



L3 MIAGE apprentissage
Université Paris-Sud
Année 2017-2018

UML

Introduction au génie logiciel et à la modélisation

Delphine Longuet
delphine.longuet@lri.fr

Organisation du cours

Partie 1 : UML (Delphine Longuet)

Partie 2 : Test (Fatiha Zaïdi)

Modalités de contrôle des connaissances :

- Partie 1 : projet en groupes
- Partie 2 : TP noté + examen

Note finale = 30 % projet + 10 % TP + 60 % examen

Page web de la première partie du cours :

<http://www.lri.fr/~longuet/Enseignements/17-18/L3-App>

Organisation de la partie 1

Cours :

- Pas de cours magistral (ou presque)
- Vidéos sur Youtube mises en ligne au fur et à mesure

Avant chaque séance : visionner la ou les vidéos requises

Pendant chaque séance :

- Pratiquer et confronter les modélisations
- Travailler sur le projet en groupes

Projet :

- Documentation sur Google Drive
- Modélisation UML sur GenMyModel
- Évaluations à dates fixes

Génie logiciel ? Logiciel ?

Répondre aux questions 1 à 4 du questionnaire en binôme.

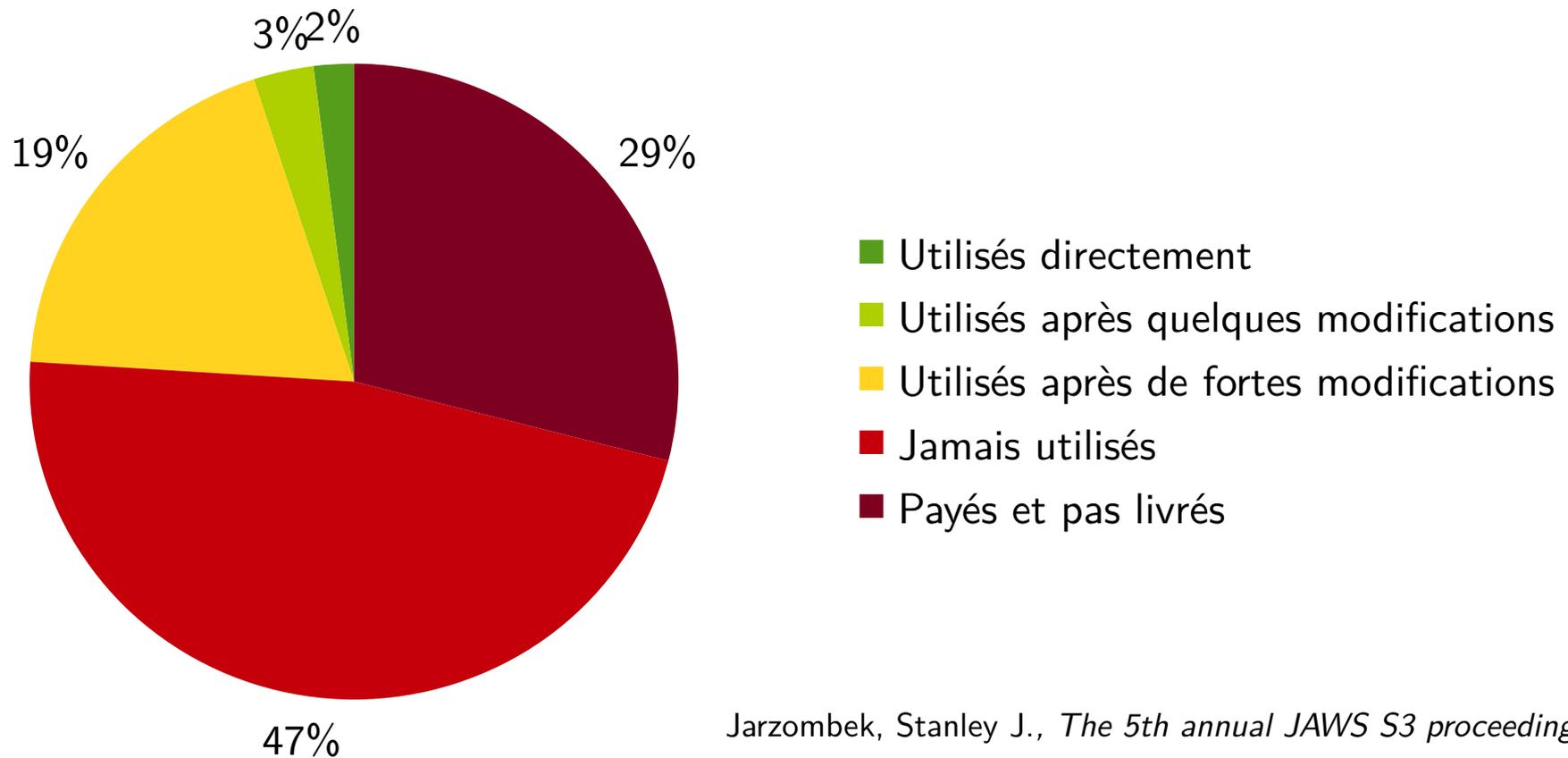
Crise du logiciel

Constat du développement logiciel fin années 60 :

- délais de livraison **non respectés**
- budgets **non respectés**
- **ne répond pas aux besoins** de l'utilisateur ou du client
- **difficile** à utiliser, maintenir, et faire évoluer

Étude du DoD 1995

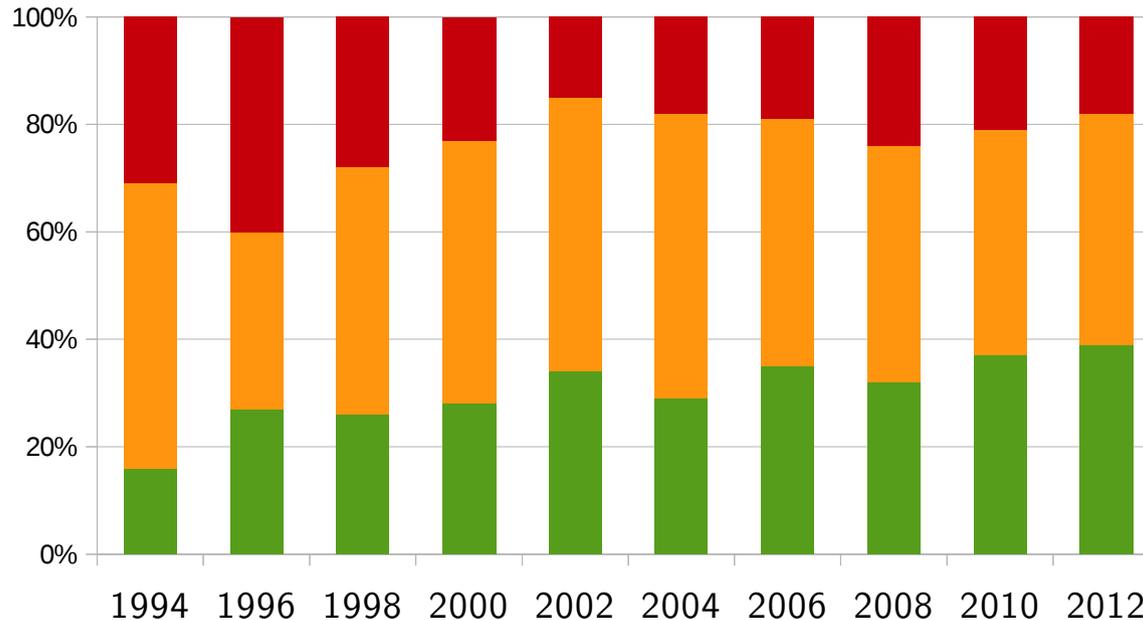
Étude du *Department of Defense* des États-Unis sur les logiciels produits dans le cadre de 9 gros projets militaires



Jarzombek, Stanley J., *The 5th annual JAWS S3 proceedings*, 1999

Étude du Standish group

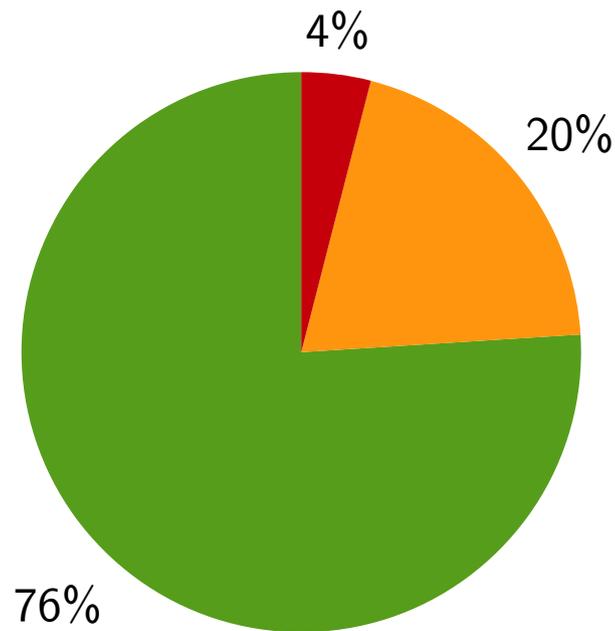
Enquête sur des milliers de projets, de toutes tailles et de tous secteurs



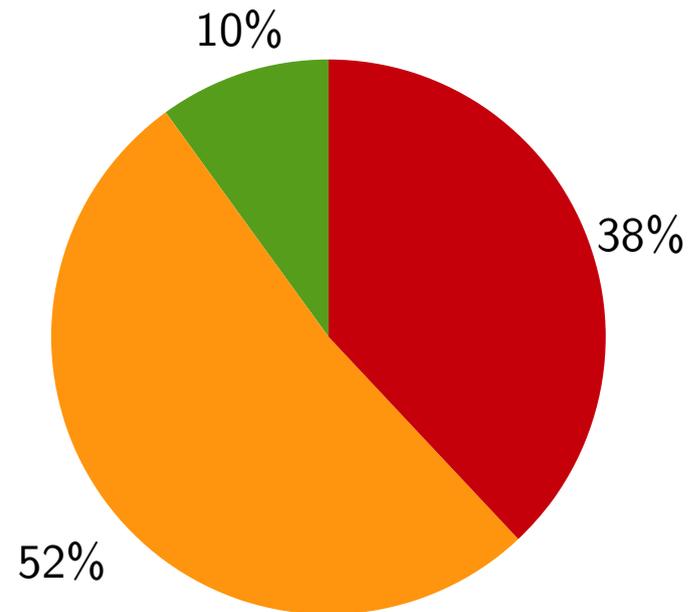
Standish group, *Chaos Manifesto 2013 - Think Big, Act Small*, 2013

- **Projets réussis** : achevés dans les délais et pour le budget impartis, avec toutes les fonctionnalités demandées
- **Projets mitigés** : achevés et opérationnels, mais livrés hors délais, hors budget ou sans toutes les fonctionnalités demandées
- **Projets ratés** : abandonnés avant la fin ou livrés mais jamais utilisés

Petits vs grands projets



Petits projets
budget \leq \$1 million

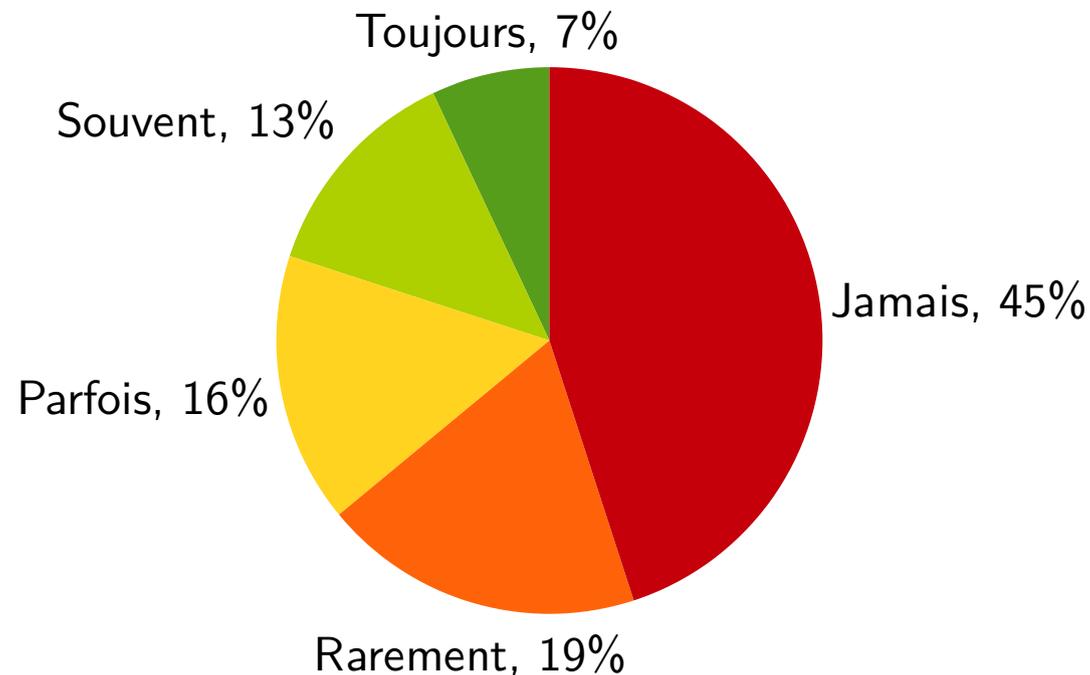


Grands projets
budget \geq \$10 millions

- Projets réussis
- Projets mitigés
- Projets ratés

Standish group, *Chaos Manifesto 2013 - Think Big, Act Small*, 2013

Utilisation des fonctionnalités implantées



Standish group, *Chaos Manifesto 2002*, 2002

« La **satisfaction** du client et la **valeur** du produit sont plus grandes lorsque les **fonctionnalités** livrées sont bien **moins nombreuses** que demandé et ne remplissent que les **besoins évidents**. »

Standish group, *Chaos Report 2015*, 2015

Raisons de la faible qualité des logiciels

Tâche complexe :

- Taille et complexité des logiciels
- Taille des équipes de conception/développement

Manque de méthodes et de rigueur :

- Manque de méthodes de conception
- Négligence et manque de méthodes et d'outils des phases de validation/vérification

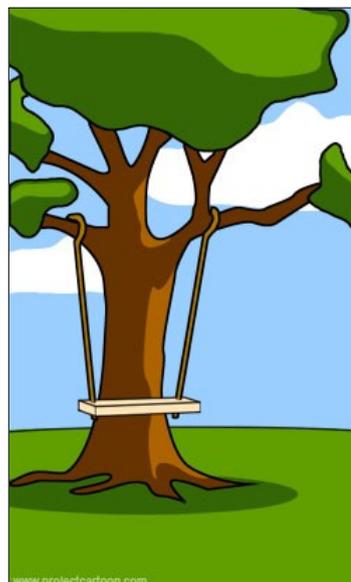
Mauvaise compréhension des besoins :

- Négligence de la phase d'analyse des besoins du client
- Manque d'implication du client dans le processus

Raisons de la faible qualité des logiciels



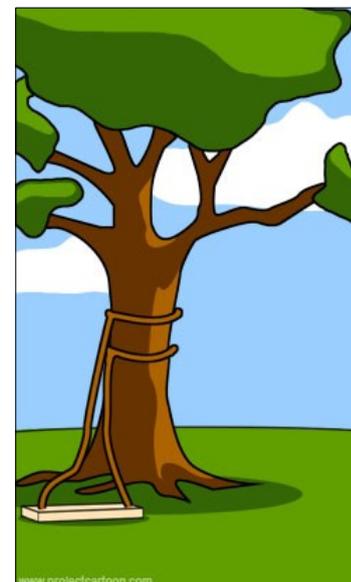
Ce que le client a expliqué



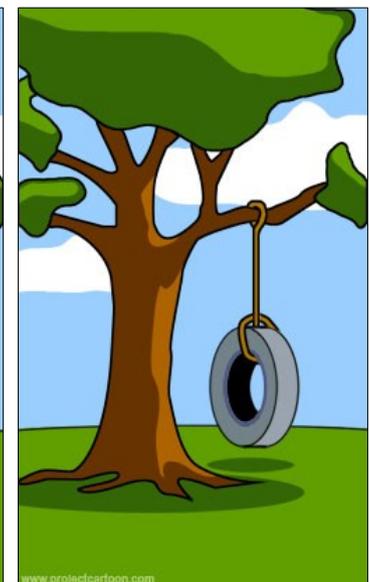
Ce que le chef de projet a compris



Ce que l'analyste a proposé



Ce que le programmeur a écrit



Ce dont le client avait vraiment besoin

Raisons de la faible qualité des logiciels

Difficultés spécifiques du logiciel :

- Produit invisible et immatériel
- Difficile de mesurer la qualité
- Conséquences critiques causées par modifications infimes
- Mises à jour et maintenance dues à l'évolution rapide de la technologie
- Difficile de raisonner sur des programmes
- Défaillances logicielles principalement humaines

Importance de la qualité des logiciels

Fiabilité, sûreté et sécurité des logiciels

- Transports automobile, ferroviaire, aéronautique
- Contrôle de processus industriels, nucléaire, armement
- Médical : imagerie, appareillage, télé-surveillance
- e-commerce, carte bancaire sans contact, passeport électronique

Raisons économiques : coût d'un bug

- Coût de la correction, du rappel des appareils défectueux
- Coût de l'impact sur l'image, de l'arrivée tardive sur le marché
- Coût en vies, coût de l'impact écologique

Génie logiciel

Idée : appliquer les **méthodes classiques d'ingénierie** au domaine du logiciel

Ingénierie (ou **génie**) : Ensemble des fonctions allant de la **conception** et des **études** à la responsabilité de la **construction** et au **contrôle des équipements** d'une installation technique ou industrielle

Génie civil, naval, aéronautique, mécanique, chimique...

Qualité du logiciel

Critères de qualité

- **Validité** : réponse aux besoins des utilisateurs
- **Facilité d'utilisation** : prise en main et robustesse
- **Performance** : temps de réponse, débit, fluidité...
- **Fiabilité** : tolérance aux pannes
- **Sécurité** : intégrité des données et protection des accès
- **Maintenabilité** : facilité à corriger ou transformer le logiciel
- **Portabilité** : changement d'environnement matériel ou logiciel

Processus de développement logiciel

Ensemble d'**activités** successives, **organisées** en vue de la production d'un logiciel

En pratique :

- Pas de processus idéal
- Choix du processus en fonction des contraintes (taille des équipes, temps, qualité...)
- **Adaptation de « processus types » aux besoins réels**

Processus de développement logiciel

Activités du développement logiciel

- Analyse des besoins
- Spécification
- Conception
- Programmation
- Validation et vérification
- Livraison
- Maintenance

Pour chaque activité : Utilisation et production de documents

Activités du développement logiciel

Analyse des besoins : Comprendre les **besoins du client**

- Objectifs généraux, environnement du futur système, ressources disponibles, contraintes de performance...
- Fournie par le **client** (expert du domaine d'application, futur utilisateur...)

Spécification :

- Établir une description claire de **ce que doit faire** le logiciel (fonctionnalités détaillées, exigences de qualité, interface...)
- Clarifier le cahier des charges (ambiguïtés, contradictions) en listant les **exigences fonctionnelles et non fonctionnelles**

Activités du développement logiciel

Conception : Élaborer une **solution concrète** réalisant la spécification

- Description **architecturale** en composants (avec interface et fonctionnalités)
- **Réalisation des fonctionnalités** par les composants (algorithmes, organisation des données)
- Réalisation des exigences non fonctionnelles (performance, sécurité...)

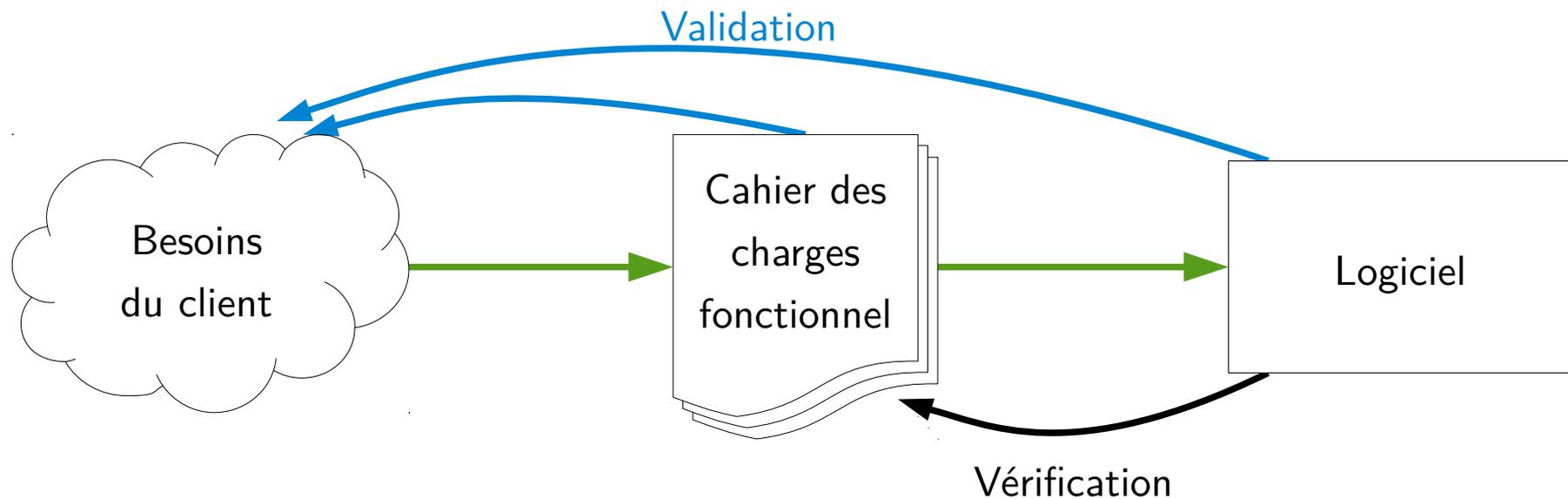
Programmation : **Implantation** de la solution conçue

- Choix de l'environnement de développement, du/des langage(s) de programmation, de normes de développement...

Validation et vérification

Objectifs :

- **Validation** : assurer que les **besoins du client** sont satisfaits (au niveau de la spécification, du produit fini...)
- **Vérification** : assurer que le logiciel satisfait sa **spécification**

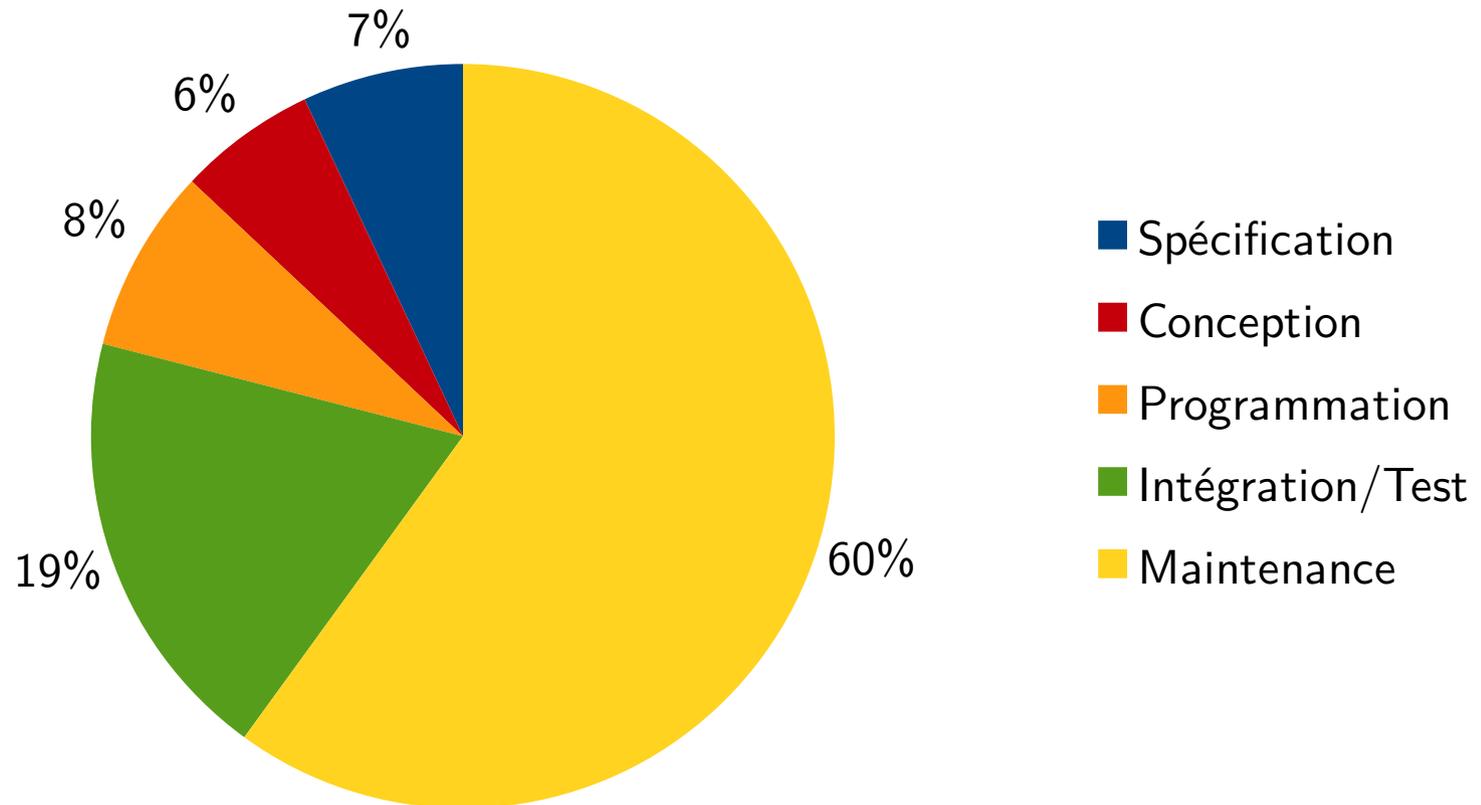


Maintenance

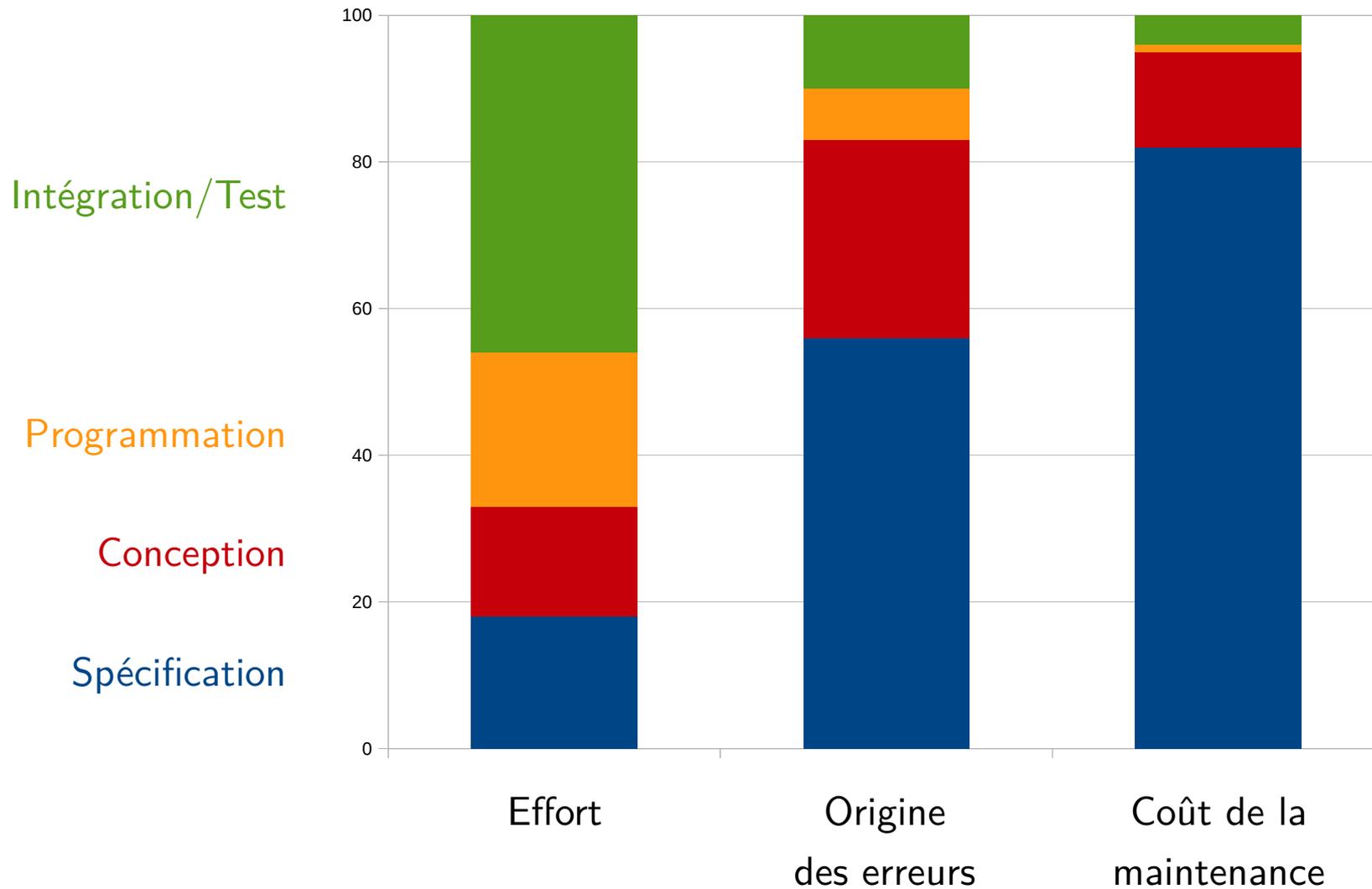
Types de maintenance :

- **Correction** : identifier et corriger des erreurs trouvées après la livraison
- **Adaptation** : adapter le logiciel aux changements dans l'environnement (format des données, environnement d'exécution...)
- **Perfection** : améliorer la performance, ajouter des fonctionnalités, améliorer la maintenabilité du logiciel

Répartition de l'effort

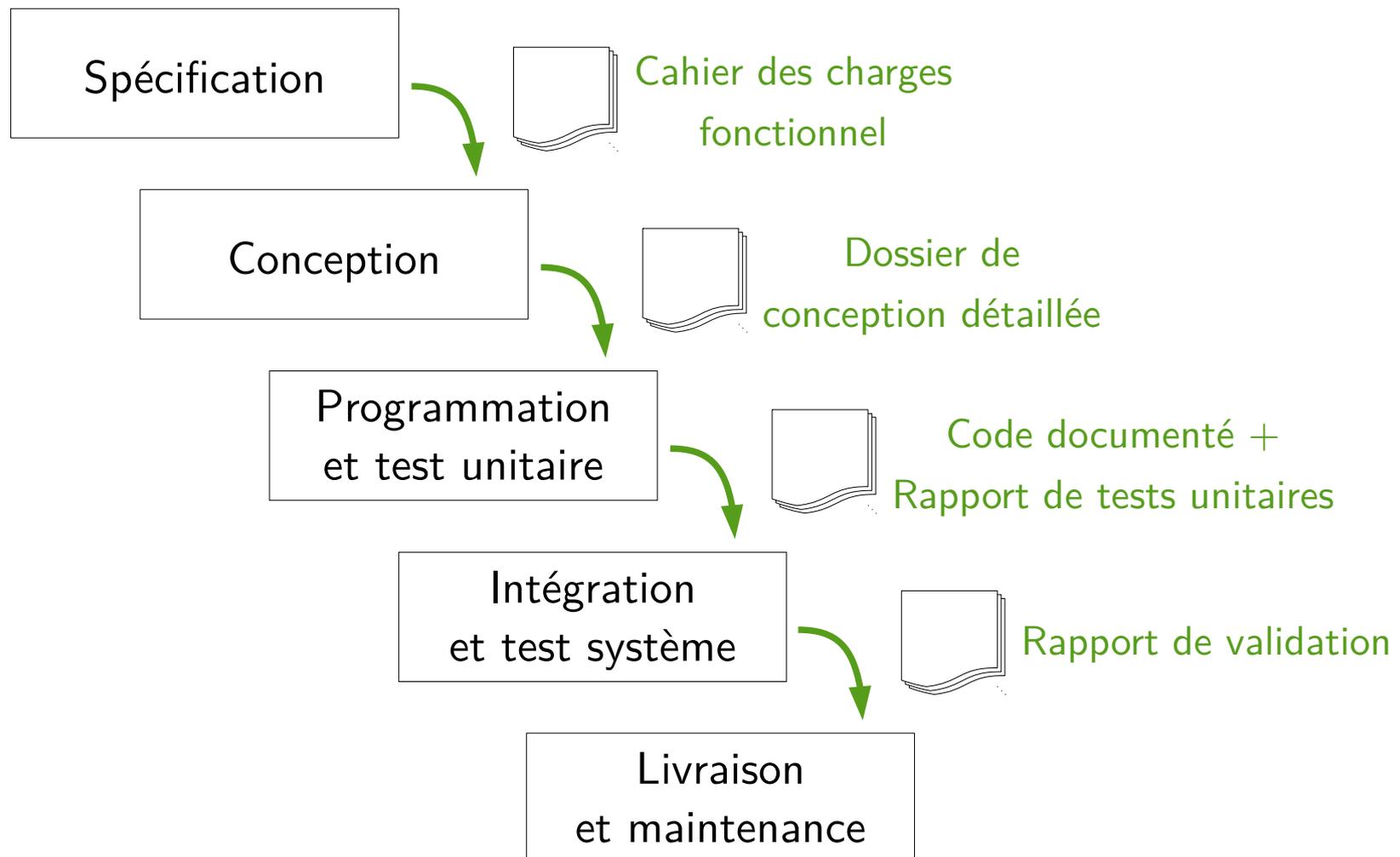


Rapport effort/erreur/coût



Processus en cascade

Chaque étape doit être terminée avant que ne commence la suivante
À chaque étape, production d'un document base de l'étape suivante



Processus en cascade

Caractéristiques :

- Hérité des **méthodes classiques** d'ingénierie
- Découverte d'une erreur entraîne retour à la phase à l'origine de l'erreur et nouvelle cascade, avec de nouveaux documents...
- Coût de modification d'une erreur important, donc **choix en amont cruciaux** (typique d'une production industrielle)

Pas toujours adapté à une production logicielle, en particulier si besoins du client changeants ou difficiles à spécifier

Autres processus de développement logiciel

1. Processus en V
2. Processus itératif ou incrémental
3. Processus agile

15 minutes de préparation par groupes de 5-6
5 minutes de présentation par un membre du groupe

Documentation

Objectif : Traçabilité du projet

Pour l'équipe :

- Regrouper et structurer les décisions prises
- Faire référence pour les décisions futures
- Garantir la cohérence entre les modèles et le produit

Pour le client :

- Donner une vision claire de l'état d'avancement du projet

Base commune de référence :

- Personne quittant le projet : pas de perte d'informations
- Personne rejoignant le projet : intégration rapide

Documents de spécification et conception

Rédaction : le plus souvent en langage naturel (français)

Problèmes :

- **Ambiguïtés** : plusieurs sens d'un même mot selon les personnes ou les contextes
- **Contradictions, oublis, redondances** difficiles à détecter
- Difficultés à **trouver une information**
- Mélange entre les **niveaux d'abstraction** (spécification vs. conception)

Documents de spécification et conception

Alternatives au langage naturel

Langages informels :

- **Langage naturel structuré** : modèles de document et règles de rédaction précis et documentés
- **Pseudo-code** : description algorithmique de l'exécution d'une tâche, donnant une vision opérationnelle du système

Langages semi-formels :

- **Notation graphique** : diagrammes accompagnés de texte structuré, donnant une vue statique ou dynamique du système

Langages formels :

- **Formalisme mathématique** : propriétés logiques ou modèle du comportement du système dans un langage mathématique

Documents de spécification et conception

Langages informels ou semi-formels :

- ✓ **Avantages** : intuitifs, fondés sur l'expérience, facile à apprendre et à utiliser, répandus
- ✗ **Inconvénients** : ambigus, pas d'analyse systématique

Langages formels :

- ✓ **Avantages** : précis, analysables automatiquement, utilisables pour automatiser la vérification et le test du logiciel
- ✗ **Inconvénients** : apprentissage et maîtrise difficiles

En pratique : utilisation de **langages formels** principalement pour logiciels critiques, ou restreinte aux **parties critiques** du système

Modélisation

Modèle : Simplification de la réalité, abstraction, vue subjective

- modèle météorologique, économique, démographique...

Modéliser un concept ou un objet pour :

- Mieux le comprendre (modélisation en physique)
- Mieux le construire (modélisation en ingénierie)

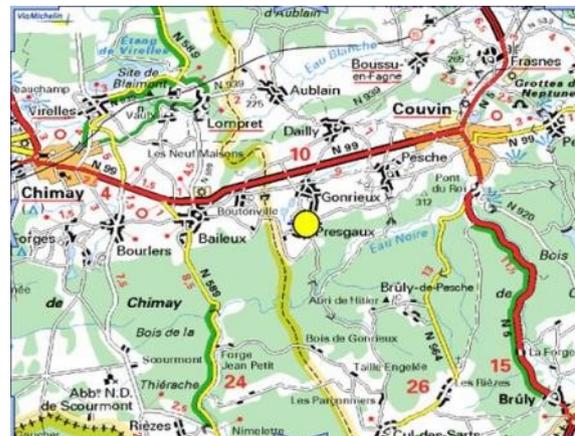
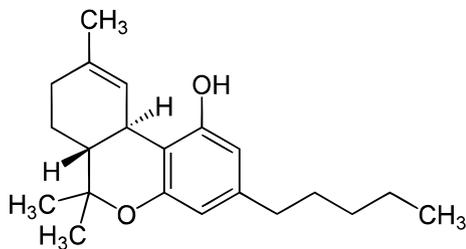
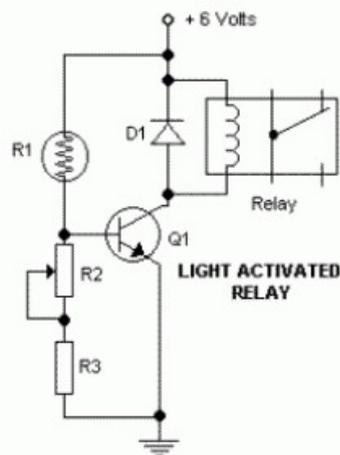
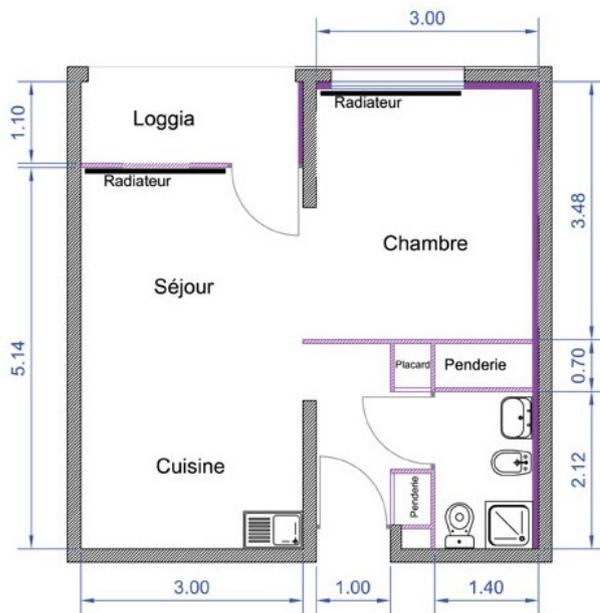
En génie logiciel :

- Modélisation = spécification + conception
- Aider la réalisation d'un logiciel à partir des besoins du client

Modélisation graphique

Principe : « Un beau dessin vaut mieux qu'un long discours »

Seulement s'il est compris par tous de la même manière



UML : Unified Modeling Language



Répondre aux questions 6 et 7 du questionnaire en binôme.

UML : Unified Modeling Language



Langage :

- **Syntaxe** et règles d'écriture
- Notations graphiques **normalisées**

... **de modélisation** :

- **Abstraction** du fonctionnement et de la structure du système
- **Spécification et conception**

... **unifié** :

- Fusion de plusieurs notations antérieures : Booch, OMT, OOSE
- **Standard** défini par l'OMG (Object Management Group)
- Dernière version : UML 2.5 (mai 2015)

En résumé : Langage graphique pour visualiser, spécifier, construire et documenter un logiciel

Pourquoi UML ?

Besoin de modéliser pour construire un logiciel

- Modélisation des aspects **statiques** et **dynamiques**
- Modélisation à différents **niveaux d'abstraction** et selon **plusieurs vues**
- Indépendant du processus de développement

Besoin de langages normalisés pour la modélisation

- Langage **semi-formel**
- **Standard** très utilisé

Conception orientée objet

- Façon efficace de **penser le logiciel**
- Indépendance du langage de programmation (langages non objet)

Conception orientée objet avec UML

UML

- **Langage graphique** : Ensemble de **diagrammes** permettant de modéliser le logiciel à selon différentes vues et à différents niveaux d'abstraction
- **Modélisation orientée objet** : modélisation du système comme un **ensemble d'objets** interagissant

UML **n'est pas une méthode** de conception

UML **est un outil indépendant** de la méthode

Diagrammes UML

14 diagrammes hiérarchiquement dépendants

Modélisation à tous les niveaux le long du processus de développement

Diagrammes structurels :

- Diagramme de classes
- Diagramme d'objets
- Diagramme de composants
- Diagramme de déploiement
- Diagramme de paquetages
- Diagramme de structure composite
- Diagramme de profils

Diagrammes comportementaux :

- Diagramme de cas d'utilisation
- Diagramme états-transitions
- Diagramme d'activité

Diagrammes d'interaction :

- Diagramme de séquence
- Diagramme de communication
- Diagramme global d'interaction
- Diagramme de temps

Exemple d'utilisation des diagrammes

Spécification

- Diagrammes de **cas d'utilisation** : **besoins** des utilisateurs
- Diagrammes de **séquence** : **scénarios** d'interactions entre les utilisateurs et le logiciel, vu de l'extérieur
- Diagrammes d'**activité** : enchaînement d'actions représentant un **comportement** du logiciel

Conception

- Diagrammes de **classes** : **structure interne** du logiciel
- Diagrammes d'**objet** : **état interne** du logiciel à un instant donné
- Diagrammes **états-transitions** : évolution de l'état d'un objet
- Diagrammes de **séquence** : scénarios d'interactions avec les utilisateurs ou **au sein du logiciel**
- Diagrammes de **composants** : composants **physiques** du logiciel
- Diagrammes de **déploiement** : organisation **matérielle** du logiciel

Prochaine séance : cas d'utilisation

Avant la séance : visionner les vidéos suivantes

- Cas d'utilisation – 1. Diagrammes de cas d'utilisation

<http://youtu.be/GC5BdRve38A>

- Cas d'utilisation – 2. Scénarios détaillés et diagrammes de séquence

<http://youtu.be/1G0omjzh1OQ>