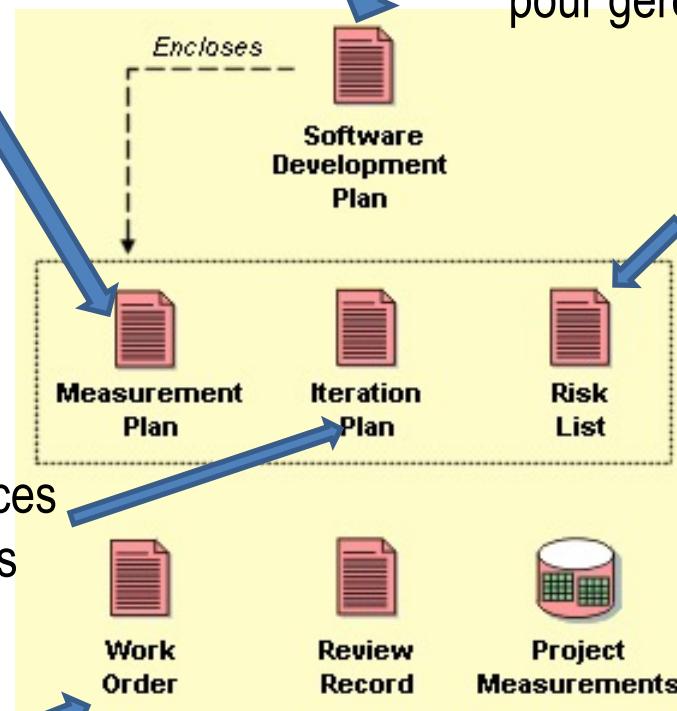
- Séquence temporelle de tâches (e.g. Gantt)
- Assignation des ressources
- Dépendances des tâches

Ordre de travail

Contrat interne entre le gestionnaire et les personnes assignées aux tâches

Plan de développement logiciel:

Ensemble complet d'artefacts contenant toute l'information pour gérer le projet.

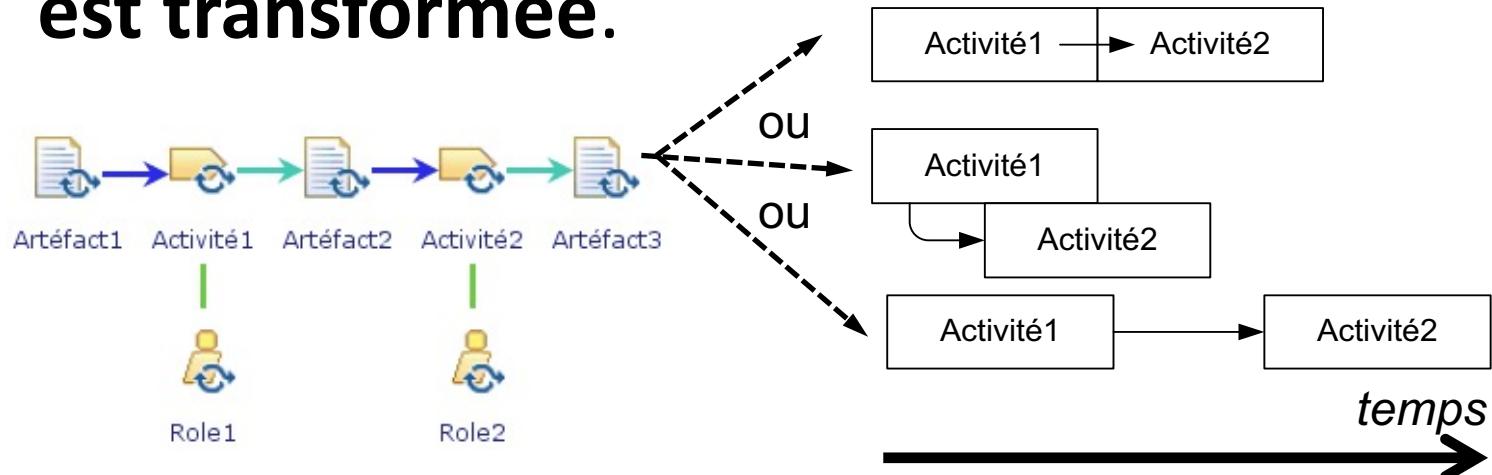


Mesures du projet:

Base de donnée contenant le répertoire des mesures.⁹

Différence entre processus et projet

- Processus: Ensemble d'**activités partiellement ordonnées** dont l'objectif est de présenter **comment l'information est transformée**.



- Projet: Ensemble de **tâches ordonnées** dont l'objectif est de **planifier l'utilisation des ressources**.

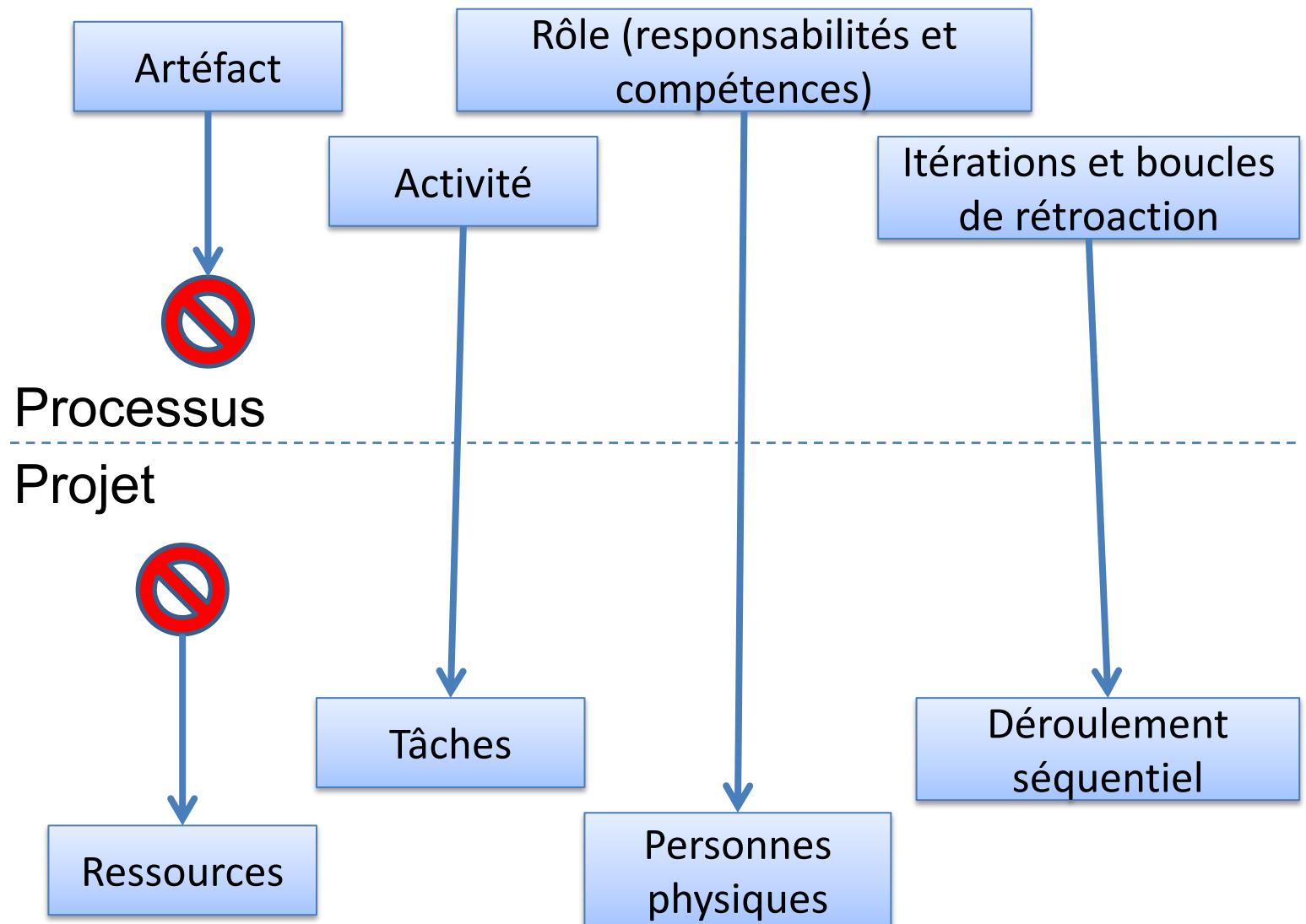
Processus vs. Projet

1. Position de la gestion de projet

2. Gestion de projet selon l'UPEDU

3. Processus vs. Projet

4. Outils



Outils de gestion de projet

Pour les gestionnaires

- **Diagrammes de Gantt:** utiles pour gestionnaires, trop complexes pour développeurs.

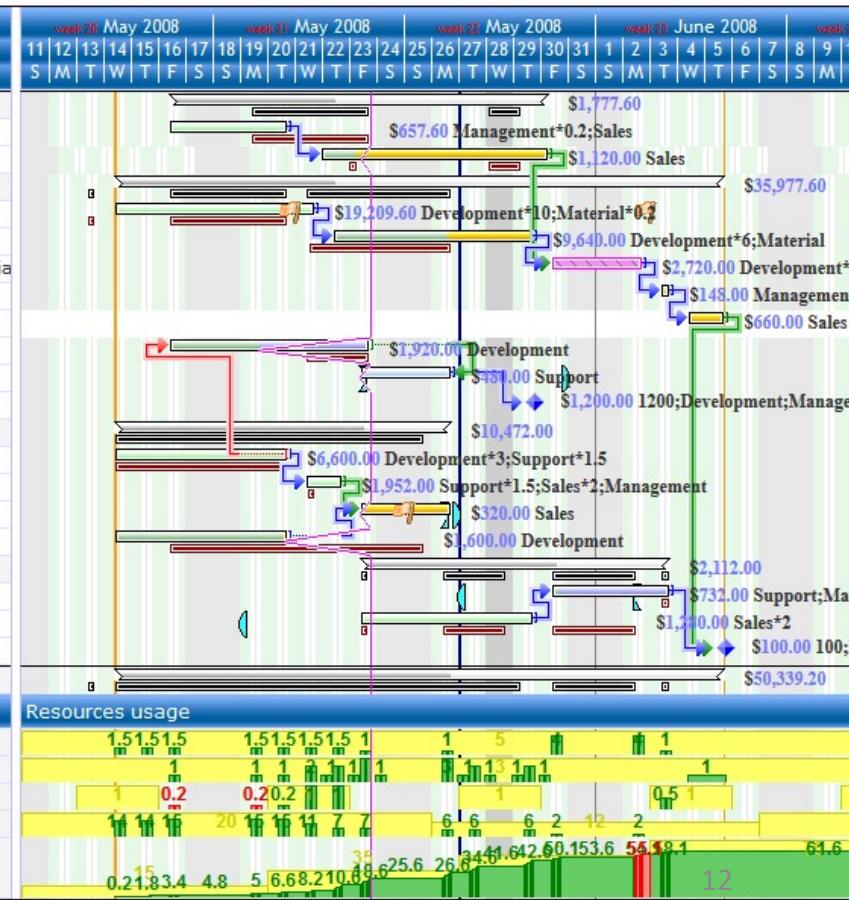
1. Position de la gestion de projet

2. Gestion de projet selon l'UPEDU

3. Processus vs. Projet

4. Outils

tion	Start	End	Dura tion	Com plete	Desce ndants	Ance stors	Calendar	Color
Task 1 (2)	May 16, 08	May 30, 08	72	47.5%				
S1	May 16, 08	May 20, 18	24	100%	2			
S2	May 21, 23	May 30, 08	56	25%	5	1	Night shi	
Task 2 (8)	May 14, 08	Jun 05, 18	128	39.4%				
S1	May 14, 08	May 21, 18	48	100%	4			
S2	May 22, 08	May 29, 18	40	50%	5	3		
S3	May 30, 08	Jun 02, 18	16	0%	6	2;4	Fuchsia	
S4	Jun 03, 09	Jun 03, 18	8	0%	7	5		
S5	Jun 04, 09	Jun 05, 18	33	0%	17	6	24 hours	
S6	May 16, 08	May 23, 18	48	30%	9ff+8h	11-2d		
S7	May 23, 08	May 26, 18	16	0%	10	8ff+8h		
M1	May 29, 18		0	0%	9			
Task 3 (4)	May 14, 08	May 26, 18	72	82.1%				
S1	May 14, 08	May 20, 18	40	100%	12:8-2d			
S2	May 21, 08	May 22, 18	16	100%	13	11		
S3	May 23, 08	May 26, 18	16	0%		12:14+2		
Task 4 (3)	May 23, 08	Jun 03, 18	56	65.7%				
S1	May 30, 08	Jun 03, 18	24	20%	17	16		
S2	May 23, 08	May 29, 18	32	100%	15			
M2	Jun 05, 18		0	0%	7:15			
Day (17)	May 14, 08	Jun 05, 18	128	54.5%				
nce	Unit price	Availability	Peak	Type	Total	Price		
	30.00	5	4	wrk	172.00	5160.00		
	20.00	3	3	wrk	225.00	4500.00		
ment	37.00	w#1/4/201	1	wrk	24.80	917.60		
ment	40.00	20/5/26/20	15	wrk	960.00	38400.00		
	1.00	15:w#5/20	1	mat	61.60	61.60		



Source : TreeGrid, outil pour présenter des diagrammes de Gantt en pages dynamiques HTML.

Outils de gestion de projet

Pour les développeurs

- **Jira/GitHub Issues:** utiles pour les développeurs, trop complexes pour les autres (clients, utilisateurs, public, etc.).

The screenshot shows a GitHub Issues page with the following details:

- Filters:** is:open is:issue
- Count:** 104 Open, 9,660 Closed
- Columns:** Author, Labels, Milestones, Assignee, Sort
- Pull Requests:**
 - #13989: .form-group-sm .form-group-lg shrink textarea (confirmed, css) - opened 11 hours ago by limitstudios, v3.2.1, 4 comments
 - #13987: Tooltip unnecessarily breaks into multiple lines when positioned to the right (confirmed, js) - opened 15 hours ago by hnrch02, v3.2.1, 0 comments
 - #13981: Tooltip Arrows in Modal example facing wrong way (css) - opened a day ago by SDCore, 6 comments
 - #13978: Table improvement (css) - opened a day ago by Tjoosten, 0 comments
 - #13977: docs/dist files (docs) - opened 2 days ago by XhmikosR, v3.2.1, 7 comments
 - #13976: Potential solution to #4647 (js) - opened 2 days ago by julioarmandof, 4 comments
 - #13974: Bootstrap site: right-hand navigation text becomes rasterized after scrolling (css, docs) - opened 2 days ago by mg1075, v3.2.1, 4 comments
 - #13972: Dropdown toggle requires two clicks (js) - opened 2 days ago by Kizmar, 1 comment

Outils de gestion de projet

Pour la public

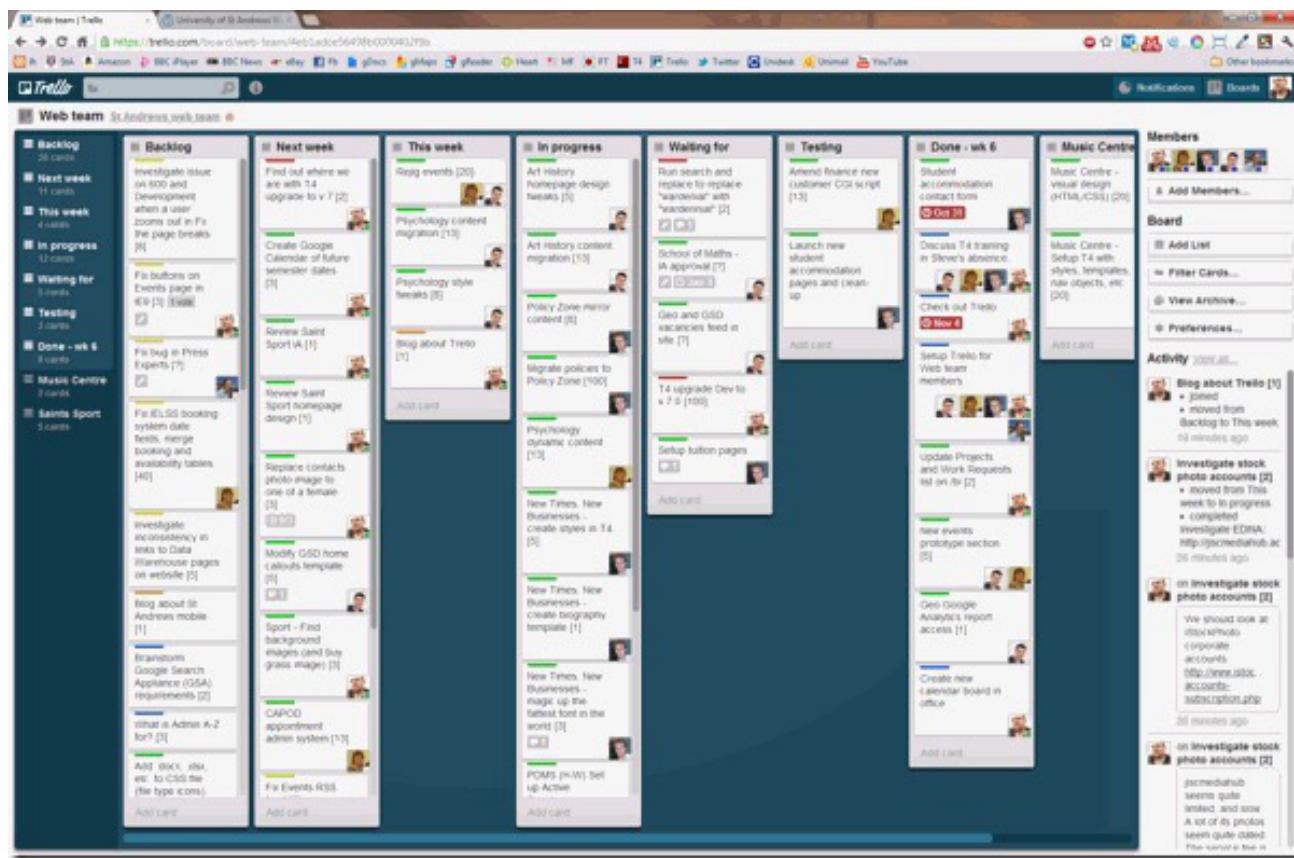
- **Trello/Kanbanchi:** utiles pour les non-développeurs, trop simple pour la gestion à long terme, peu de mémoire.

1. Position de la gestion de projet

2. Gestion de projet selon l'UPEDU

3. Processus vs. Projet

4. Outils



Source : Trello, tel qu'utilisé par l'équipe web de l'université de St-Andrews

LOG3000

Processus du génie logiciel

- 1. Concepts des diagrammes de Gantt
- 2. Contraintes de date
- 3. Surutilisation des ressources
- 4. Parallélisation
- 5. Marges et jalons

Discipline de gestion de projet:
Diagrammes de Gantt

Concepts de Diagrammes de Gantt

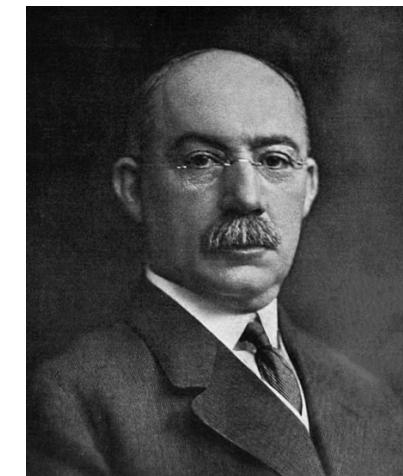
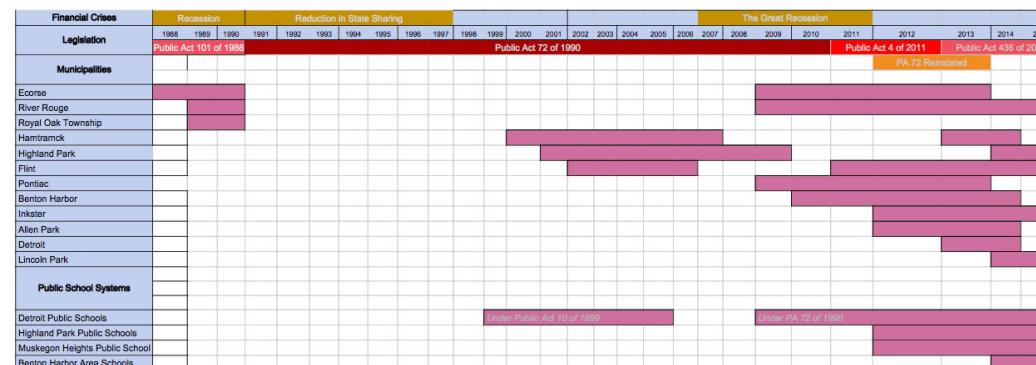
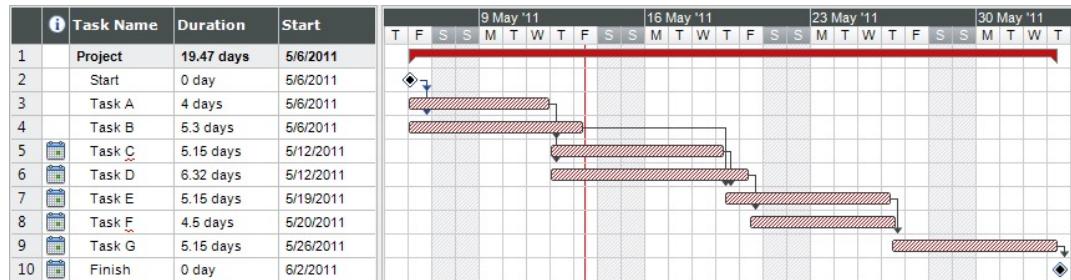
1. Concepts des diagrammes de Gantt

2. Contraintes de date

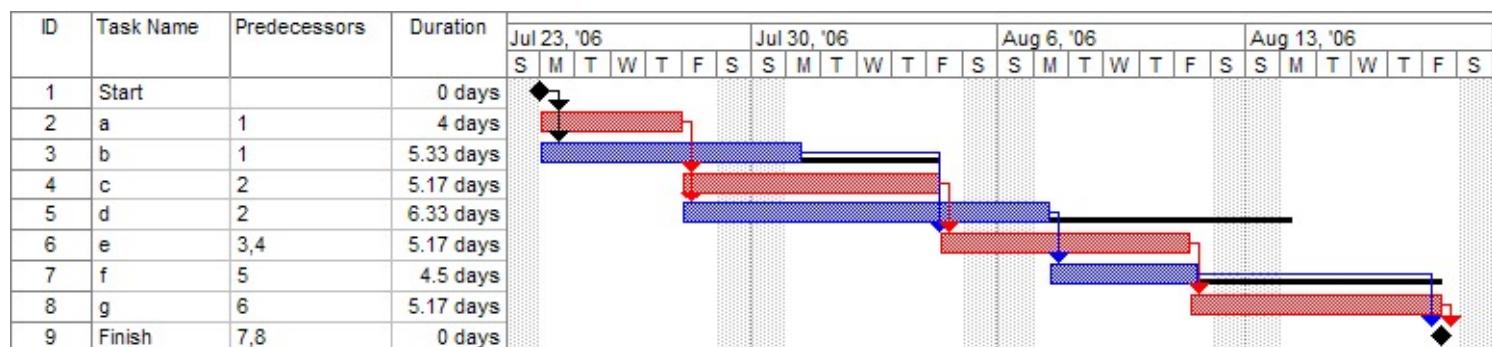
3. Surutilisation des ressources

4. Parallélisation

5. Marges et jalons



Henry Gantt
(1861-1919)
Inventeur du
diagramme de Gantt



Contraintes de date

- À utiliser seulement lorsque nécessaire.
 - Ex.: Lorsque la date est connue avant le début du projet.
- Pourquoi? Il faut être capable de changer rapidement la planification.
 - Ex.: Une tâche a pris du retard.
 - Ex.: Un jalon doit être devancé.
- Les outils logiciels permettent de recalculer automatiquement le projet.
 - Trop de contraintes bloquent ce calcul et force de changer les dates à la main → Ouch !

1. Concepts des diagrammes de Gantt

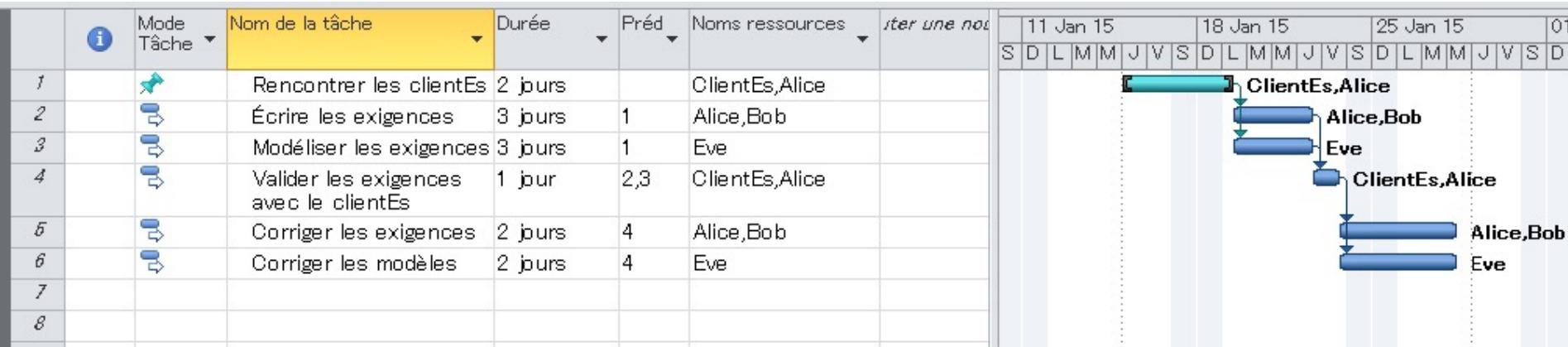
2. Contraintes de date

3. Surutilisation des ressources

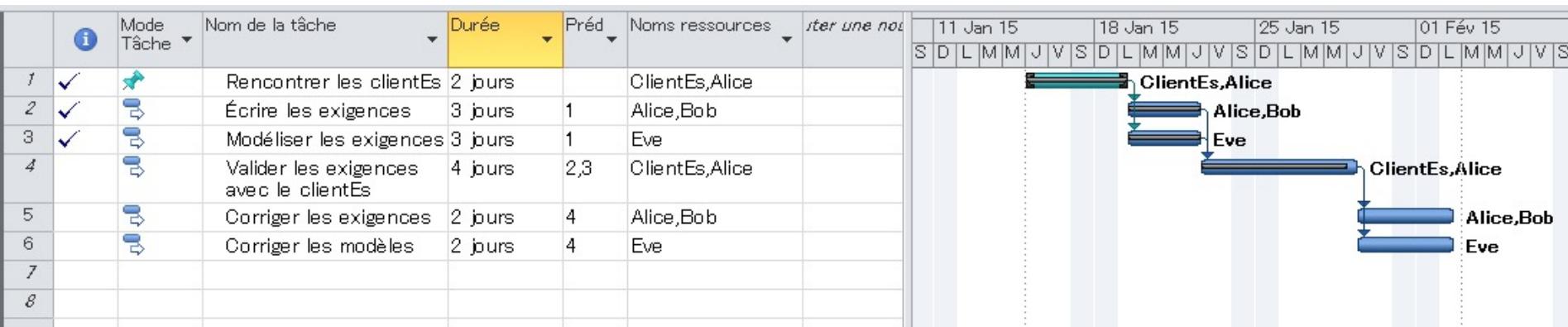
4. Parallélisation

5. Marges et jalons

Exemple: Planification initiale



... sauf que la validation des exigences a pris plus de temps que prévu ... pas de problèmes !



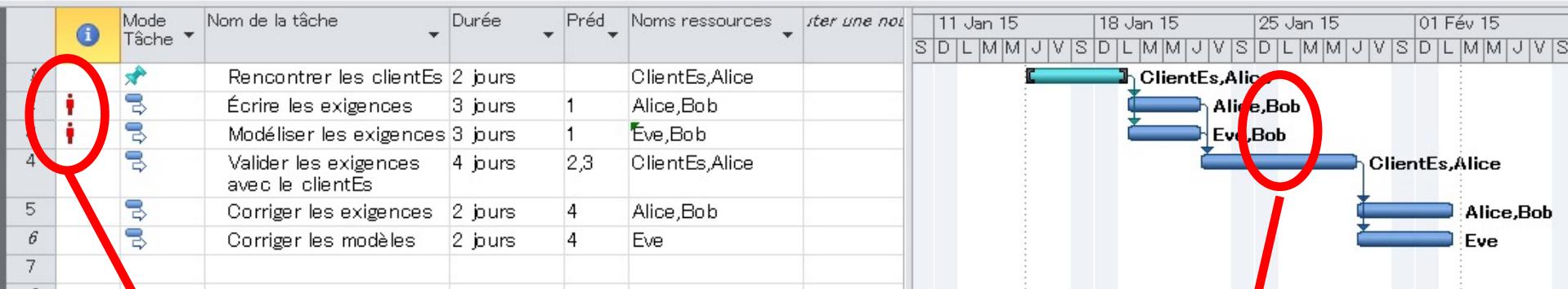
On change la durée de la tâche et toutes les dépendances se recalculent automatiquement.

Surutilisation des ressources

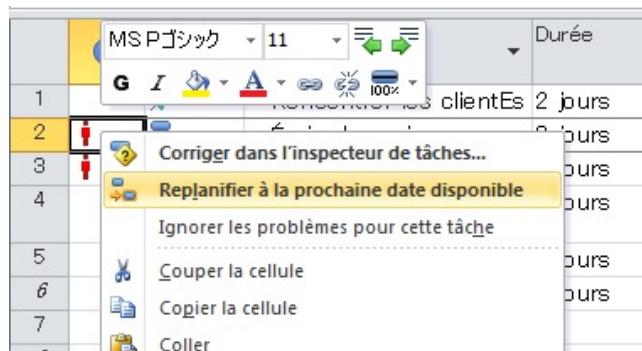
- Dans un gros projet qui change souvent, il peut être difficile de se rendre compte qu'on assigne une personne à deux tâches en même temps.
- Les outils logiciels nous avertissent quand ça arrive, ce qui nous permet de corriger le tir.
 - Il y a généralement des outils qui permettent de replacer la tâche à un moment où les ressources seront disponibles.
 - Sinon, on peut changer l'assignation à une autre ressource disponible.
 - ... ou bien faire travailler la personne deux fois plus ...



Exemple: Planification initiale



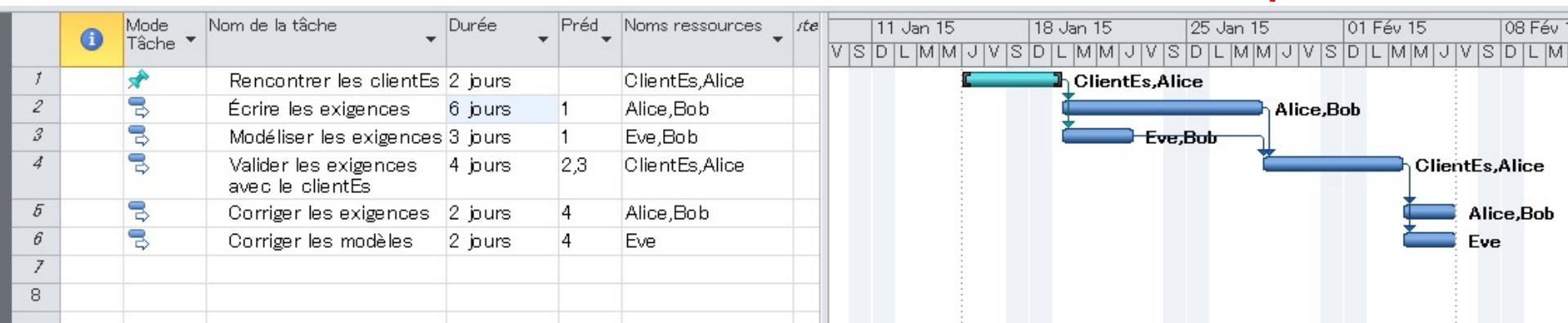
Indicateur de ressource surutilisée



Ah oui, Bob fait deux tâches en même temps ... pauvre Bob !

Bouton droit, replanifier ...

La tâche va se terminer plus tard.



Parallélisation

- Paralléliser permet de gérer le risque.
 - Si une tâche bloque, au moins on peut faire autre chose en parallèle.
- Paralléliser les tâches demande de paralléliser l'utilisation des ressources.
 - Une personne ne devrait pas faire deux tâches en même temps.
- La planification parallèle des tâches doit donc aussi tenir en compte que ces tâches doivent être faites par des personnes différentes.
 - Duh, mais ça demande quand même une planification plus élaborée.

1. Concepts des diagrammes de Gantt

2. Contraintes de date

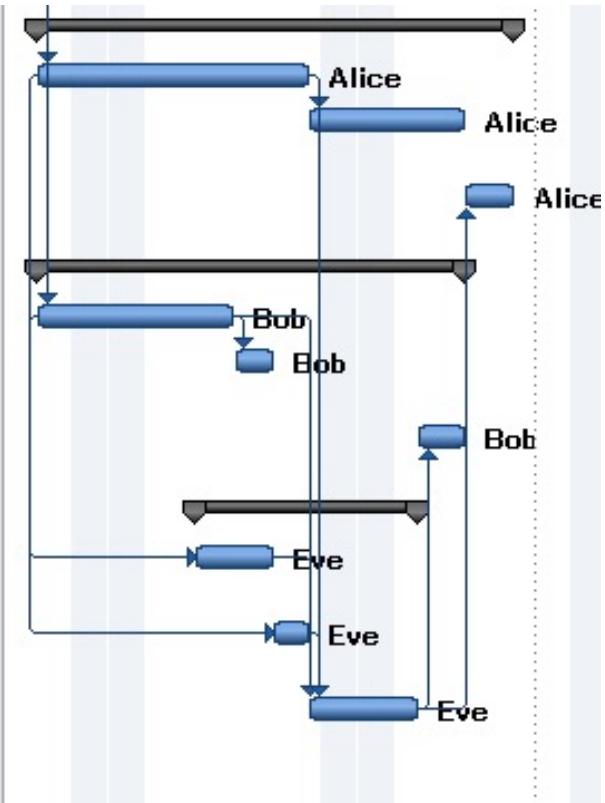
3. Surutilisation des ressources

4. Parallélisation

5. Marges et jalons

Exemple: Modules codés en parallèle, tests unitaires faits par quelqu'un d'autre.

9		Module A	8 jours		
10		Coder le module A	5 jours	5,6	Alice
11		Documenter le module A	2 jours	10	Alice
12		Corriger les erreurs du module A	1 jour	20	Alice
13		Module B	7 jours		
14		Coder le module B	3 jours	5,6	Bob
15		Documenter le module B	1 jour	14	Bob
16		Corriger les erreurs du module B	1 jour	20	Bob
17		Tests unitaires	4 jours		
18		Écrire les tests unitaires du module	2 jours	10DD+2 jours	Eve
19		Écrire les tests unitaires du module	1 jour	14DD+2 jours	Eve
20		Documenter les résultats des tests unitaires	1 jour	10,14,18	Eve



Problèmes :

- Plus difficile à planifier,
- Plus difficile d'éviter des temps morts.

Marges et jalons

- Un jalon doit être lié à au moins une tâche.
 - Cela nous permet de comprendre à quelle(s) tâche(s) le jalon fait référence.
- Les tâches ne devraient pas se terminer pile sur le jalon → Il faut se laisser des marges de manœuvre en cas d'imprévus.
 - $\approx 5\%$ pour un projet où tout est connu (peu risqué).
 - $\approx 25\%$ pour un projet avec beaucoup d'incertitudes (risqué).

1. Concepts des diagrammes de Gantt

2. Contraintes de date

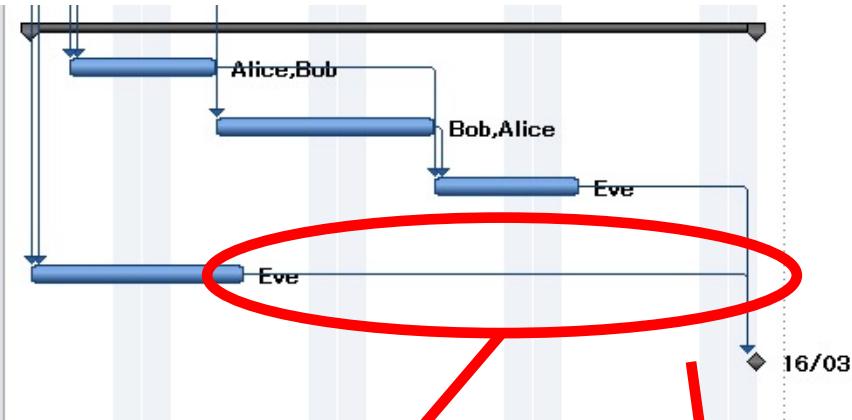
3. Surutilisation des ressources

4. Parallélisation

5. Marges et jalons

Exemple: Tests et livraison du produit

22		Tests	15,75 jour		
23		Exécuter les tests en boîte noire	3 jours	12,16	Alice,Bob
24		Exécuter les tests en boîte blanche	5 jours	12,16	Bob,Alice
25		Corriger les bogues trouvés par les tests	3 jours	23,24	Eve
26		Finaliser la documentation du produit	5 jours	11,15	Eve
27	CALENDAR	Livraison du produit	0 jour	25,26	



On comprend que le jalon est lié à la correction des bogues et à la finalisation de la doc.

Contrainte calendaire pour le jalon ...

... qui nous laisse environ 7 jours de marge. Acceptable pour un projet de 65 jours (10% de marge).

LOG3000

Processus du génie logiciel

1. Concepts de la gestion de risques
2. Stratégies de gestion de risques
3. Le « Iron Triangle »

Discipline de gestion de projet:
Gestion de risques

Gestion de risques

- Risque: Variable dont la valeur peut mettre en danger ou réduire le succès d'un projet.
 - Tout ce qui peut entraver la réussite du projet.



1. Concepts de la gestion de risques

2. Stratégies de gestion de risques

3. Le « Iron Triangle »

Concepts de la gestion de risques

- Type de risque
 - **Direct/indirect:** Degré auquel l'équipe a contrôle sur le risque.
 - Exemple direct: *Nous n'avons jamais programmé d'IA par le passé* → on peut engager un expert, faire de la formation, se donner des marges de manœuvre, etc.
 - Exemple indirect: *Les clients ne savent pas ce qu'ils et elles veulent* → on peut choisir une approche qui minimise ce risque, mais on reste à leur merci ...
 - **Connu/inconnu:** Degré auquel le risque est évident dès le début du projet.
 - Exemple connu: *La date de livraison ne peut pas être repoussée* → Projet 3.
 - Exemple inconnu: *La librairie choisie ne fonctionne pas tel que la documentation le suggérait* → Projet 3 et XCode ...

1. Concepts de la gestion de risques

2. Stratégies de gestion de risques

3. Le « Iron Triangle »

Concepts de la gestion de risques

- Attributs des risques
 - **Probabilité** d'occurrence: Faible, moyen, élevé, ...
 - **Impact** potentiel: Mineur, modéré, grave, ...
 - **Niveau** de risque = Probabilité * Impact

1. Concepts de la gestion de risques

2. Stratégies de gestion de risques

3. Le « Iron Triangle »

Risque	Prob. (1 – 5)	Impact (1 – 5)	Niveau = P * I	Stratégie
Une personne annule le cours en milieu de projet.				
Les bibliothèques de XCode et d'iPad ne fonctionnent pas comme prévu.				
Le dépôt git est en maintenance et on ne peut pas « push » les modifications.				

Stratégies de gestion de risques

- **Proactives:** Éviter, Mitiger, Transférer
- **Réactives:** Contourner
- **« Sans-action »:** Accepter (mais surveiller)

1. Concepts de la
gestion de
risques

2. Stratégies de
gestion de
risques

3. Le « Iron
Triangle »

Stratégies de gestion de risques

1. Concepts de la gestion de risques

2. Stratégies de gestion de risques

3. Le « Iron Triangle »

Proactives:

- **Éviter:** Réorganiser pour éliminer le risque.
 - Ex.: Ne pas mettre le jeu en 3D.
- **Mitiger:** Réduire la probabilité ou l'impact.
 - Ex.: Faire des itérations plus courtes afin de corriger le tir rapidement et minimiser l'impact.
- **Transférer:** Sous-contracter la partie risquée.
 - Ex.: Embaucher un studio pour développer la partie 3D du jeu.

Réactives :

- **Contourner:** Planifier un plan ‘B’.
 - Ex.: Si le prototype avec la librairie X ne marche pas, réessayer avec la librairie Y.



Stratégies de gestion de risques

« Sans-action » :

- **Accepter:** Vivre avec (mais surveiller).
 - Pour des risques où le niveau est faible.

1. Concepts de la
gestion de
risques

2. Stratégies de
gestion de
risques

3. Le « Iron
Triangle »

Exemple: Poker night !

Soirée souvent arrosée... comment m'assurer que je peux revenir à la maison ?

Risque: Je bois trop que je ne peux pas rentrer chez moi.

- **Éviter:** Je n'irai pas à la soirée.
- **Mitiger:** Je ne prendrai qu'un seul verre.
- **Transférer:** Je vais nommer un chauffeur désigné.
- **Contourner:** Si nécessaire, j'appellerai un taxi.
- **Accepter:** Il est peu probable que je boive beaucoup, alors j'y vais en voiture.



Gestion de risques: Le « Iron Triangle »

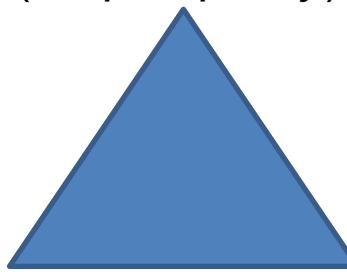
1. Concepts de la gestion de risques

2. Stratégies de gestion de risques

3. Le « Iron Triangle »

- **Triangle traditionnel:**

Portée plus grande, qualité meilleure
(*scope, quality*)



Diminuer les coûts des outils et des matières premières
(*cost*)

Échéancier plus serré
(*schedule*)

- **Principe:** Si on priorise un des pôles, on s'éloigne automatiquement des deux autres.

Exemple traditionnel du « Iron Triangle »

1. Concepts de la gestion de risques

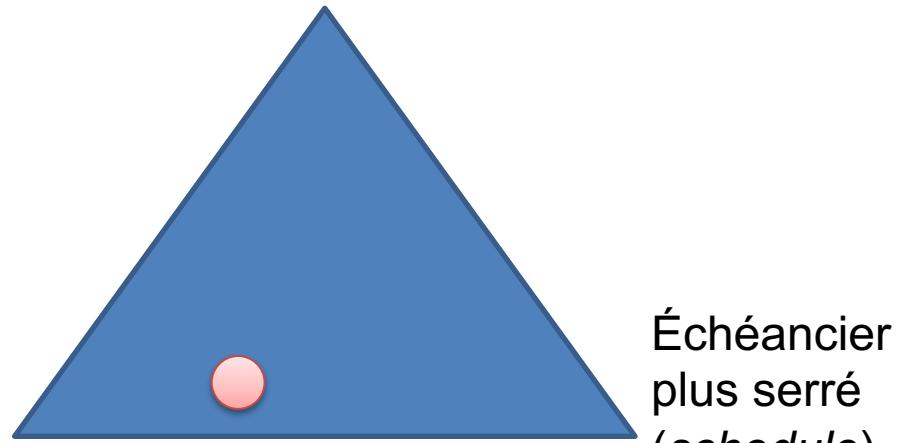
2. Stratégies de gestion de risques

3. Le « Iron Triangle »

- *Je veux produire du pain en grande quantité le moins cher possible.*

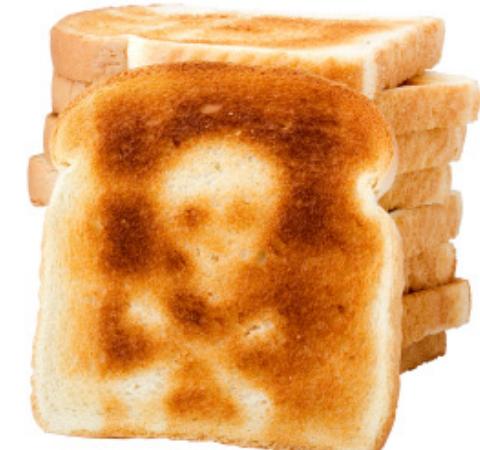
Portée plus grande, qualité meilleure (scope, quality)

Diminuer le coût des outils et matières premières (cost)



Échéancier plus serré (schedule)

- Conséquence: je ne peux produire qu'une sorte de pain (portée limitée) et sa qualité sera médiocre.



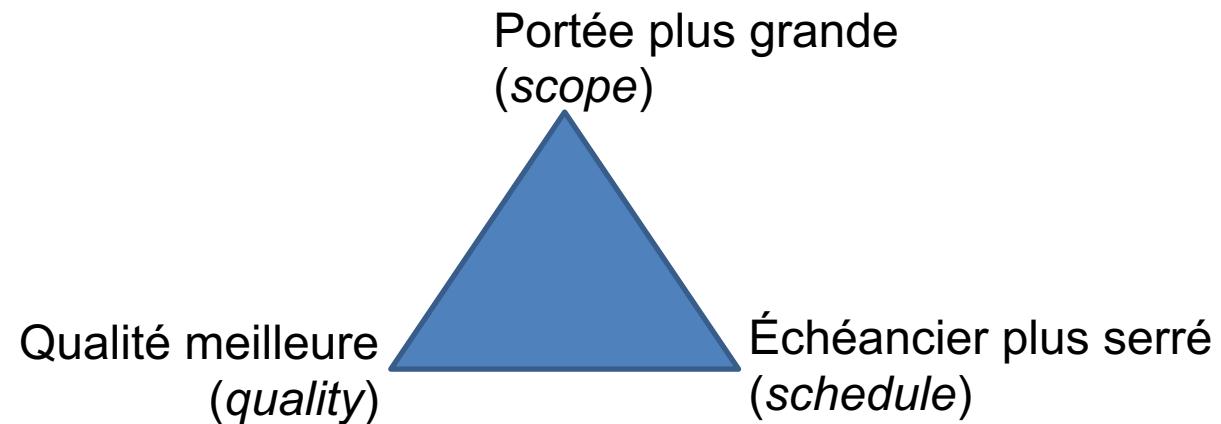
Gestion de risques: Le « Iron Triangle »

1. Concepts de la gestion de risques

2. Stratégies de gestion de risques

3. Le « Iron Triangle »

- **Adaptation au génie logiciel:**



- **Principe:** Si on priorise un des pôles, on s'éloigne automatiquement des deux autres.

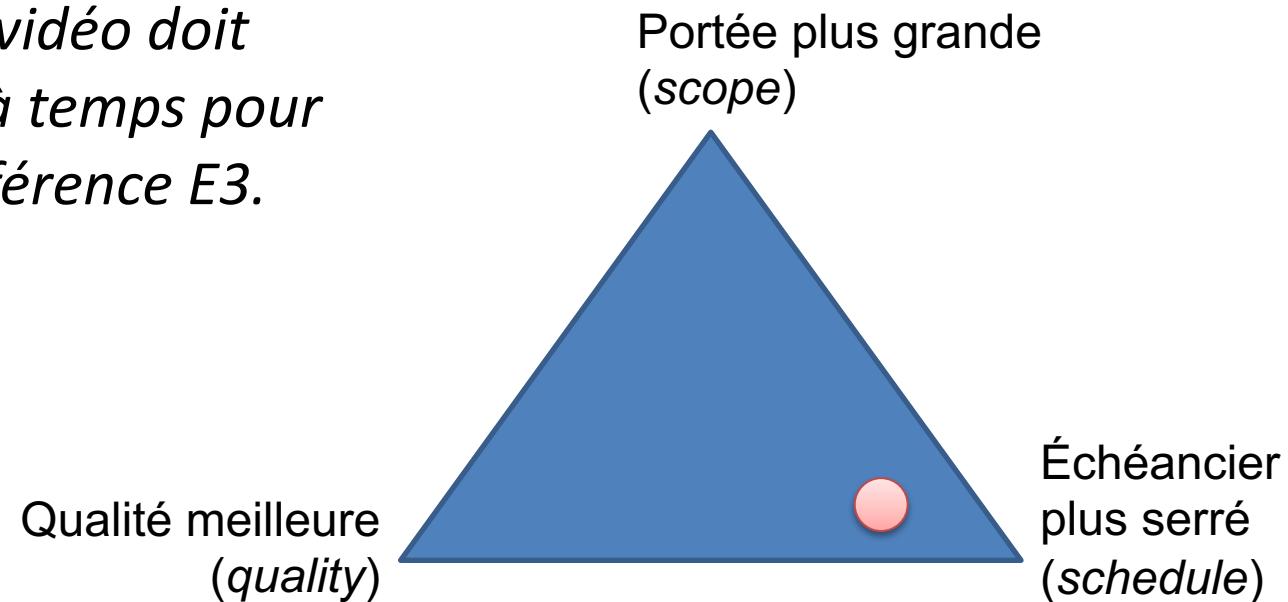
Exemple logiciel du « Iron Triangle »

1. Concepts de la gestion de risques

2. Stratégies de gestion de risques

3. Le « Iron Triangle »

- *Le jeu vidéo doit sortir à temps pour la conférence E3.*



- Conséquence: je vais couper sur les fonctionnalités (scope), sur les bonnes pratiques de développement et sur les tests (quality).



LOG3000

Processus du génie logiciel

1. Introduction
2. Analyse qualitative de processus
3. Analyse quantitative de processus

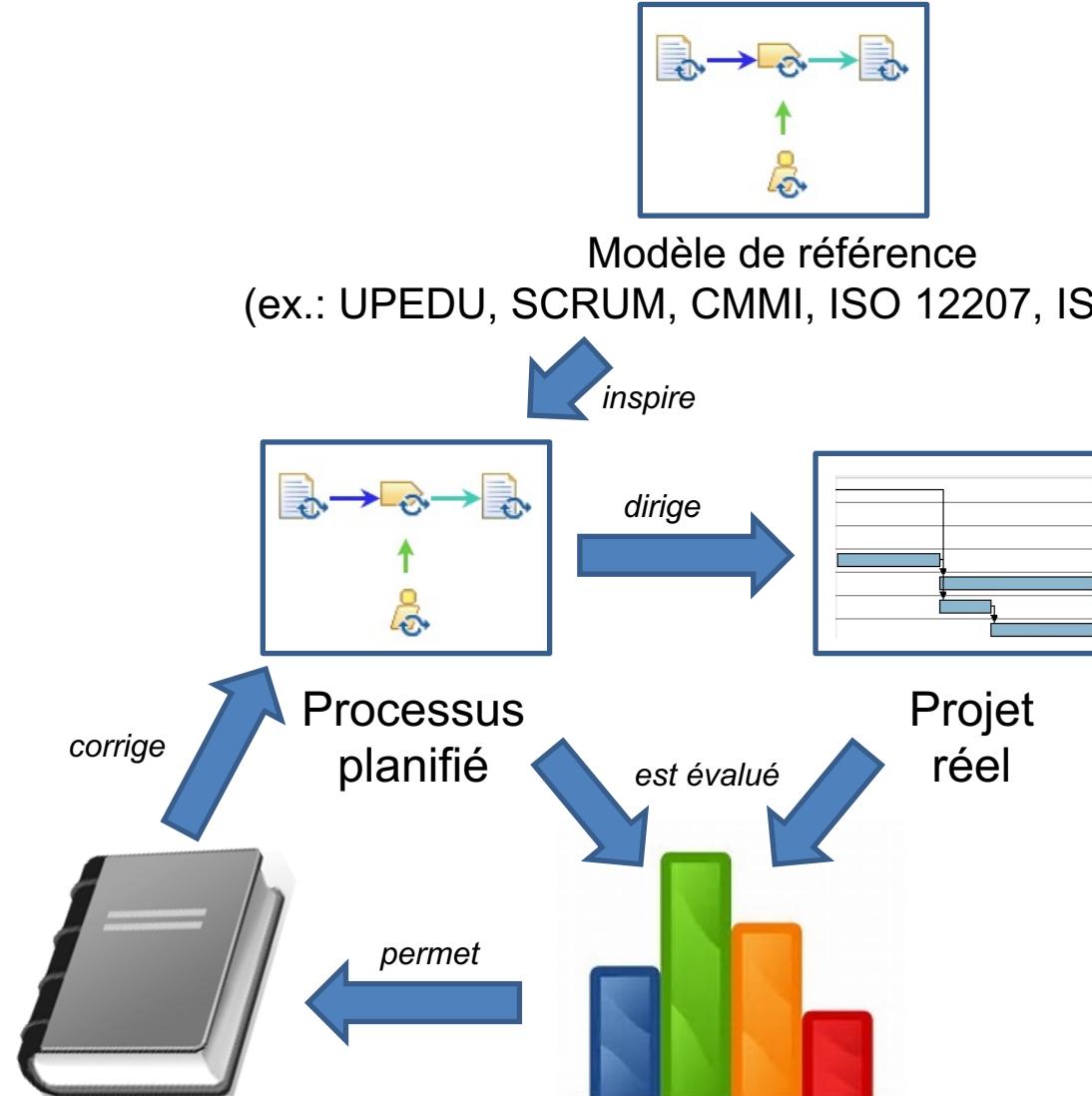
Analyse de processus

Pourquoi analyser les processus ?

1. Introduction

2. Analyse qualitative de processus

3. Analyse quantitative de processus



Rapport d'évaluation de la qualité du processus

Approche d'analyse
(ex.: CMMI, ISO 15504)

Résumé de l'analyse de processus

1. Introduction

2. Analyse qualitative de processus

3. Analyse quantitative de processus

- L'analyse de processus vérifie les points suivants:
 - **Éléments de trop:** Pertinence des activités, rôles et artefacts présents.
 - **Éléments manquants:** Activités, rôles ou artefacts manquants.
 - **Complexité des éléments:** Activités, rôles ou artefacts trop complexes devant être scindés, ou bien trop simples et devant être fusionnés.
 - **Cycle de vie:** Validation du choix du cycle de vie, de la définition des phases et des jalons.
 - **Ordonnancement:** Validation des flots définis et de l'ordre des activités.
- L'analyse se base sur les données recueillies lors des exécutions (présente et passées) du processus.

Analyse de données: Qualitatif vs. quantitatif

1. Introduction

2. Analyse qualitative de processus

3. Analyse quantitative de processus

Aujourd'hui nous avons travaillé sur le module de transformation d'images, sans succès. Nous avons essayé plusieurs solutions trouvées en ligne, mais celles-ci ne fonctionnent pas avec la nouvelle version de la librairie.

Analyse qualitative



Conclusions

101023023789529386723
112890478129347909092
120938105785623763278
101023023789529386723
112890478129347909092
120938105785623763278
101023023789529386723
112890478129347909092
120938105785623763278
120938105785623763278

Analyse quantitative



Analyse qualitative de processus

1. Introduction
2. Analyse qualitative de processus
3. Analyse quantitative de processus

- Interprétation d'observations, souvent basée sur un jugement d'expert.
 - Ex.: Il ne reste que cinq bogues non résolus, mais d'après moi l'un d'entre eux est critique et il vaut mieux mettre une semaine de plus sur le développement.
- Très utilisé dans les sciences sociales, quand un contexte complexe influe sur les résultats.
 - Ex.: La quantité de bogues dans un logiciel peut être causée par : expertise, pression de finir à temps, client qui change d'idée, coéquipiers qui part en milieu de projet, etc.

Analyse qualitative de processus

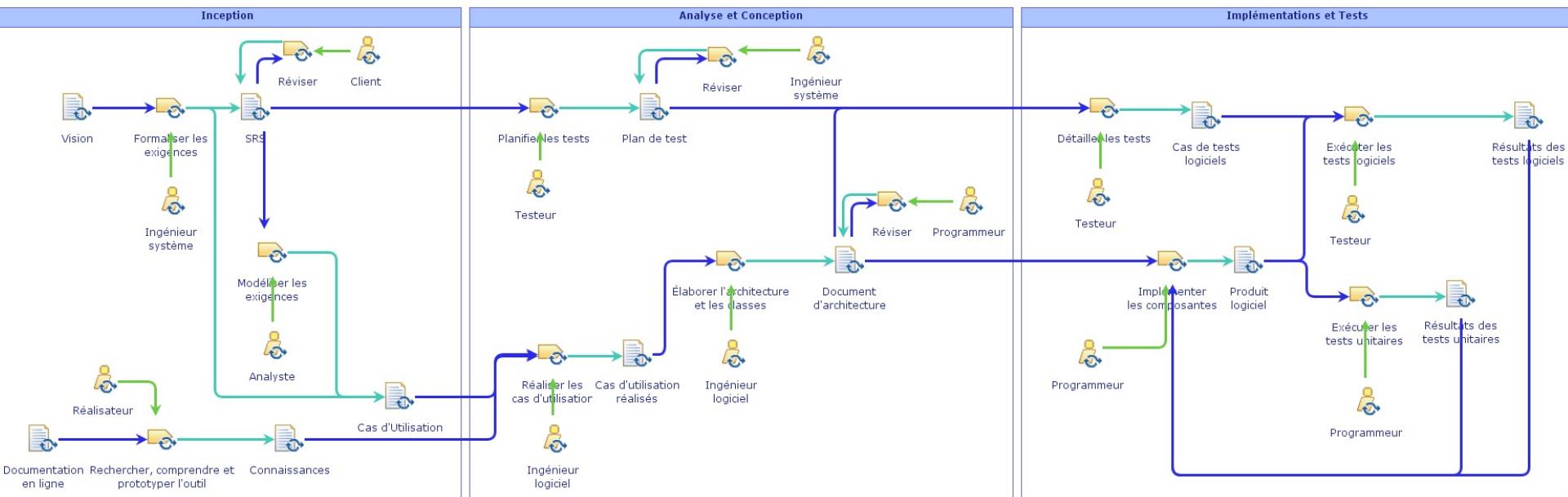
- 1. Introduction
- 2. Analyse qualitative de processus
- 3. Analyse quantitative de processus

- Exemple d'analyse qualitative de processus :
 - Évaluation faite sur la base d'un modèle de référence.
 - Ex.: Analyse CMMI/ISO du projet 3.
 - Discussion sur la pertinence d'une activité, d'un artéfact,
 - Ex.: Les revues de code nous prennent beaucoup de temps, est-ce qu'elles nous permettent de trouver des erreurs ?
 - Évaluation des compétences requises pour une activité,
 - Ex.: Cette activité demande une bonne compréhension des exigences, de Github, de Drupal et des makefile sur Linux. Est-ce faisable ?
 - Etc.

Décisions basées sur un jugement d'expert.

Exemple: Analyse qualitative basée sur jugement d'expert

- **Contexte:** Projet intégrateur de 4^e année, les étudiants ont un projet qui implique beaucoup d'incertitudes (du point de vue faisabilités techniques).
- Est-ce que le processus planifié suivant introduit des risques importants affectant le succès du développement et/ou la qualité des résultats?
Expliquez.



Analyse qualitative: Codification

1. Introduction
2. Analyse qualitative de processus
3. Analyse quantitative de processus

Permet de faire des analyses statistiques.

- 😊 Permet de transformer des données qualitatives difficilement comparables en données quantitatives catégorisées.
- 😢 La transformation de données qualitatives complexes en codes simples fait perdre beaucoup de détails : carte ≠ terrain.



Exemple: Codification

- Prenons les bogues suivants :

Bogues observés

Lorsque l'on part l'application, l'écran de démarrage (*splashscreen*) apparait en anglais au lieu du langage de la machine.

La fonctionnalité de transformation prend près de 30 secondes pour faire le travail, ce qui est inacceptable pour le client.

Le système plante quand on entre un nom d'utilisateur incorrect.

Le bouton de sauvegarde n'apparait pas sur l'interface graphique de l'administrateur.

Il est possible de faire une injection SQL à travers le champ nom d'utilisateur de la page web du système.

Il n'est pas possible défaire (*undo*) une action de copier/coller.

La fonctionnalité de transformation plante avec des fichiers de plus de 10 Ko.

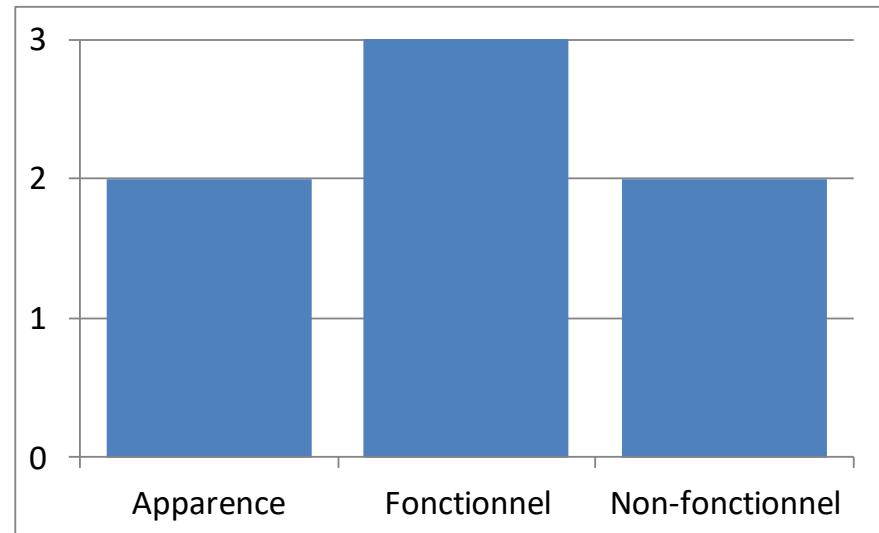
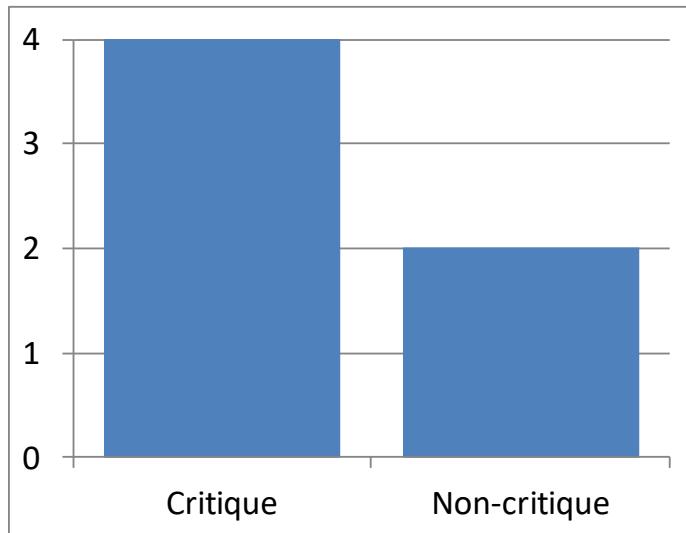
Exemple: Codification

- Codifications les bogues:

Bogues observés	Codes
Lorsque l'on part l'application, l'écran de démarrage (<i>splashscreen</i>) apparait en anglais au lieu du langage de la machine.	Apparence, Non-critique.
La fonctionnalité de transformation prend près de 30 secondes pour faire le travail, ce qui est inacceptable pour le client.	Non-fonctionnel, Critique.
Le système plante quand on entre un nom d'utilisateur incorrect.	Critique, Fonctionnel.
Le bouton de sauvegarde n'apparait pas sur l'interface graphique de l'administrateur.	Apparence, Critique.
Il est possible de faire une injection SQL à travers le champ nom d'utilisateur de la page web du système.	Non-fonctionnel, Critique.
Il n'est pas possible défaire (<i>undo</i>) une action de copier/coller.	Fonctionnel, Non-critique.
La fonctionnalité de transformation plante avec des fichiers de plus de 10 Ko.	Fonctionnel, Critique.

Exemple: Codification

- À partir des bogues observés et codifiés, il est possible de monter deux systèmes de codification.
- Cela nous permet de monter les graphes suivants :



- 😊 Cela permet d'avoir une vue d'ensemble de l'état du logiciel.
- 😢 On a évacué beaucoup de détails dans notre catégorisation ...

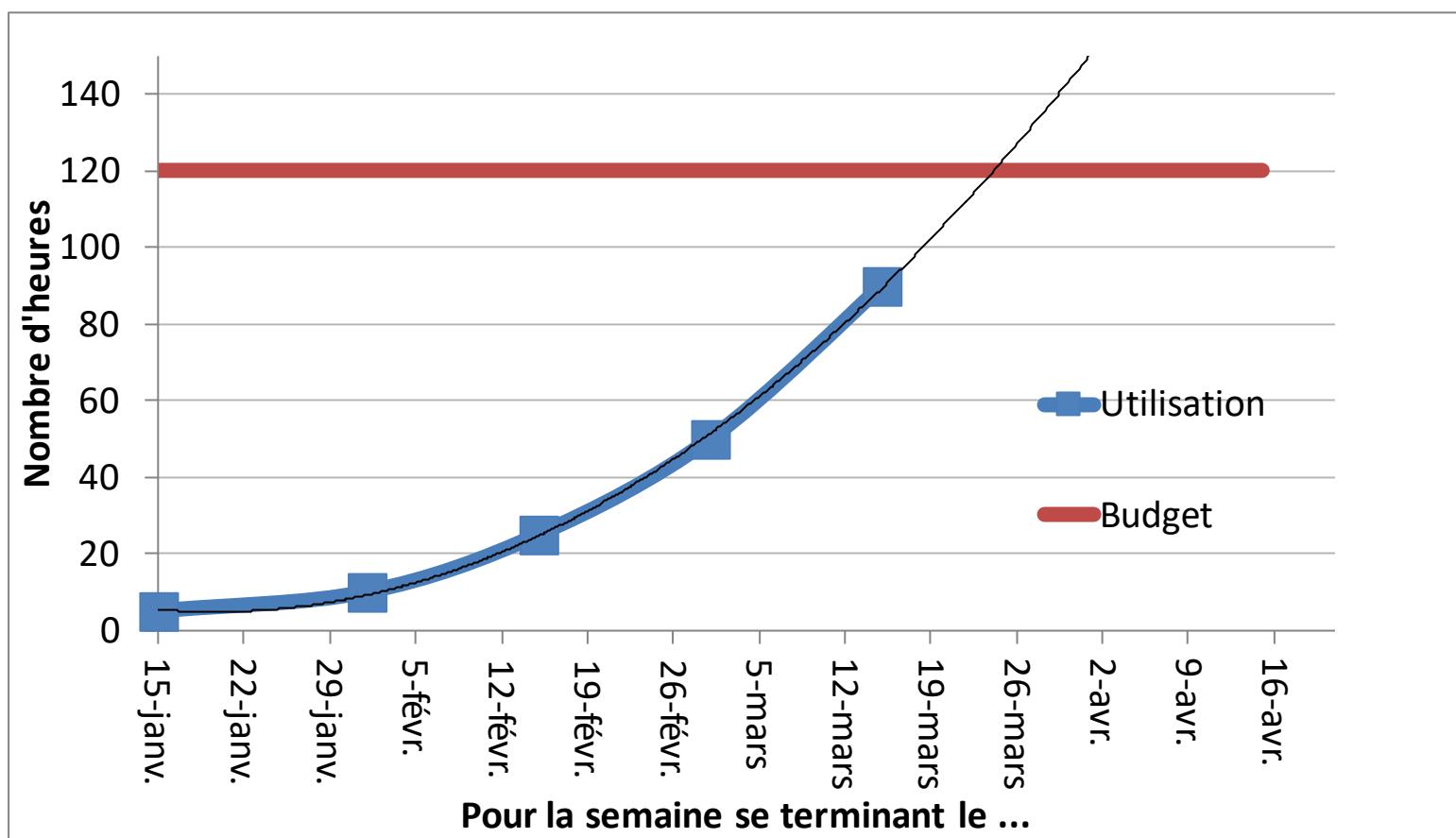
Analyse quantitative de processus

1. Introduction

2. Analyse qualitative de processus

3. Analyse quantitative de processus

- Données objectives non-sujettes à l'interprétation.
- Base des sciences pures : Mathématiques, physique, etc.
- Exemple: Utilisation des ressources dédiées aux tests dans un projet de développement logiciel.



Analyse quantitative de processus

1. Introduction
2. Analyse qualitative de processus
3. Analyse quantitative de processus

- L'analyse quantitative de processus se base sur des données numériques tirées à différents endroits du processus.
 - Ex.: Heures travaillées, lignes de code écrites, *user stories* implémentées, bogues résolus, etc.
- Nécessite d'avoir une bonne approche de mesure :
 - Savoir quoi et comment mesurer,
 - Stocker adéquatement les données, et les jeter lorsqu'elles sont obsolètes,
 - Être capable d'analyser les données obtenues.
 - Etc.

Exemple 1: Analyse quantitative



Cascade

- Les release sortent dans les dates arbitraires

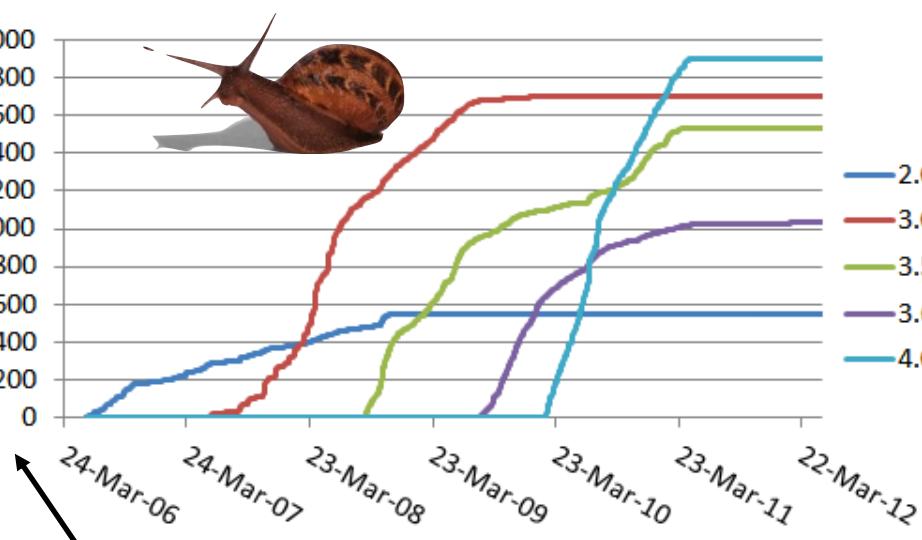
Incrémental

- Les releases sortent chaque 6 semaines

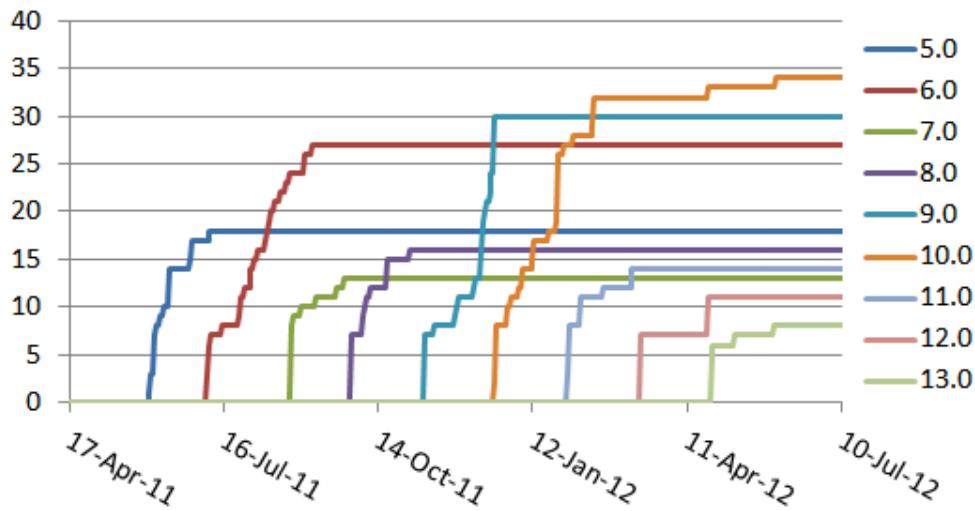
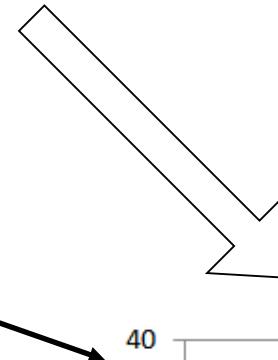
2011

Les tests s'est toujours basé sur des volontaires qui doivent télécharger la dernière version, puis rapporter des problèmes rencontrés en exécutant des scénarios d'exécution prescrites par des cas de test.

Quels sont les impacts de la nouveau pratique de releases sur les activités de tests?

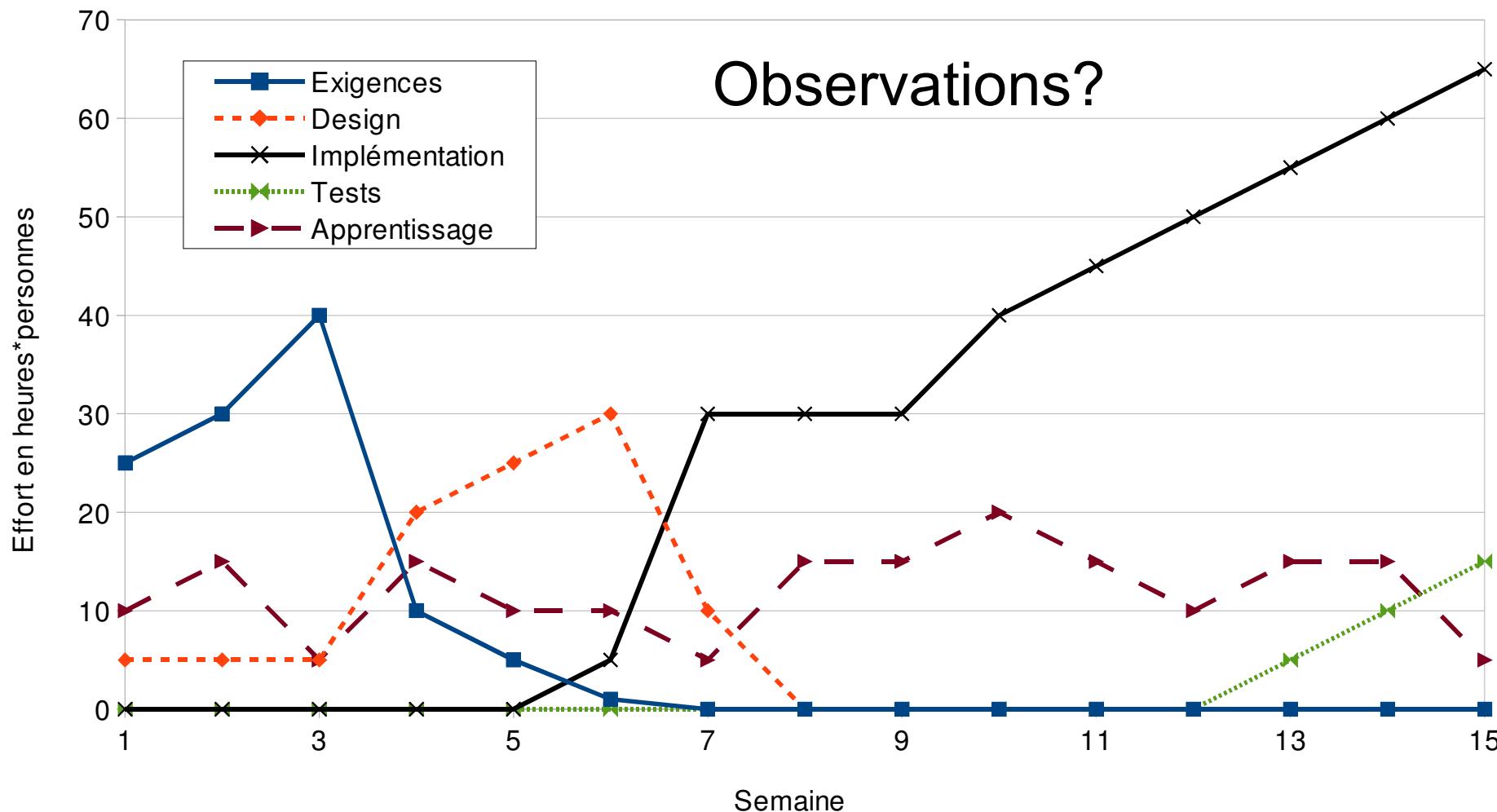


#testeurs manuels



Exemple 2: Analyse quantitative

Contexte: Vos collègues viennent de finir leur projet intégrateur de 4ième année, suivant un processus discipliné de type cascade. Malheureusement, le produit final est d'une qualité médiocre, et il n'implémente même pas toutes les exigences initialement prévues.



Approche Qualitative

Codification

Interprétation

Approche Quantitative

LOG3000

Processus du génie logiciel

1. Définitions
2. Échelles de mesure
3. Corrélation et causalité
4. Utilité de la mesure

Théorie de la mesure

Monde abstrait et monde empirique

1. Définitions

2. Échelles de mesure

3. Corrélation et causalité

4. Utilité de la mesure

MONDE ABSTRAIT



Théorie

« $g = 9,82 \text{ m/s}^2$ »

Proposition

« Accélération \propto
Gravité »

Hypothèse

« Tous les objets sont
attirés vers la Terre. »

Observation

« Une pomme tombe
vers le sol. »

MONDE EMPIRIQUE

Primitive de base
de mon système



Monde abstrait et monde empirique

1. Définitions
2. Échelles de mesure
3. Corrélation et causalité
4. Utilité de la mesure

MONDE ABSTRAIT



Concept

« *Fiabilité* »

Définition

« *Proportion de temps durant une année où le système était accessible.* »

Qualité opérationnelle

« $8h/8760h = 99,9\%$ »

Mesures réelles

« *Panne le 23 janvier entre 08h00 et 16h00.* »

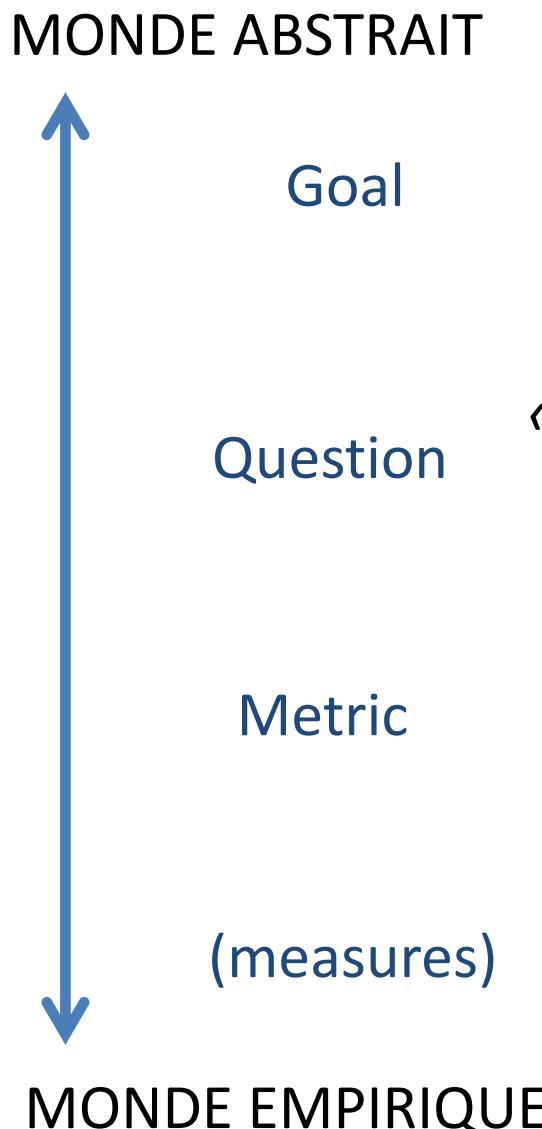
MONDE EMPIRIQUE

Primitive de base de mon système



Approche « GQM »

1. Définitions
2. Échelles de mesure
3. Corrélation et causalité
4. Utilité de la mesure



« L'élément abstrait que je veux évaluer »

« Les réponses à trouver permettant d'évaluer mon objectif. »

« Les calculs qui me permettent de répondre à mes questions »

« Les données réelles prises sur le terrain. »



Victor Basili

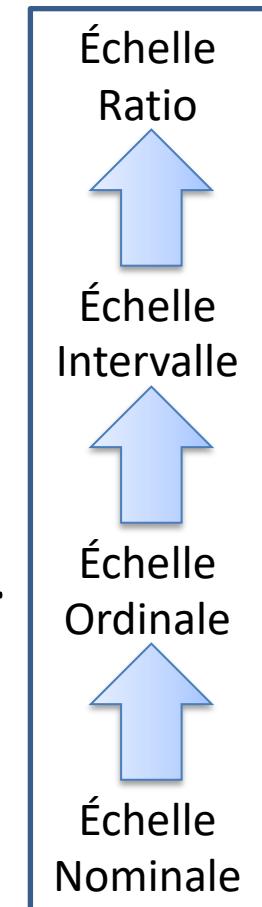
Exemple - Maintenabilité

Étape	Description
Goal	Examiner la maintenabilité du logiciel du projet 2e année.
Question(s)	Est-ce que le code est facile à comprendre? Est-ce que le code est bien structuré en modules?
Metric(s)	Métriques de cohésion et couplage. Ratio de commentaires. Complexité du code Temps requis pour corriger un bogue. Perception qualitative du débogueur.
[measure(s)]	Couplage moyen de 3,3. Complexité cyclomatique moyenne de 4,2. 15% de commentaires. 28 minutes pour corriger un bogue. Perception « bonne » par le débogueur.

Échelles de mesure

1. Définitions
2. Échelles de mesure
3. Corrélation et causalité
4. Utilité de la mesure

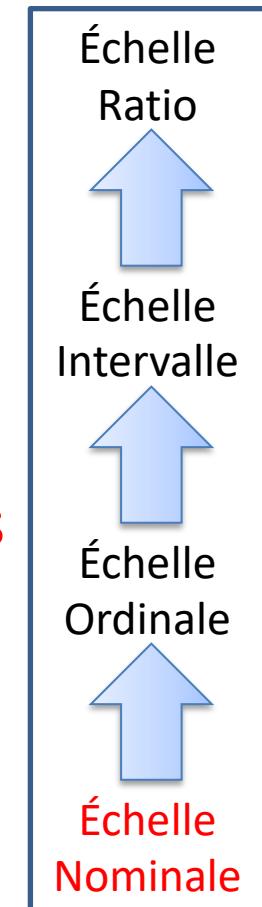
- Problème
 - De bonnes mesures demandent de bonnes métriques.
 - Une bonne métrique demande d'utiliser une bonne échelle de mesure.
- Échelles de mesure
 - Les échelles forment une **hiérarchie**.
 - Le niveau hiérarchique supérieur possède toutes les propriétés des niveaux inférieurs.
 - Niveau hiérarchique élevé → Grande **puissance d'analyse**.
 - À considérer lors de la conception des **définitions opérationnelles** (c.-à-d. les métriques).



Échelle nominale

1. Définitions
2. Échelles de mesure
3. Corrélation et causalité
4. Utilité de la mesure

- Reflète une classification, un groupement, une **étiquette**.
 - Ex.: Section 2, cours 3000.
- Les catégories doivent être :
 - Conjointement exhaustives : Couvrir toutes les possibilités,
 - Mutuellement exclusives : Un élément ne peut être dans deux catégories.
- **Aucune relation de comparaison entre les catégories.**



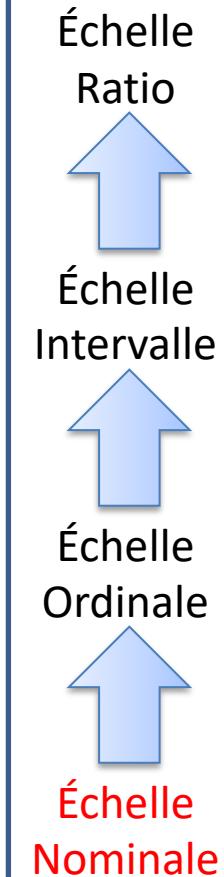
Échelle nominale (exemple)

- Le joueur portant le chandail 30 est différent du joueur portant le chandail 20. **VRAI**
- Le joueur portant le chandail 30 est meilleur que le joueur portant le chandail 20. **FAUX**
- J'ai le chandail 10. La différence entre moi et le joueur avec le chandail 20 est la même qu'avec le joueur avec le chandail 20 et le joueur avec le chandail 30. **FAUX**
- J'ai le chandail 10. Le joueur avec le chandail 30 est trois fois plus bon que moi. **FAUX**

Numéro de
chandail des
joueurs



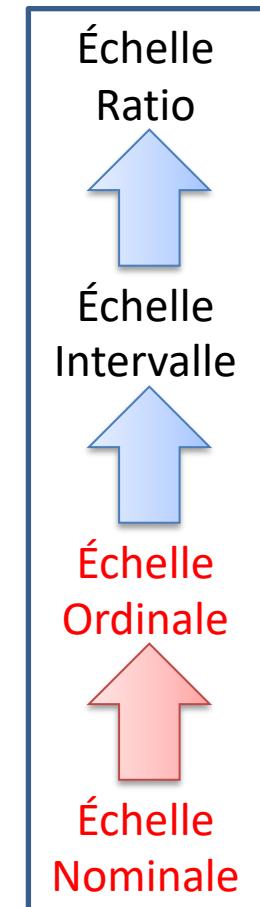
Source: CTV News



Échelle ordinaire

1. Définitions
2. Échelles de mesure
3. Corrélation et causalité
4. Utilité de la mesure

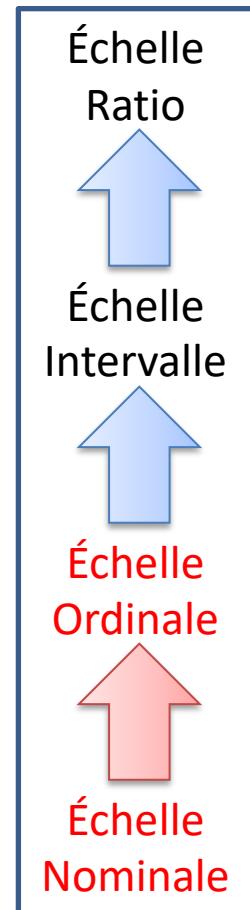
- Reflète un ordre de **comparaison**.
- Asymétrie
 $X > Y \rightarrow \neg(Y > X)$
- Transitivité
 $(X > Y) \wedge (Y > Z) \rightarrow (X > Z)$
- **Aucune information sur l'envergure des différences entre les éléments.**
 - On ne peut donc pas utiliser les opérations mathématiques élémentaires (+, -, ×, ÷).



Échelle ordinaire (exemple)

- La maison portant l'adresse 20 est différente de la maison portant l'adresse 30. **VRAI**
- Je suis à l'adresse 10 et l'adresse 20 est à ma droite. Alors l'adresse 5 est aussi à ma gauche. **VRAI**
- La distance entre l'adresse 10 et l'adresse 20 est la même que la distance entre l'adresse 20 et l'adresse 30. **FAUX**
- L'adresse 30 est trois fois plus loin que l'adresse 10. **FAUX**

Adresses civiques



Échelle intervalle

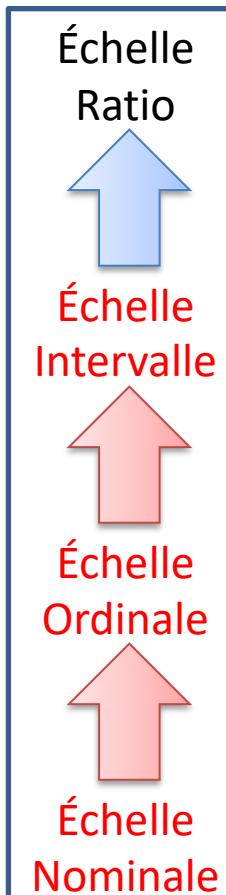
1. Définitions

2. Échelles de mesure

3. Corrélation et causalité

4. Utilité de la mesure

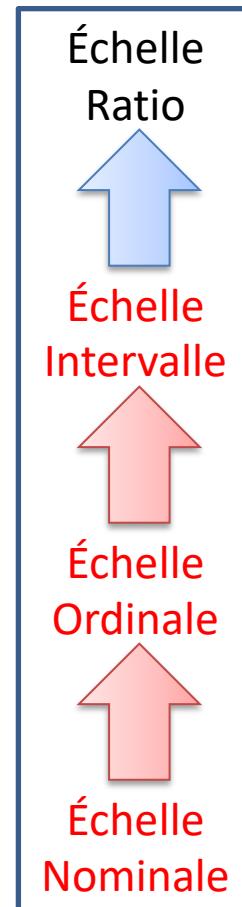
- Indique avec exactitude les **écart**s entre les éléments.
- Mesures entières ou fractionnaires.
- **Absence de zéro absolu** non-arbitraire.
- Certaines opération élémentaires permises (+, -).
- Type d'échelle rare en pratique.



Échelle intervalle (exemple)

- La température 20°C est différente de la température 30°C. **VRAI**
- La température 30°C est plus élevée que la température 20°C. **VRAI**
- La différence entre 10°C et 20°C est la même que la différence entre 20°C et 30°C. **VRAI**
- Hier il faisait -10°C et aujourd’hui il fait -20°C. Il fait donc deux fois plus froid qu’hier.
FAUX

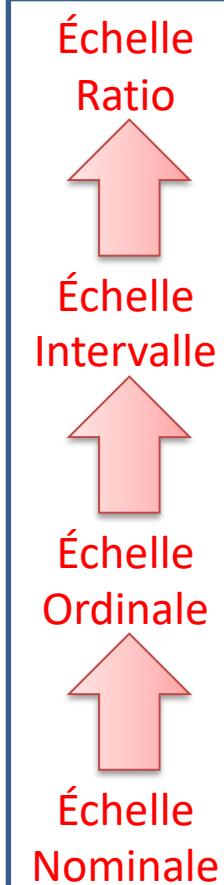
Thermomètres
(Celsius ou
Farhenheit)



Échelle ratio

1. Définitions
2. Échelles de mesure
3. Corrélation et causalité
4. Utilité de la mesure

- Présence d'un **zéro absolu** non-arbitraire → La valeur zéro veut dire l'absence de ce qui est mesuré.
- Capacité de calculer des proportions (\times , \div)



Échelle ratio (exemple)

- 20km/h est différent de 30km/h. **VRAI**
- 30km/h est plus rapide que la 20km/h. **VRAI**
- La différence entre 10km/h et 20km/h est la même que la différence entre 20km/h et 30km/h. **VRAI**
- 20km/h est deux fois plus rapide que 10km/h. **VRAI**

Indicateur
de vitesse



Corrélation et causalité

1. Définitions

2. Échelles de mesure

3. Corrélation et causalité

4. Utilité de la mesure

- Idéalement, il y a une relation causale entre nos mesures et le phénomène réel.
 - Ex.: Bogues \uparrow = Effort \uparrow
 - Ex.: Compréhensibilité du code \downarrow = Coût de la maintenance \uparrow
- Mais ce n'est pas toujours le cas !

Exemple populaire :

 - Au 18^e siècle, il y avait beaucoup de pirates et peu de gaz à effet de serre.
 - La Somalie a beaucoup de pirates, et produit peu de gaz à effet de serre.
 - On peut donc conclure que plus il y a de pirates, moins il y a de gaz à effet de serre.



Corrélation et causalité

1. Définitions

2. Échelles de mesure

3. Corrélation et causalité

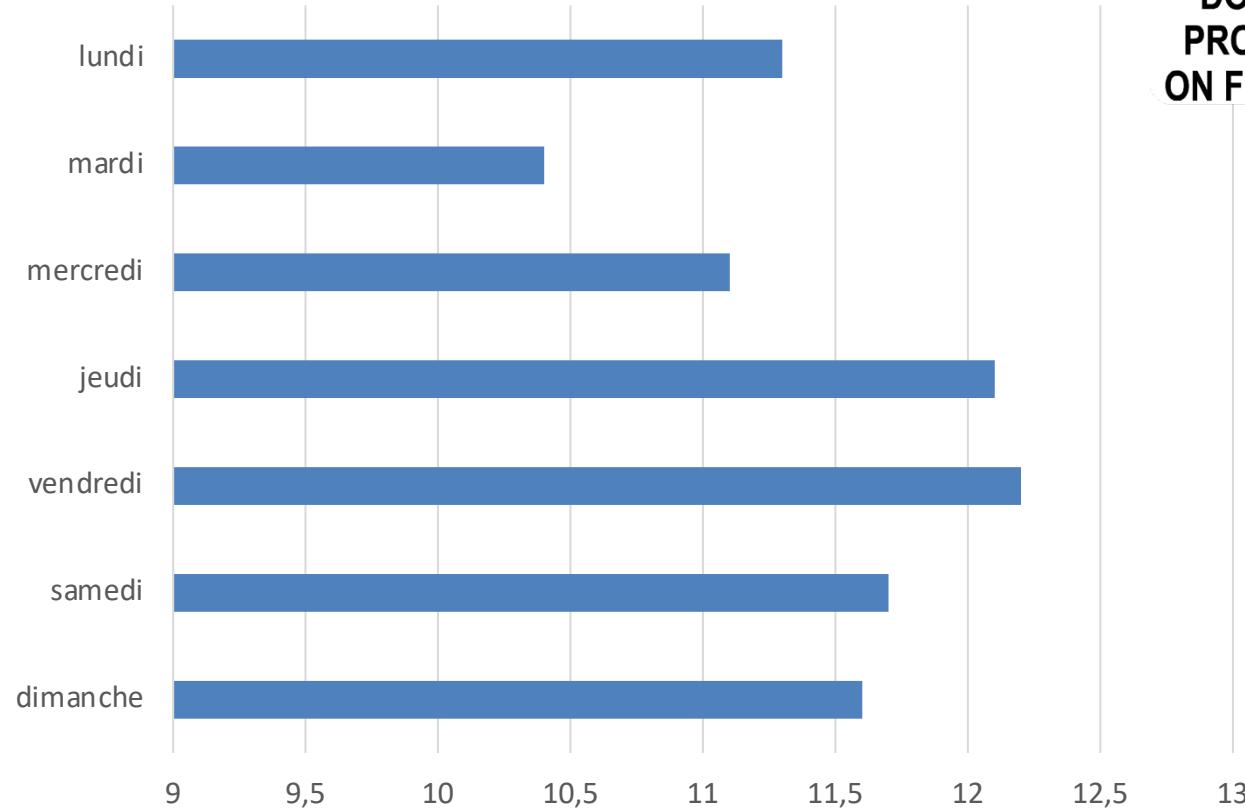
4. Utilité de la mesure

- **Corrélation n'équivaut donc pas à une relation de cause à effet**
- Pour qu'il y ait causalité :
 - Il faut que la cause précède l'effet :
 - Dans le temps,
 - Dans le déroulement logique des opérations.
 - La corrélation doit être validée sur le terrain.
 - Et pas seulement aperçue sur papier.
 - La corrélation ne doit pas être le résultat d'une relation parasite, comme par exemple :
 - La présence d'une cause commune,
 - La présence d'une variable intermédiaire.
- Une corrélation observée peut aussi être due à une simple coïncidence.
- Conclusion → **Toujours valider nos corrélations sur le terrain.**

Ne programmez pas les vendredis ;-)



1. Définitions
2. Échelles de mesure
3. Corrélation et causalité
4. Utilité de la mesure



Pourcentage de commits introduisant un bogue dans Eclipse

Utilité de la mesure

1. Définitions

2. Échelles de mesure

3. Corrélation et causalité

4. Utilité de la mesure

- Pourquoi les mesures sont importantes ?
 - Pour comprendre :
 - Apprendre sur le processus de développement, le produit, les services;
 - Pour évaluer :
 - Effectuer des analyses de coûts et bénéfices;
 - Déterminer si nos objectifs ont été atteints;
 - Pour **contrôler** :
 - Faire le **suivi du projet**, des ressources, des processus, des produits, des services;
 - Pour **prédir** :
 - **Planifier les calendriers**, les ressources, les risques;
 - Pour améliorer :
 - Identifier les causes des défauts, les inefficacités du processus.

Pensées à retenir

"You cannot control what you cannot measure."

Anthony A. DeMarco

Expert en analyse et estimation de projets logiciels

"There is no sense being precise about something when you don't even know what you are talking about."

John von Neumann (1903-1957)

Mathématicien, physicien, statisticien, économiste, ...

"Nobody was permitted to see the Emperor of China, and the question was, what is the length of the Emperor of China's nose ? To find out, you go all over the country asking people what they think the length of the Emperor of China's nose is, and you average it. And that would be very 'accurate' because you average so many people."



Richard Feynman (1918-1988)
Physicien