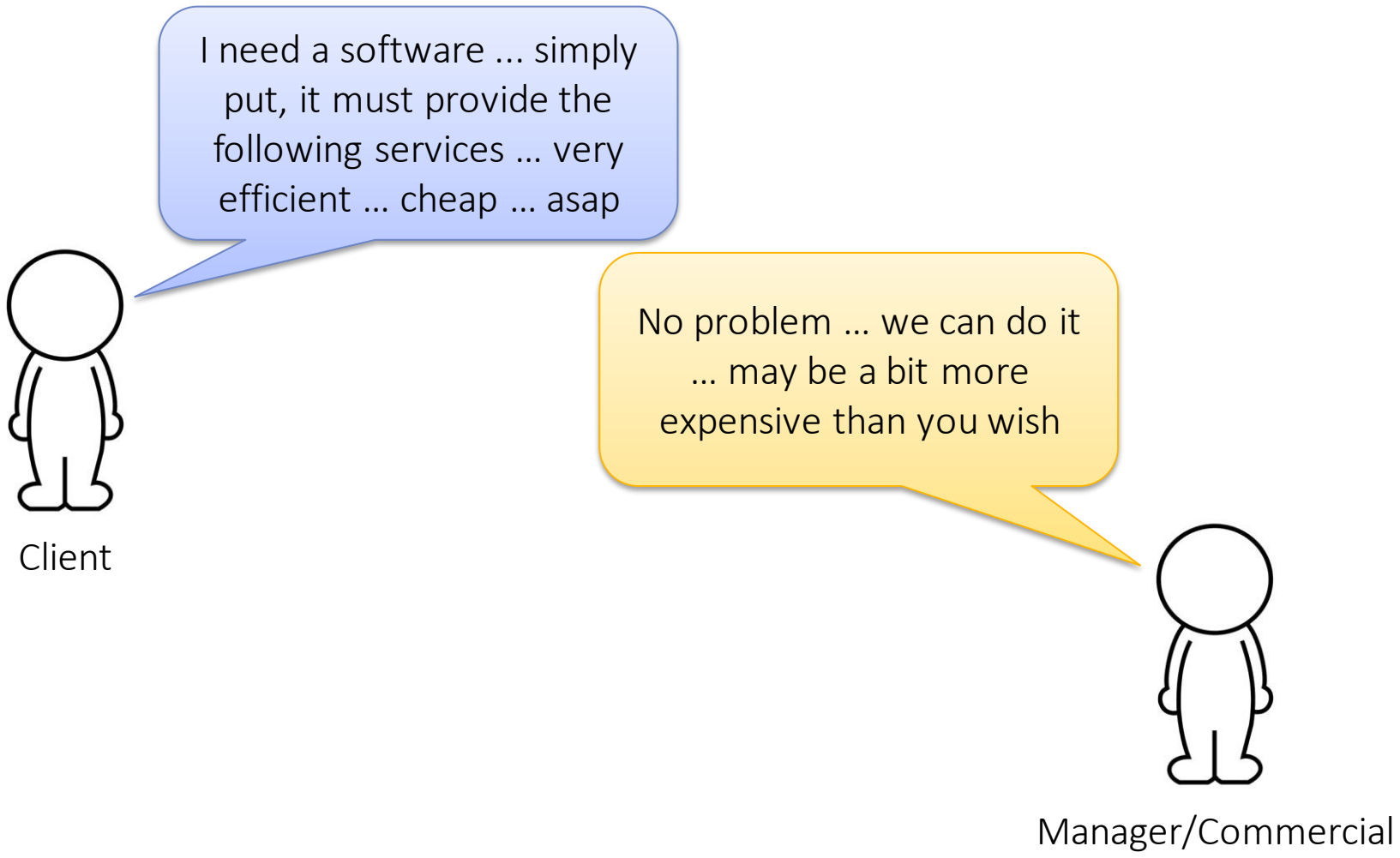
GRENOBLE INP UGA Esisar VALENCE

Introduction au Génie Logiciel

Stéphanie CHOLLET

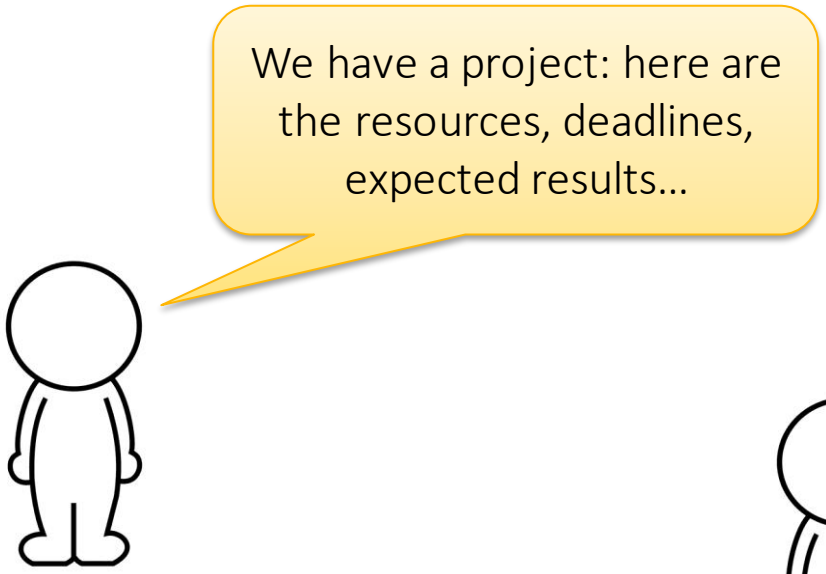
En partie d'après le cours de Pr. Philippe LALANDA

Une histoire qui se répète…

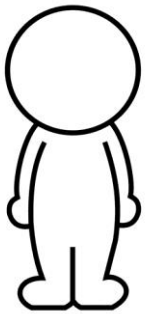


stephanie.chollet@grenoble-inp.fr

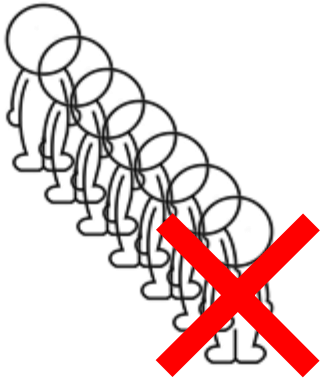
Une histoire qui se répète…



Manager/Commercial



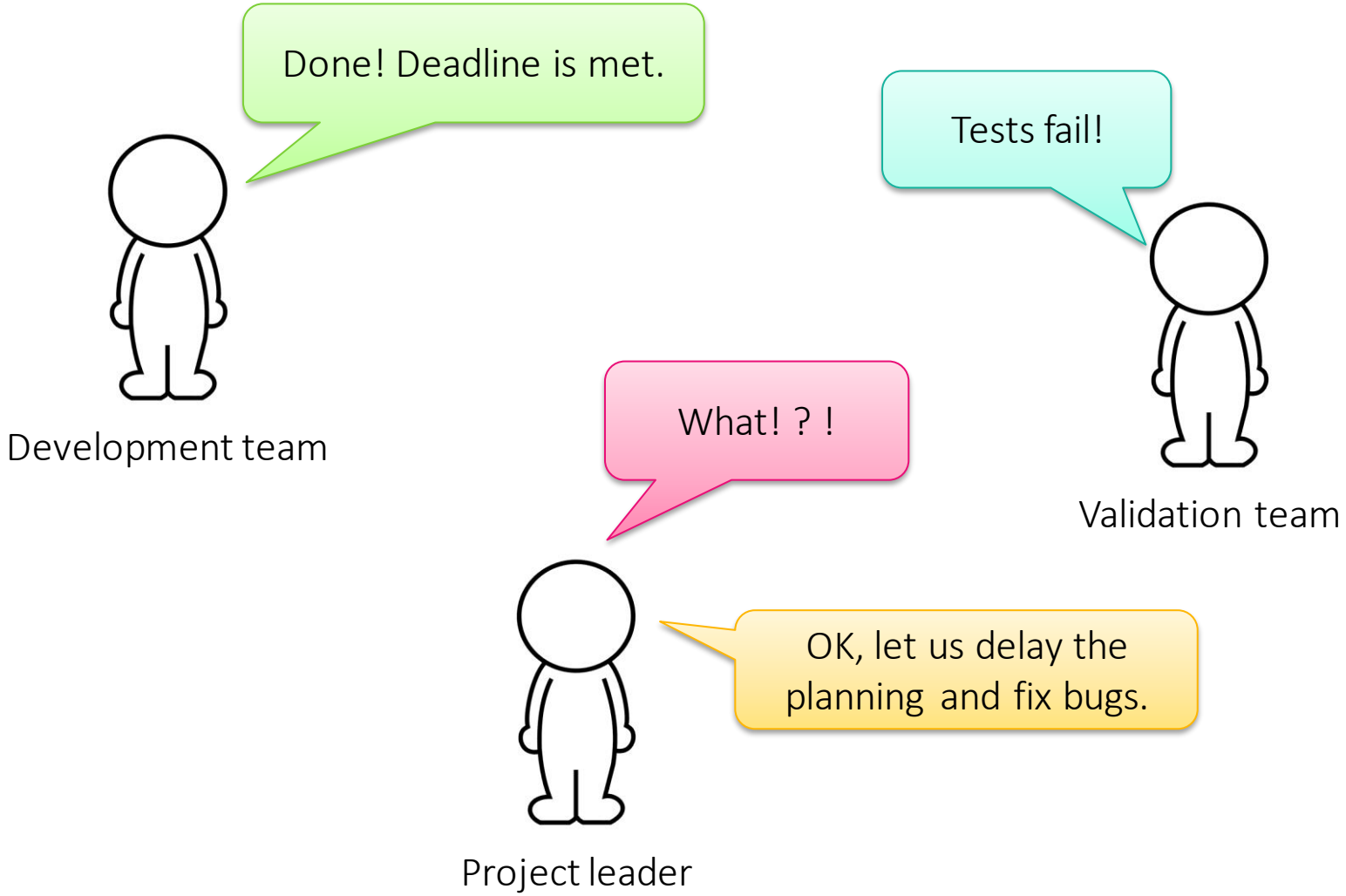
Project leader and team



No control!

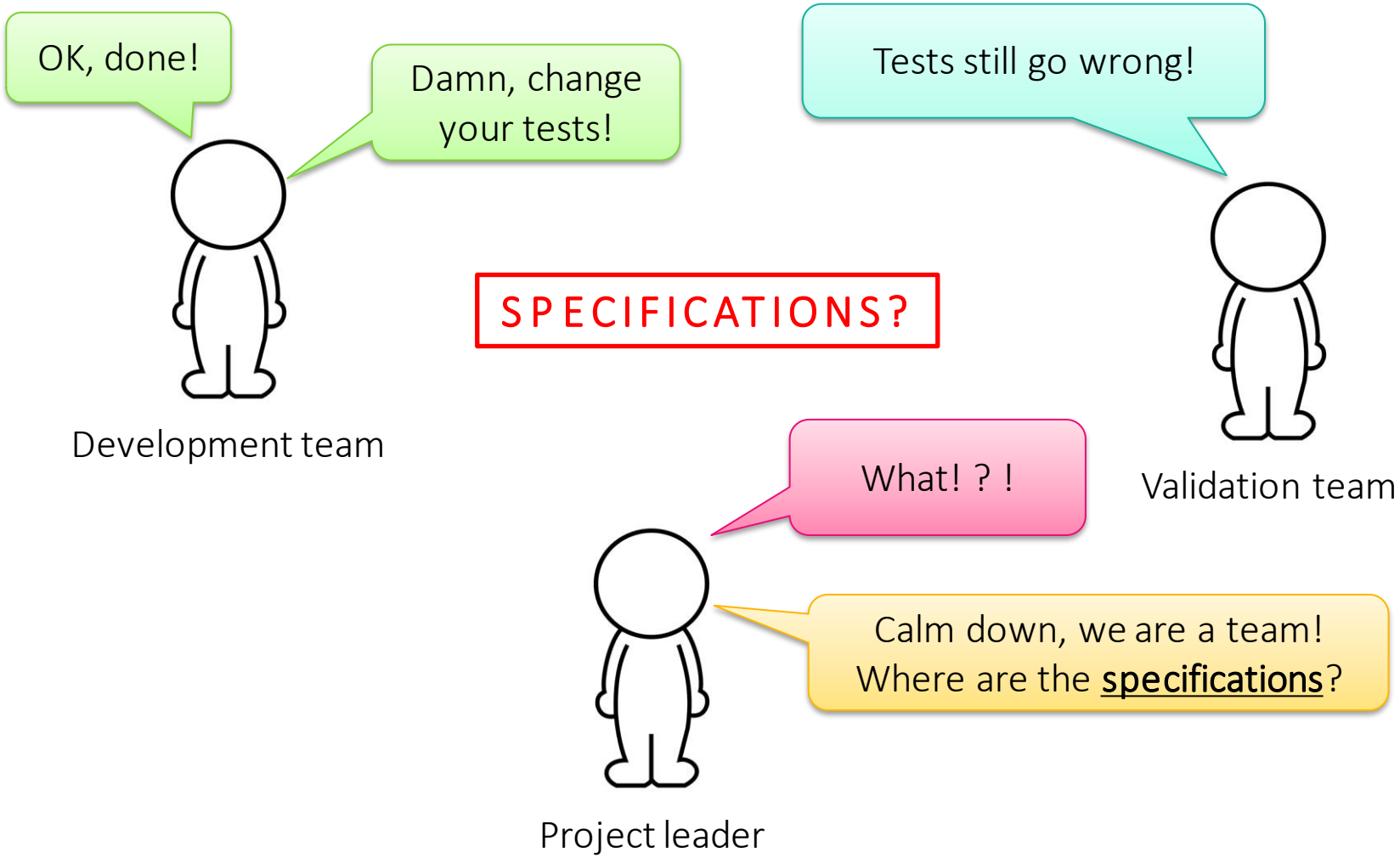
stephanie.chollet@grenoble-inp.fr

Une histoire qui se répète…



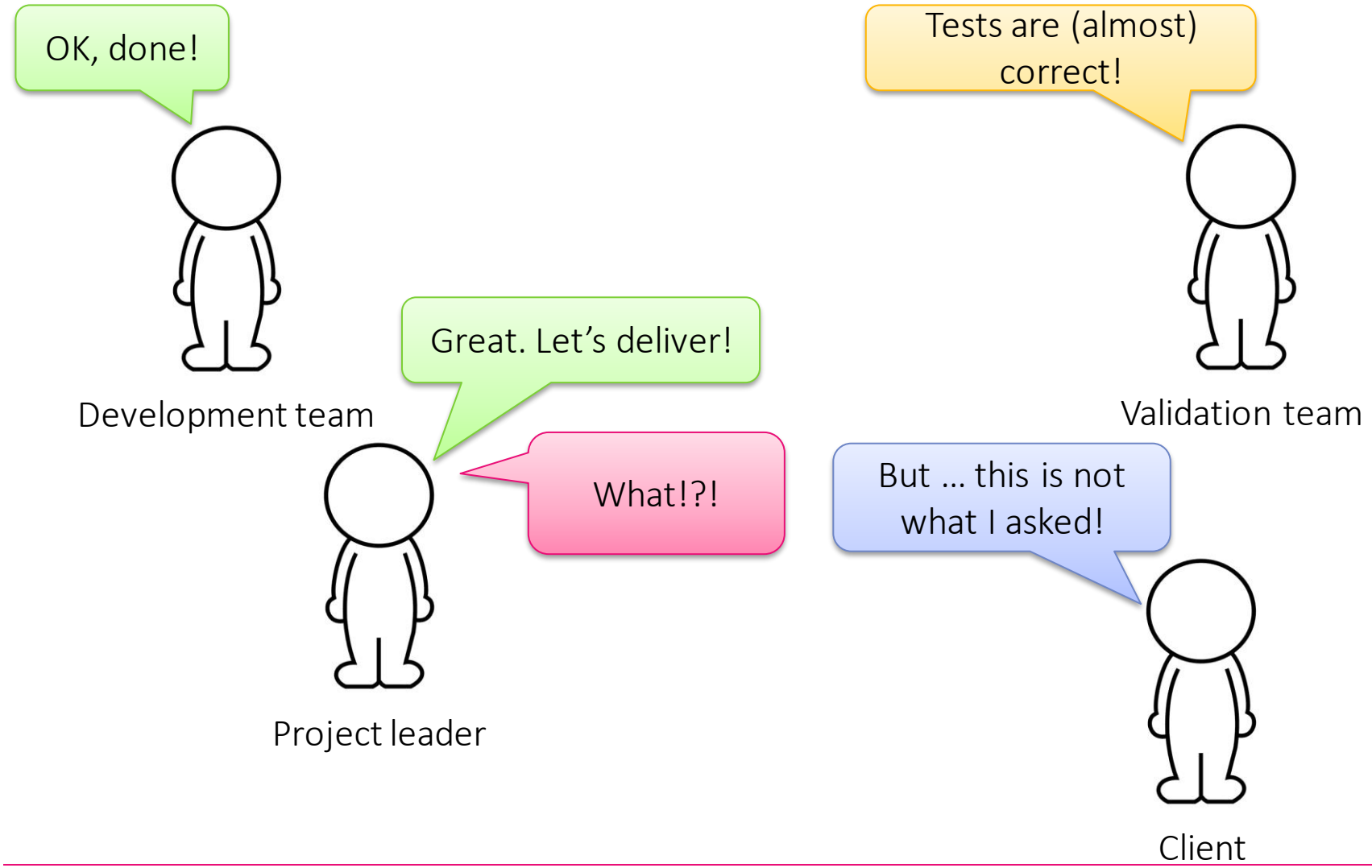
stephanie.chollet@grenoble-inp.fr

Une histoire qui se répète…



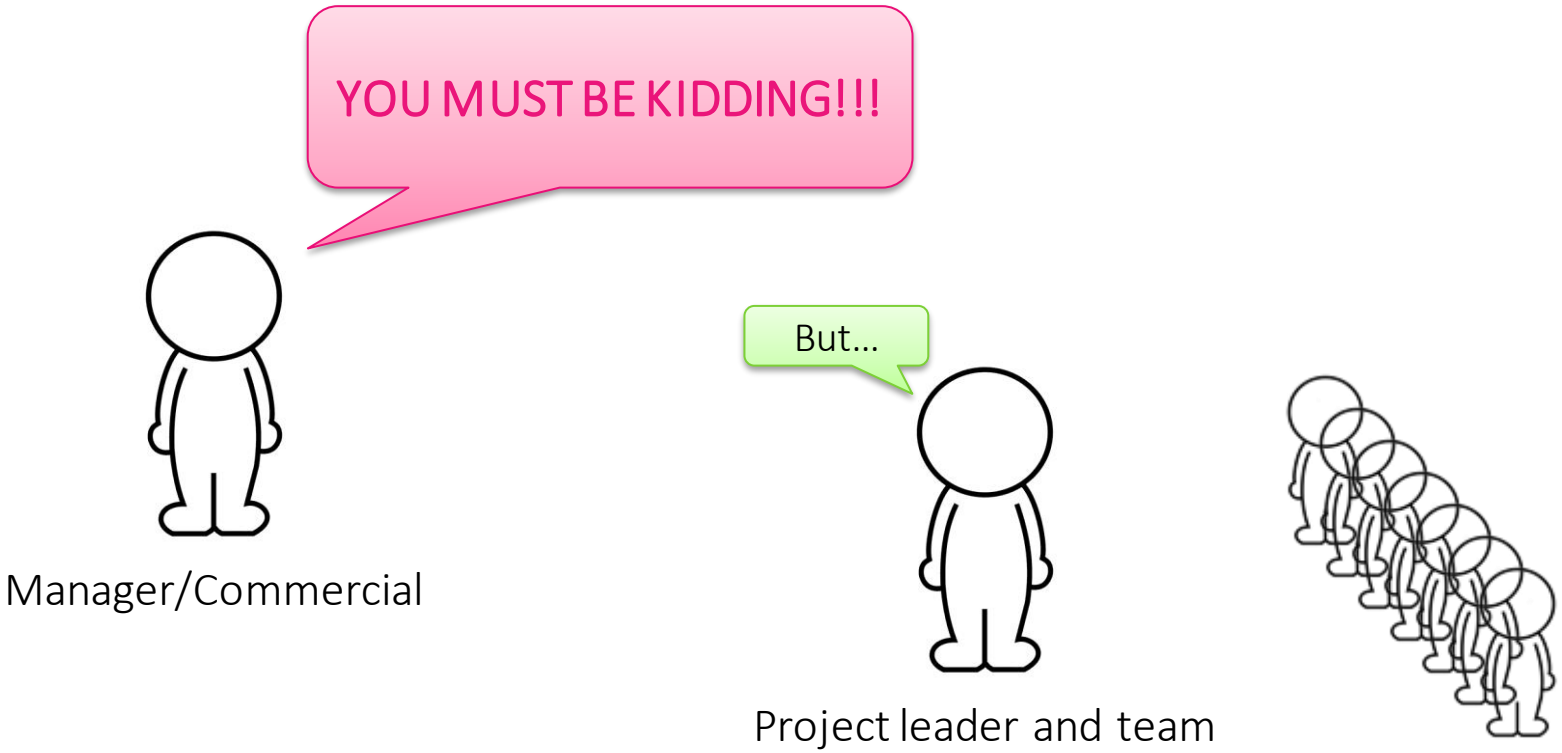
stephanie.chollet@grenoble-inp.fr

Une histoire qui se répète…



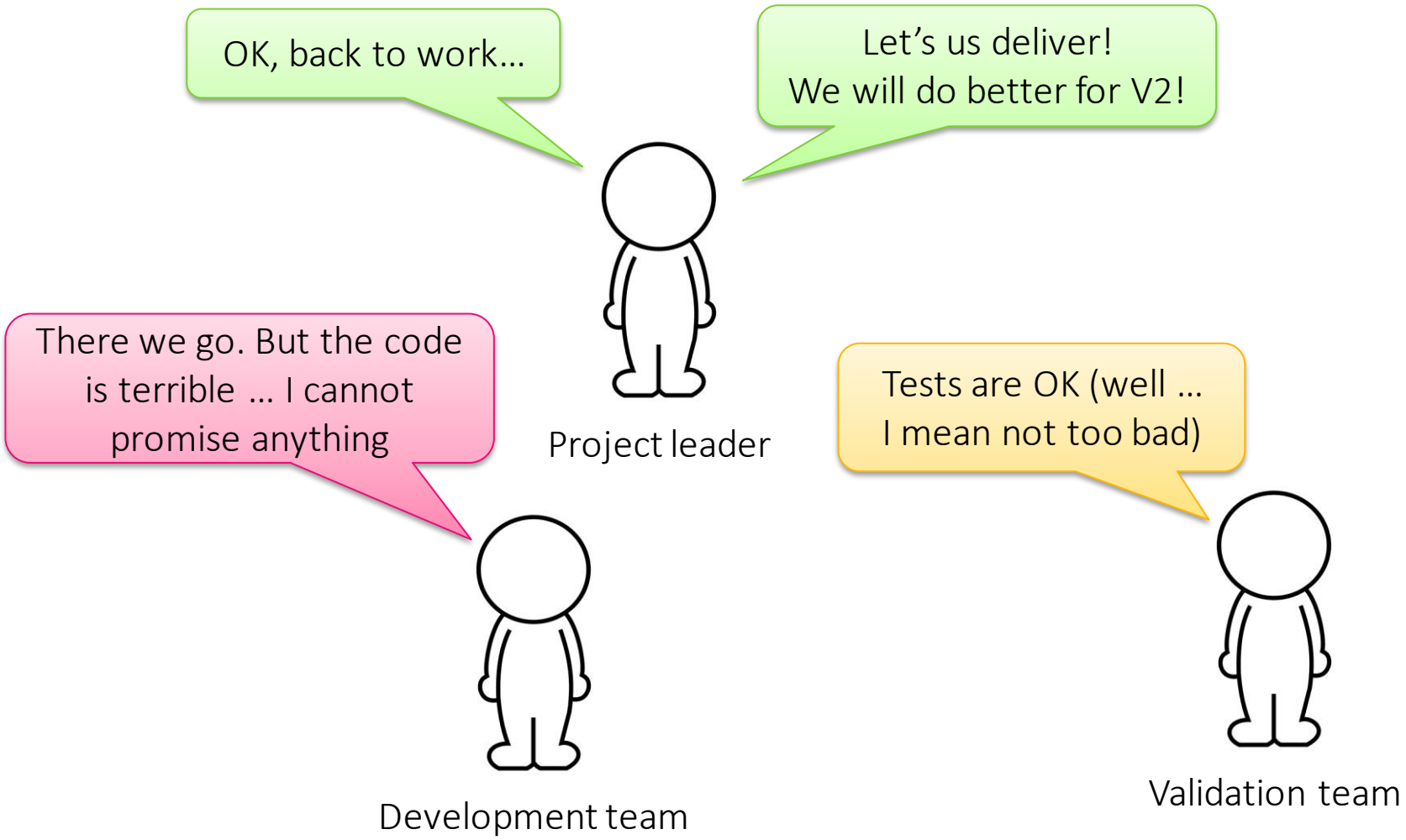
stephanie.chollet@grenoble-inp.fr

Une histoire qui se répète…



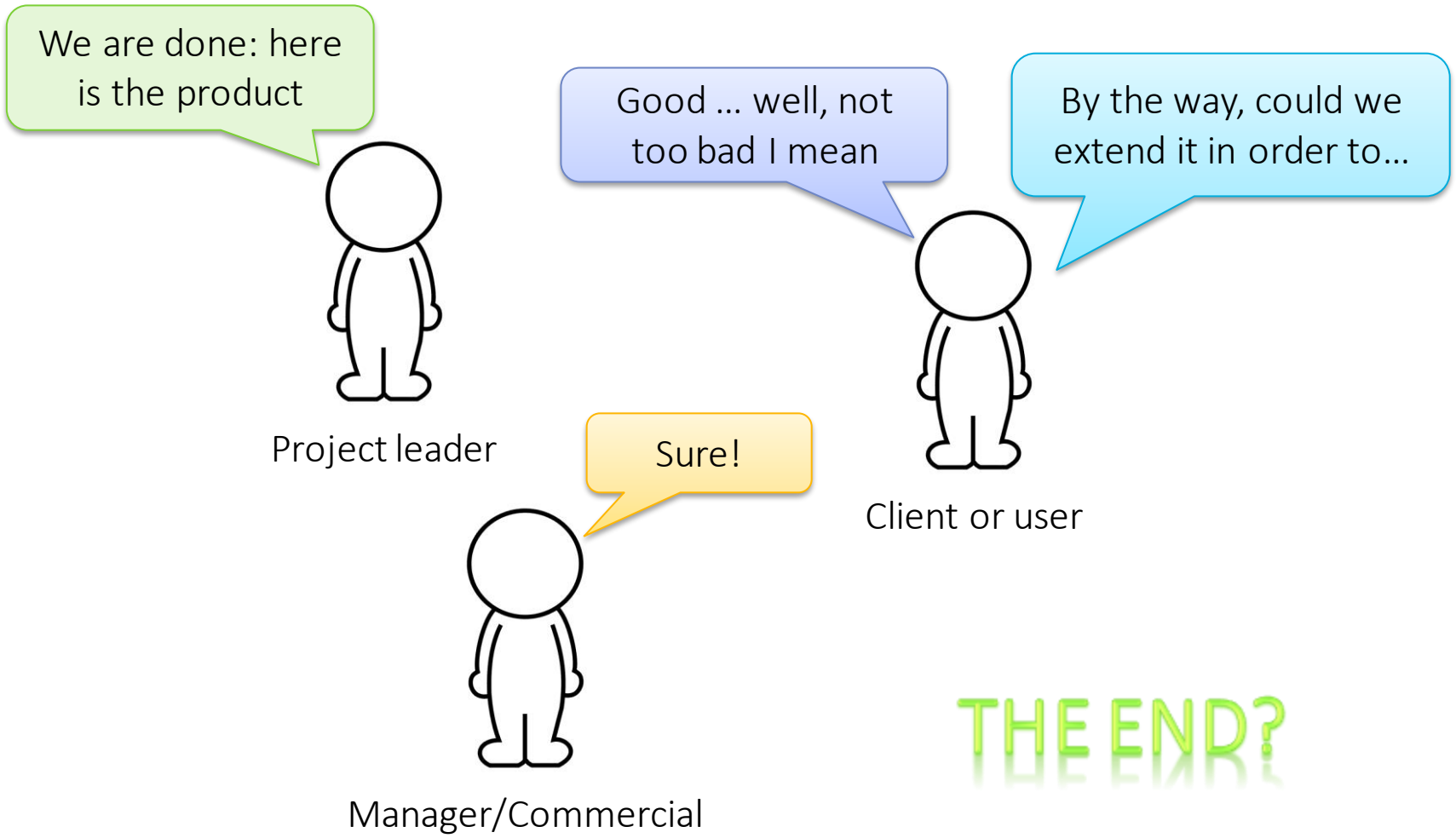
stephanie.chollet@grenoble-inp.fr

Une histoire qui se répète…



stephanie.chollet@grenoble-inp.fr

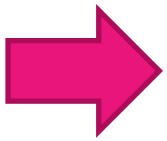
Une histoire qui se répète…



stephanie.chollet@grenoble-inp.fr

Synthèse de l'exemple

- Une histoire qui se répète… ?
  - Des besoins pas clairs
  - Manque de spécifications, de préparation des tests…
  - Mauvaises estimations
  - Manque de support de la hiérarchie
  - Manque d'expérience du chef d'équipe
  - Manque d'implication des intervenants
  - Problèmes humains : communication, compétences…



Domaine d'étude du Génie Logiciel

stephanie.chollet@grenoble-inp.fr

Quizz connaissances en informatique

Un peu d’histoire…

Histoire de l'informatique et naissance du Génie Logiciel

Un peu d'histoire…

- Dans les années 1960 :
  - Evolution du matériel, premiers langages
  - Emergence d'un nouveau métier : PROGRAMMEUR
    - Distinction entre les utilisateurs et les programmeurs
    - Distinction entre spécification et programmation
  - Quelques projets de grande taille :
    - Dans des organismes scientifiques (MIT, IBM)
    - Supportés par quelques petits groupes d'experts (pionniers)
  - Quelques problèmes mais beaucoup d'espoir

stephanie.chollet@grenoble-inp.fr

Un peu d'histoire…

- Dans les années 1970 :
  - Evolution majeure du matériel
  - De nombreux projets de grande taille
    - Exemple : le système d'exploitation IBM OS-360

  - Le temps de la désillusion
    - Faible qualité, insatisfaction des utilisateurs, délais et budgets non respectés
    - Pas de passage à l'échelle des techniques existantes, nouveaux problèmes rencontrés
  - Organisation de conférences, apparition de nouveaux termes :
    - « Software crisis »
    - « Software engineering »

stephanie.chollet@grenoble-inp.fr

Un peu d'histoire…

- Dans les années 1970 (suite) :
  - D'après une étude américaine sur 100 projets :
    - Retards : 52%
    - Dépassement de budget : 72%
    - Dépassement du budget matériel : 15%
    - Faible qualité : 30 à 85 bugs pour 1000 instructions

- **Identification de problèmes :**
  - Mauvaise compréhension des objectifs
  - Gestion humaine (manque de motivations, départs…)
  - Evolution technique, instabilité des besoins
  - …

stephanie.chollet@grenoble-inp.fr

Un peu d'histoire…

- **Solutions proposées au fil des années :**
  - De meilleures habitudes de gestion de projet
  - Des nouveaux paradigmes de programmation (et des langages)
  - Utilisation de méthodes formelles
  - Création de standards, de normes…

- **Leçons retenues :**
  - Il n'y a pas une seule solution gagnante !
  - Il a été reconnu que le développement de logiciel était une pratique d'ingénierie :
    - Le logiciel est une pratique d'ingénierie en soi
    - Des méthodes, techniques adaptées sont nécessaires
  - **Le Génie Logiciel est né !**
    - 1968 à la conférence de l'OTAN

stephanie.chollet@grenoble-inp.fr

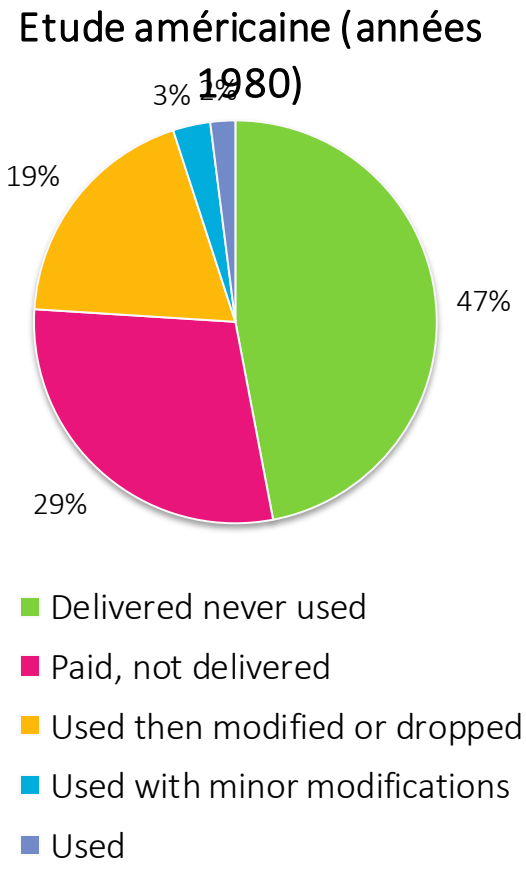
Un peu d'histoire…

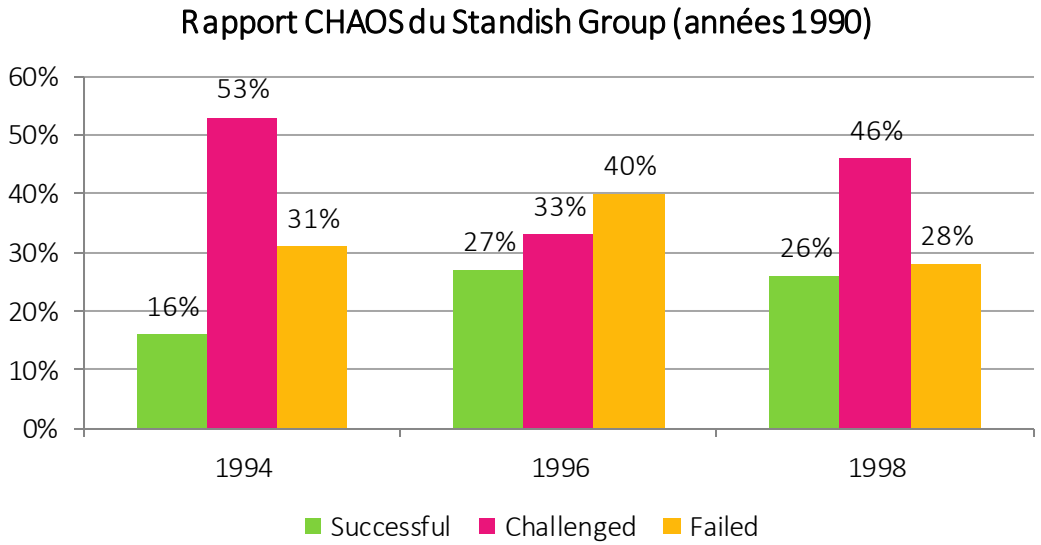
- Dans les années 1980 – 1990 :
  - De nombreuses améliorations dans de nombreux domaines :
    - Interconnexion des ordinateurs (Internet)
    - De meilleures organisations, mieux adaptées
    - Suivi des processus, amélioration permanente
    - Séparation claire entre spécifications / modèles / programmes
    - Conception par objets, programmation structurée
    - Implication des utilisateurs
  - La place des logiciels s'étend considérablement
    - Les projets grossissent encore et encore
    - Les systèmes logiciels envahissent de nombreux domaines applicatifs
  - Malgré tout, les problèmes demeurent !

stephanie.chollet@grenoble-inp.fr

Un peu d'histoire…

- Dans les années 1980 – 1990 (suite) :





- 53% des projets dépassent les ressources allouées
  - Coût : 198% et Temps : 222%
- En 1994, parmi les 16% des projets qui réussissent
  - Seulement 61% répondent aux besoins de départ
- A noter : dans les grandes entreprises, seuls 9% des projets réussissent.

stephanie.chollet@grenoble-inp.fr

Quelques echecs

- Ariane 5 – vol 501
  - Début du programme Ariane 5 : 1987
  - Lancement : 4 juin 1996
  - Nouveau fleuron aérospatial
    - 500 millions de dollars
    - 1100 industriels

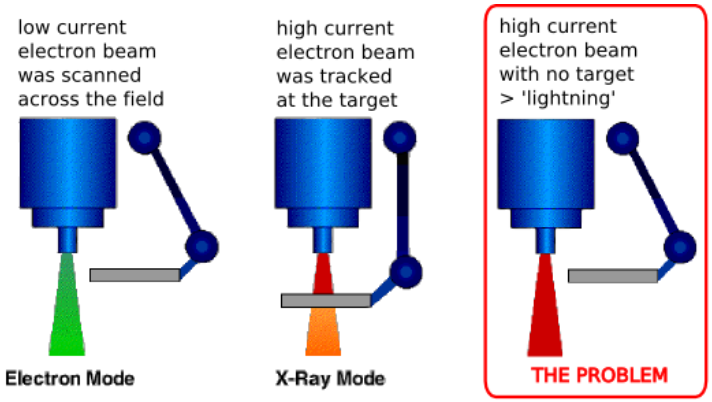
  - Aucune victime !

  - Echec dû notamment à une erreur informatique dans le programme de gestion de gyroscopes conçu pour Ariane 4, pas testé dans la configuration Ariane 5. Programme d'ailleurs inutile pour Ariane 5.

stephanie.chollet@grenoble-inp.fr

Quelques echecs

- Therac 25 (Entre 1985 et 1987)
  - Machine de radiothérapie développée conjointement entre le Canada et la France
    - 6 morts
    - De nombreuses personnes irradiées



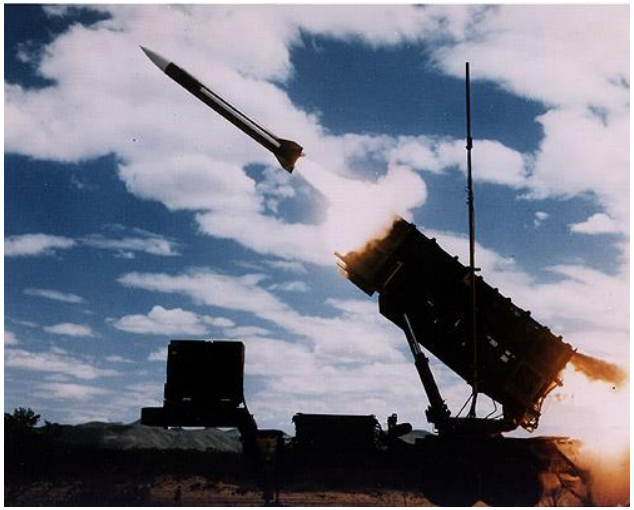
tray including the target, a flattening filter, the collimator jaws and an ion chamber was moved OUT for "electron" mode, and IN for "photon" mode.

  - Les causes :
    - Négligence du test
    - Documentation inadéquate
    - Programmé en assembleur
    - Réutilisation de briques logicielles (du Therac 20)

stephanie.chollet@grenoble-inp.fr

Quelques echecs

- Missile PATRIOT (1991)
  - Système de défense anti-missile
    - Non interception d'un missile SCUD
    - 28 morts, 98 blessés
  - Les causes :
    - Erreur d'approximation
    - Non-respect des hypothèses
    - Composants logiciels non adaptés
    - Délai de propagation d'une mise à jour



stephanie.chollet@grenoble-inp.fr

Autres échecs…

- AT&T en 1992 :
  - Interruption du service de téléphonie pendant 5h à l'est des Etats-Unis
  - Raison : Propagation en chaîne de messages d'erreur
- Oslo, en septembre 1993
  - Erreur dans le système de comptage des votes du parlement
- Aéroport de Denver en 1994
  - Logiciel de suivi des bagages
  - Coût 3 milliards de dollars, retard de 18 mois
- …

stephanie.chollet@grenoble-inp.fr

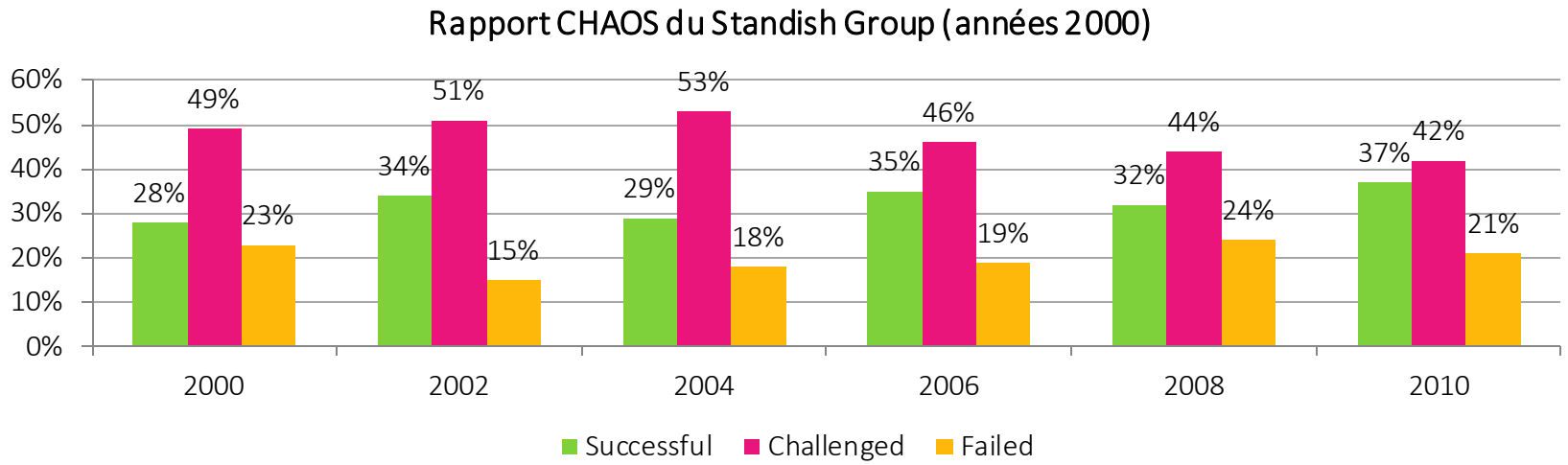
Un peu d'histoire…

- Dans les années 2000-2010 :
  - La place du logiciel continue de s'étendre :
    - Ère de la mobilité et des données partagées
    - Multiplication des systèmes embarquées pour le grand public (téléphone, tablette…)
    - Emergence des objets connectés, développement de l'Internet des objets
    - Multiplication des applications pour objets connectés
    - Développement du cloud computing
  - De nouvelles méthodes de développement : méthodes Agile
  - De nouveaux paradigmes de programmation : programmation orientée services, programmation par contrat…
  - Soutenus par de nouveaux outils/technologies de gestion de projet, de développement et de mise en production

stephanie.chollet@grenoble-inp.fr

Un peu d'histoire…

- Dans les années 2000 :

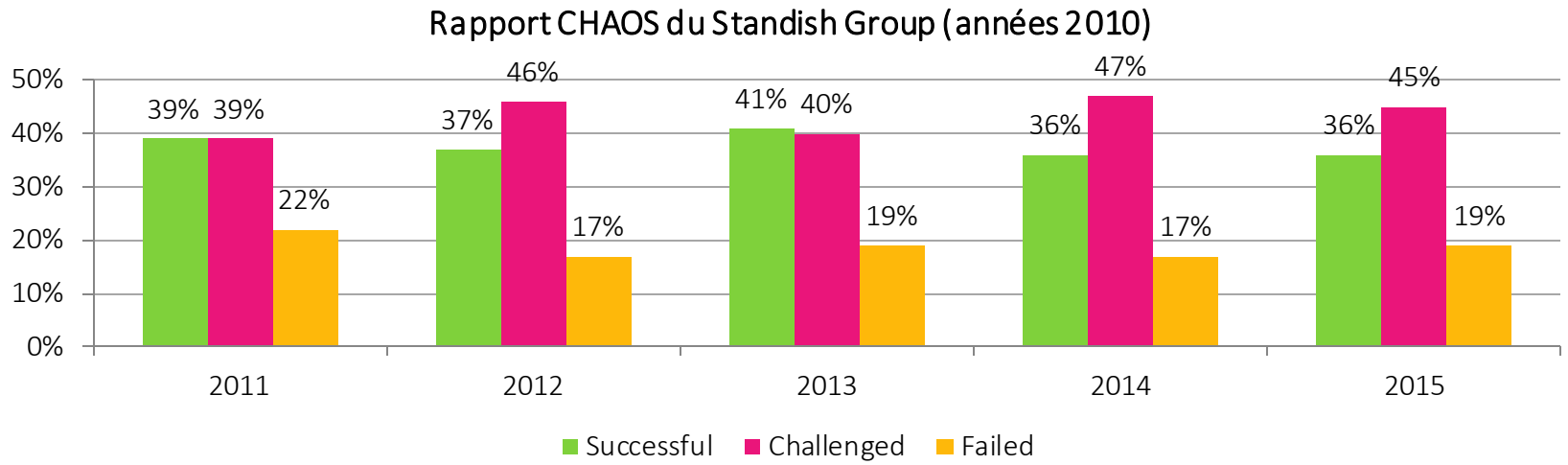


- 49% des projets dépassent les ressources allouées
  - Coût : **43%** et Temps : **63%**
- En 2000, parmi les 28% des projets qui réussissent
  - Seulement 67% répondent aux besoins de départ
- Résultats encourageants !

stephanie.chollet@grenoble-inp.fr

Un peu d'histoire…

- Dans les années 2010 :



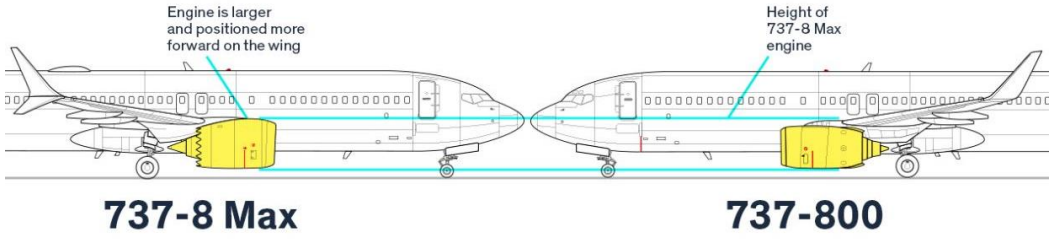
- 56% des projets dépassent les ressources allouées
  - Coût : **44%** et Temps : **60%**

- Progrès faibles depuis les années 2000 !

stephanie.chollet@grenoble-inp.fr

Quelques echecs

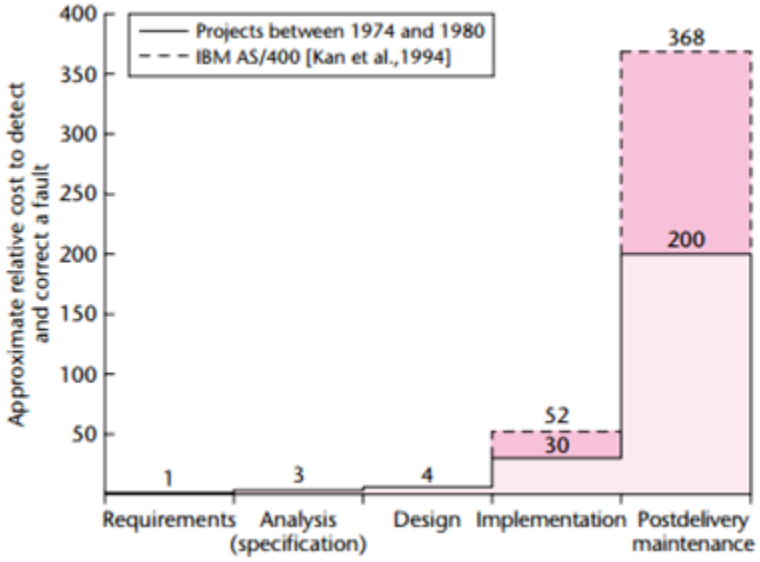
- Voitures General Motors en 2014
  - Non détection des crashs considérés comme des tests : 1 mort et 3 blessés
- Starbucks en 2015
  - Impossibilité d'encaissement suite à une mise à jour pendant 2h pour 14000 points de vente : 3,5 millions d'euros
- Chrysler en 2015
  - Prise de contrôle à distance (15km) d'une Jeep roulant à 100km/h : 64 millions d'euros
- F-35 en 2016
  - Avion de combat américain dont le radar crashe aléatoirement, problème de calcul des coordonnées de vol, gestion des appareils amis/ennemis défectueuse : 20 à 100 milliards de dollars
- Boeing 737 Max en 2019
  - 2 crashs d'avion (346 morts)



stephanie.chollet@grenoble-inp.fr

Coût

- Les bugs et les échecs sont un véritable préjudice pour les entreprises :
  - Perte financière lorsqu'un projet est annulé
    - L'argent déjà dépensé est perdu
  - Occasions manquées
  - Lancements commerciaux manqués
  - Manque des fonctions désirées ou nécessaires (dans un logiciel existant)
    - Manque de productivité
    - Manque de services
    - Manque de support
  - Coût du débogage (maintenance corrective)

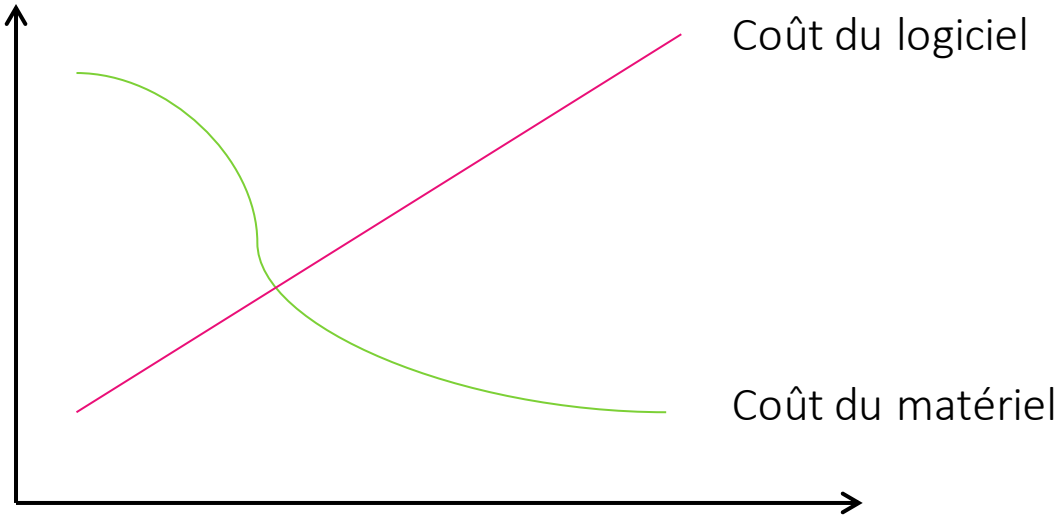


Coût des défauts

stephanie.chollet@grenoble-inp.fr

Pourquoi le logiciel est complexe ?

Hardware vs. Software



- Le matériel devient moins cher et plus puissant
- Pourquoi ce n'est pas le cas du logiciel ?

stephanie.chollet@grenoble-inp.fr

D'où vient la complexité ?

- Nous faisons des choses difficiles dans les logiciels
  - Un tâche simple et compréhensible est difficile à implanter !
- Le logiciel est intrinsèquement difficile à développer et à maintenir
- Les attentes sont croissantes en terme de qualité (fiabilité forte)
- Nous avons fait des progrès en Génie Logiciel mais
  - Les logiciels sont partout
  - Explosion du nombre de fonctionnalités

Complexité du logiciel

- Le logiciel est intrinsèquement complexe à produire et à maintenir pour de nombreuses raisons

- Un logiciel est :
  - Unique
  - Malléable
  - Intangible
  - Complexe et de grande taille
  - L'humain entre en jeu
  - Etrange !

stephanie.chollet@grenoble-inp.fr

Qualités attendues d'un logiciel ?

- Qualités externes :
  - Visibles par les utilisateurs
  - Fiable, efficace, utilisable

- Qualités internes :
  - Préoccupations des développeurs
  - Doivent faciliter le développement des qualités externes
  - Vérifiable, maintenable, extensible…

stephanie.chollet@grenoble-inp.fr

Quelques qualités logicielles – 1/2

- **Capacité fonctionnelle** (externe) :
  - Capacité qu'ont les fonctionnalités à répondre aux exigences et besoins définis dans les spécifications
  - Sous-catégories : précision, interopérabilité, conformité aux normes, sécurité

- **Facilité d'utilisation** (externe) :
  - Effort nécessaire pour apprendre à manipuler le logiciel
  - Sous-catégories : facilité de compréhension, d'apprentissage et d'exploitation, robustesse
  - Une utilisation incorrecte n'entraîne pas de dysfonctionnement

- **Fiabilité** (externe) :
  - Capacité d'un logiciel de rendre des résultats corrects quelles que soient les conditions d'exploitation
  - Sous-catégorie : tolérance aux pannes

stephanie.chollet@grenoble-inp.fr

Quelques qualités logicielles – 2/2

- **Performance** (externe) :
  - Rapport entre la quantité de ressources utilisées (moyens matériels, temps, personnel) et la quantité de résultats délivrés
  - Sous-catégories : temps de réponse, débit, extensibilité

- **Maintenabilité** (interne) :
  - Effort nécessaire à corriger ou transformer le logiciel
  - Sous-catégorie : extensibilité

- **Portabilité** (interne) :
  - Aptitude d'un logiciel à fonctionner dans un environnement matériel ou logiciel différent de son environnement initial
  - Sous-catégorie : facilité d'installation et de configuration

stephanie.chollet@grenoble-inp.fr

Evaluation de la qualité logicielle

- La qualité doit être mesurable
  - Pour l'évaluer
  - Pour l'améliorer
  - Contre-exemple : « Le code doit être commenté » - non mesurable
  - Exemple : « Le code doit comprendre au moins 20% de commentaires » - mesurable et améliorable
- La mesure implique la qualité doit être clairement définies
  - Même si elle est incomplète
- De nombreuses métriques existent pour ces différents critères de qualité
  - Exemple : nombre de lignes de code, complexité cyclomatique, le nombre de commentaires, indice de maintenabilité…

stephanie.chollet@grenoble-inp.fr

Synthèse

- Le logiciel doit faire face à des demandes croissantes, notamment en terme de qualité logicielle

- Des techniques spécifiques doivent être définies et utilisées pour des projets logiciels
  - Le **Génie Logiciel** est une **discipline** !

stephanie.chollet@grenoble-inp.fr

Le Génie Logiciel

Définition

- Le Génie Logiciel est l'étude et l'utilisation de **processus systématiques** et de **technologies** pour supporter les activités de **développement logiciel** et de **maintenance**.

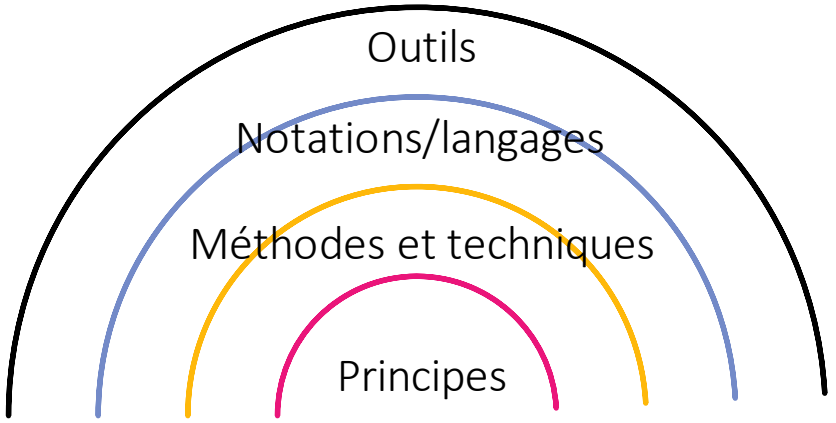
- Il faut :
  - Contrôler le coût
  - Contrôler les délais
  - Assurer la qualité

- En lien avec tous les aspects de la production logicielle :
  - Des spécifications jusqu'à la maintenance

stephanie.chollet@grenoble-inp.fr

Génie Logiciel

- Le Génie Logiciel repose un ensemble de principes mis en œuvre par des méthodes, des techniques et des outils



[Ghezzi, 1993]

- Principes du Génie Logiciel :
  - Rigueur
  - Modularité
  - Abstraction
  - Séparation des préoccupations

stephanie.chollet@grenoble-inp.fr

Principe : Rigueur

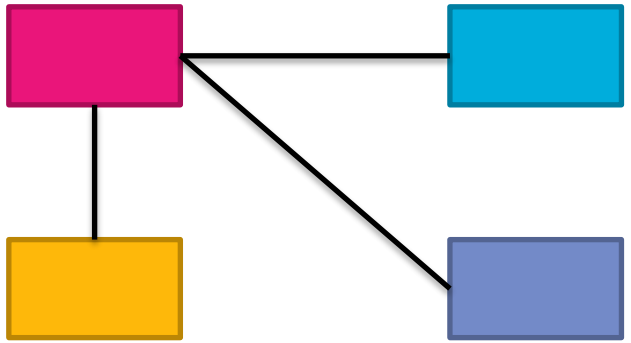
- Comme tous domaines d'ingénierie, le génie logiciel nécessite de la rigueur :
  - Définition de processus, de techniques, de méthodes
  - Définition des documents associés, des deadlines
  - Validation
  - Attitude professionnelle

- A noter :
  - La rigueur ne tue pas la créativité
  - La rigueur n'est pas équivalente aux techniques mathématiques

stephanie.chollet@grenoble-inp.fr

Principe : Modularité

- Le principe est de remplacer le problème initial par des modules de moindre complexité :
  - Chaque module traite une partie du problème
  - Chaque module est compréhensible, homogène et indépendant
  - Les modules sont liés

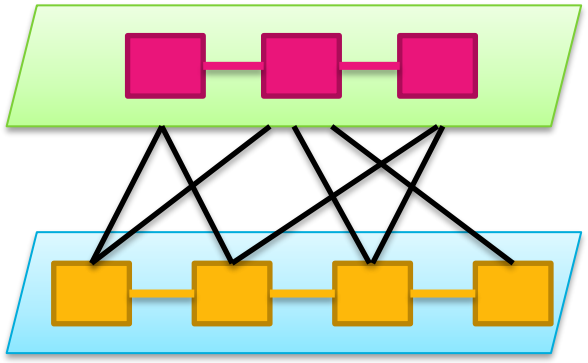


- A noter :
  - On recherche un **faible couplage**
  - On recherche une **forte cohésion**

stephanie.chollet@grenoble-inp.fr

Principe : Abstraction

- Organiser les informations (ou modules) suivant différents niveaux :
  - Définition de niveaux de généralisation
  - A un niveau donné, on ne considère que les informations ayant le même niveau sémantique
  - Un niveau doit être compréhensible, homogène et complet



- A noter :
  - On recherche des **niveaux clairement découplés**
  - On recherche un **passage aisé d'un niveau à l'autre**

stephanie.chollet@grenoble-inp.fr

Principe : Séparation des préoccupations

- Se concentrer sur un seul aspect du problème à la fois et le traiter de façon indépendante

- Exemples :
  - Séparation des rôles des différents acteurs d'un projet
  - Séparation des phases de développement
  - Séparation des propriétés fonctionnelles et non-fonctionnelles (sécurité)

- A noter :
  - Il faut choisir des aspects suffisamment indépendants

stephanie.chollet@grenoble-inp.fr

Domaines du Génie Logiciel

- Processus logiciel
  - Définition des activités, des rôles…
- Langages de programmation
  - Structuré, objet, fonctionnel…
- Méthodologies
  - Fonctionnelle, objet (UML)…
- Gestion de projet logiciel
  - Cycle de vie, planning, gestion des risques, gestion de la sous-traitance, gestion humaine
- …

stephanie.chollet@grenoble-inp.fr

Les activités logicielles

Projet informatique

- Ensemble des **activités** et des **actions** à entreprendre pour **répondre au besoin d'informatisation** d'un ensemble de tâches dans un **contexte défini**

- Un projet doit concilier :
  - Les objectifs fonctionnels
  - Les spécifications (aspects techniques)
  - Les contraintes temporelles
  - Les contraintes budgétaires
  - Les contraintes matérielles (ressources allouées)

stephanie.chollet@grenoble-inp.fr

Les activités

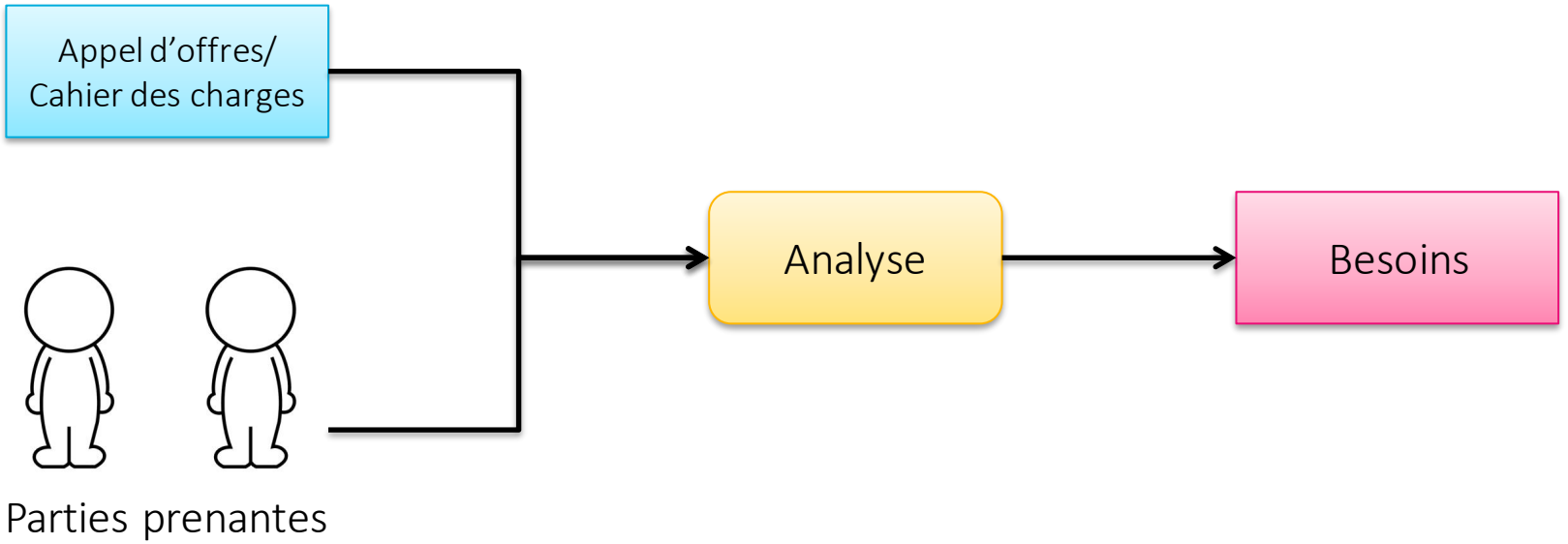
- Le processus de développement logiciel comprend un ensemble d'activités :
  - Analyse des besoins (*Requirements*)
  - Conception (*Design*)
  - Implantation (*Implementation*)
  - Validation (*Validation*)
  - Intégration (*Integration*)
  - Déploiement (*Deployment*)
  - Maintenance (*Maintenance*)

- Avec des activités transverses et permanentes :
  - Documentation
  - Gestion de projet
  - Gestion de la qualité
  - Gestion des risques
  - Gestion de la sous-traitance
  - …

stephanie.chollet@grenoble-inp.fr

Analyse des besoins/Spécifications

- Objectifs :
  - Identifier ce que le client veut et ses contraintes
  - Spécifier ses besoins



stephanie.chollet@grenoble-inp.fr

Exemple de spécifications

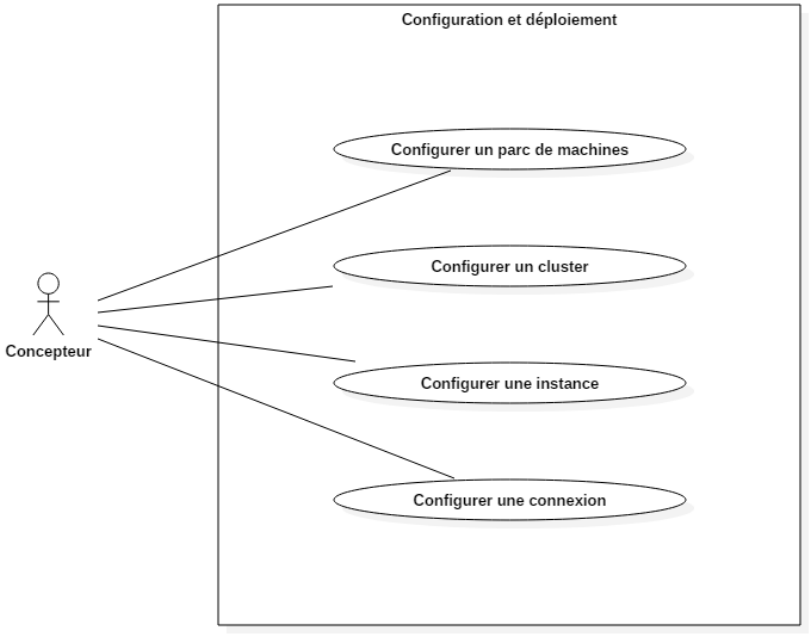


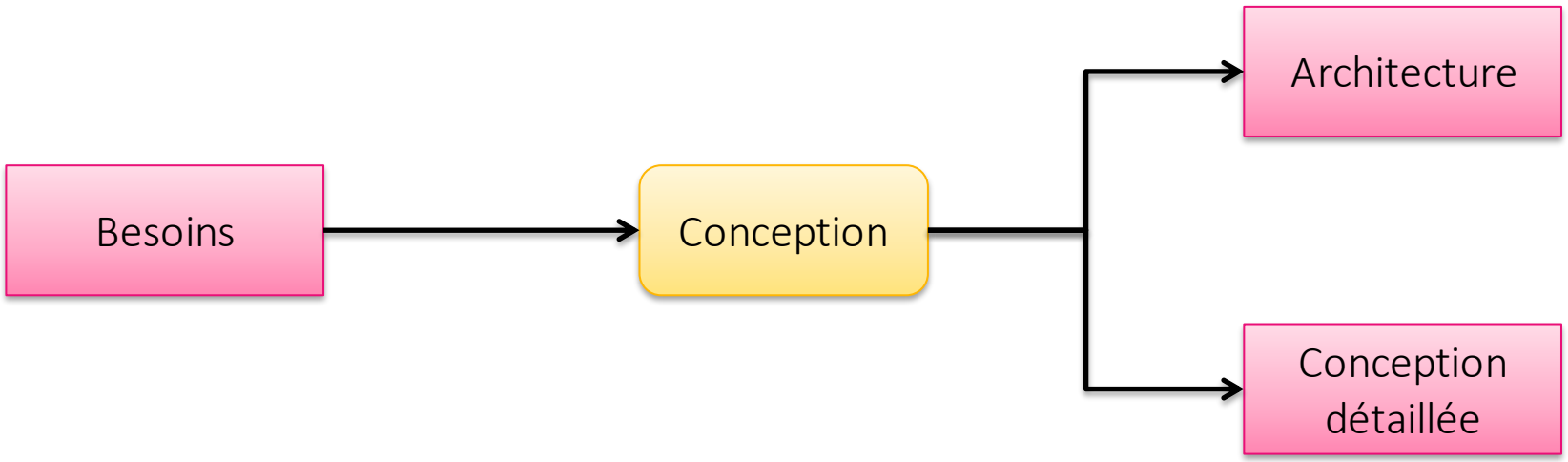
Diagramme de cas d'utilisation

CU101 : Configurer un parc de machines	
But	Ajouter une machine à un parc de machines
Acteur	Concepteur
Pré-condition	Avoir créé ou ouvert un projet
Scénario nominal	1. Créer une machine dans le parc de machines 2. Nommer la machine 3. Mettre à jour les propriétés de la machine
Post-condition	La machine est créée dans le parc de machines
Exception	Machine sans adresse physique

stephanie.chollet@grenoble-inp.fr

Conception

- Objectifs :
  - Définition de l'organisation logique du code
  - Fournir une solution au problème issu de l'analyse



stephanie.chollet@grenoble-inp.fr

Exemple de modèles de conception

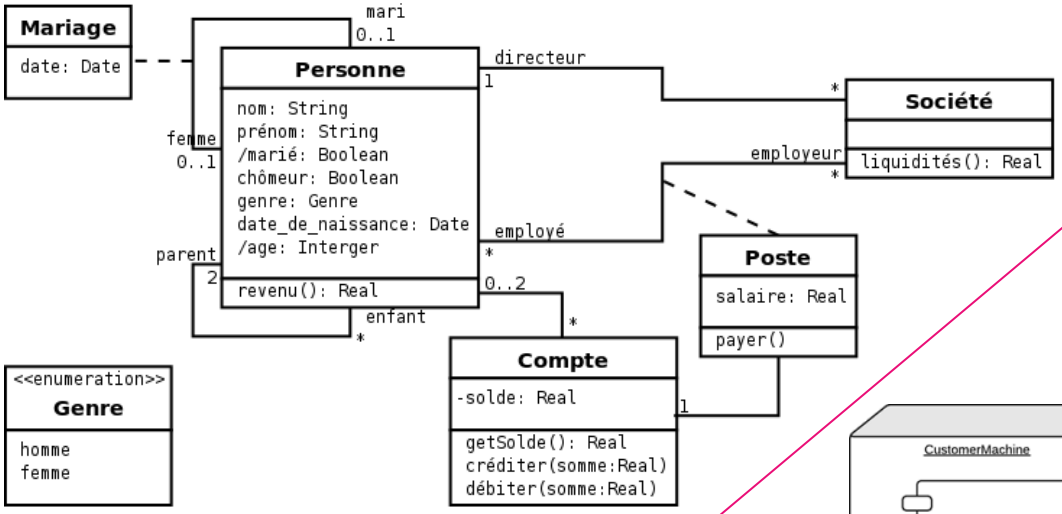
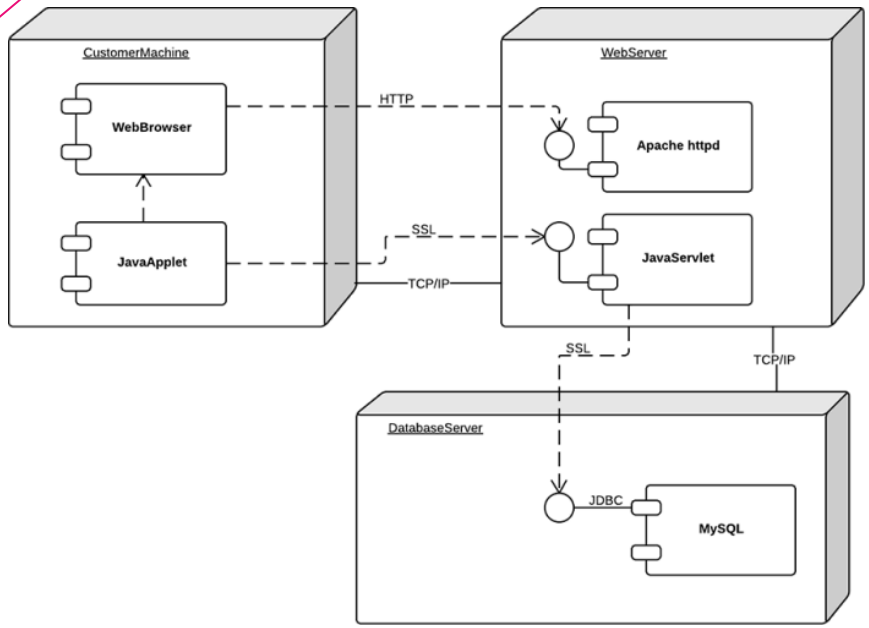


Diagramme de classes

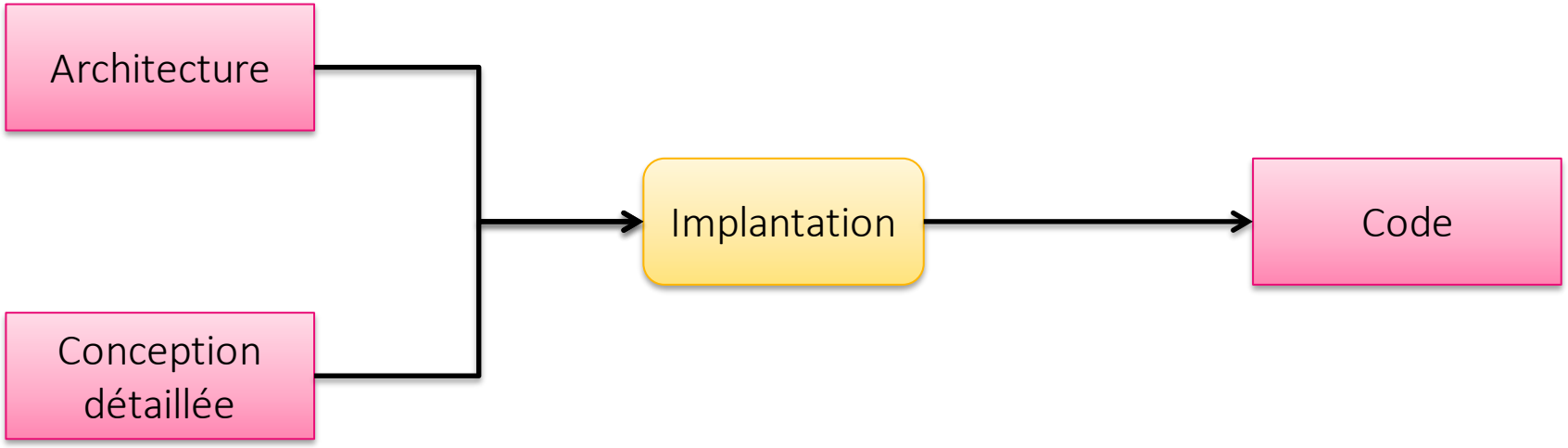
Diagramme de déploiement



stephanie.chollet@grenoble-inp.fr

Implantation

- Objectif :
  - Développer le code correspondant aux spécifications
  - Produire un exécutable réalisant la conception
  - Ajout d'optimisation si nécessaire



stephanie.chollet@grenoble-inp.fr

Exemple de codes

```
        .file   "test.c"
        .section        .rodata
.LC0:
        .string "hello world"
        .text
        .globl  main
        .type   main, @function
main:
.LFB0:
        .cfi_startproc
        pushq   %rbp
        .cfi_def_cfa_offset 16
        .cfi_offset 6, -16
        movq    %rsp, %rbp
        .cfi_def_cfa_register 6
        movl    $.LC0, %edi
        movl    $0, %eax
        call    printf
        movl    $0, %eax
        popq    %rbp
        .cfi_def_cfa 7, 8
        ret
        .cfi_endproc
.LFE0:
        .size   main, .-main
        .ident  "GCC: (Ubuntu 5.4.0-6ubuntu1~16.04.4) 5.4.0 20160609"
        .section        .note.GNU-stack,"",@progbits
```

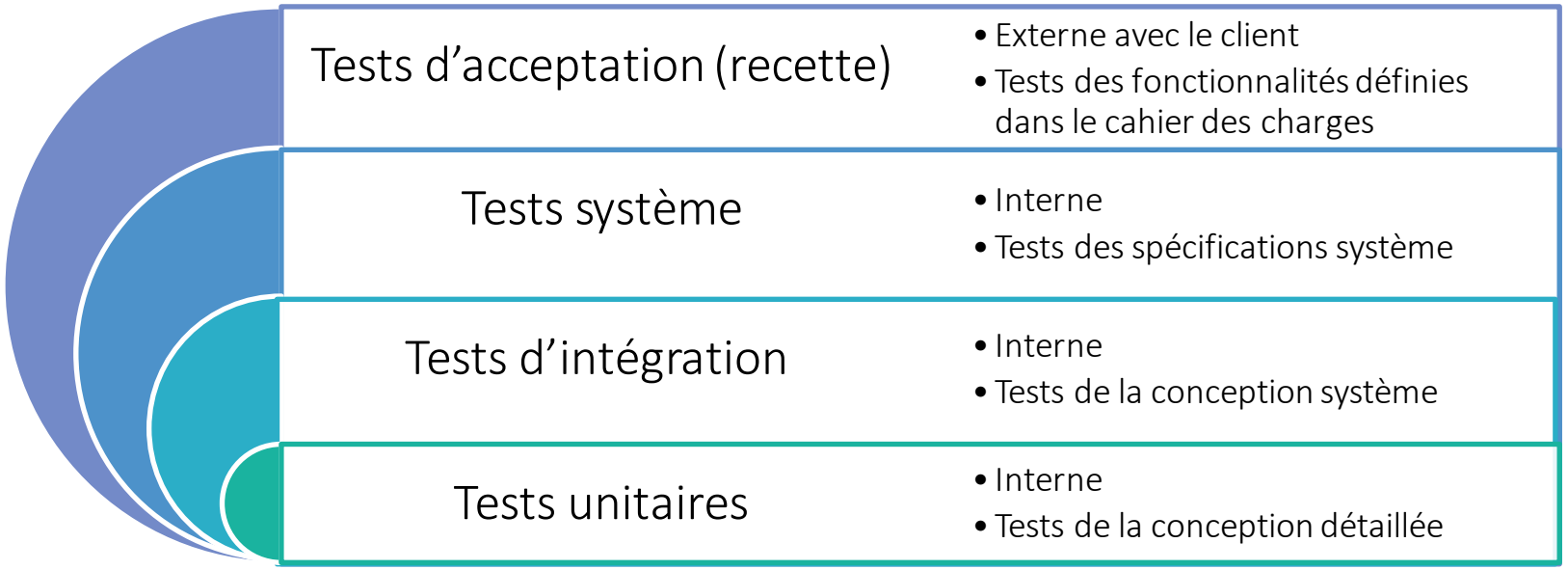
hr.rb

```ruby
class Employee
  def initialize(name, salary, hire_year)
  @name = name
  @salary = salary
  @hire_year = hire_year
end

  def to_s
    "Name is #{@name}, salary is #{@salary}, " +
    "hire year is #{@hire_year}"
  end

  def raise_salary_by(perc)
    @salary += (@salary * 0.10)
  end
end

class Manager < Employee
  def initialize(name, salary, hire_year, asst)
  super(name, salary, hire_year)
  @asst = asst
end

  def to_s
    super + ",\tAssistan info: #{@asst}"
  end

  def raise_salary_by(perc)
    perc += 2007 - @hire_year
    super(perc)
  end
end
```


stephanie.chollet@grenoble-inp.fr

Validation

- Objectif :
  - Tester et valider les différents artéfacts logiciels et matériels
  - Activité transverse/permanente



stephanie.chollet@grenoble-inp.fr

Exemple de techniques de validation

- Audit

Vérification des métriques de qualité (ex : quantité de commentaires…)

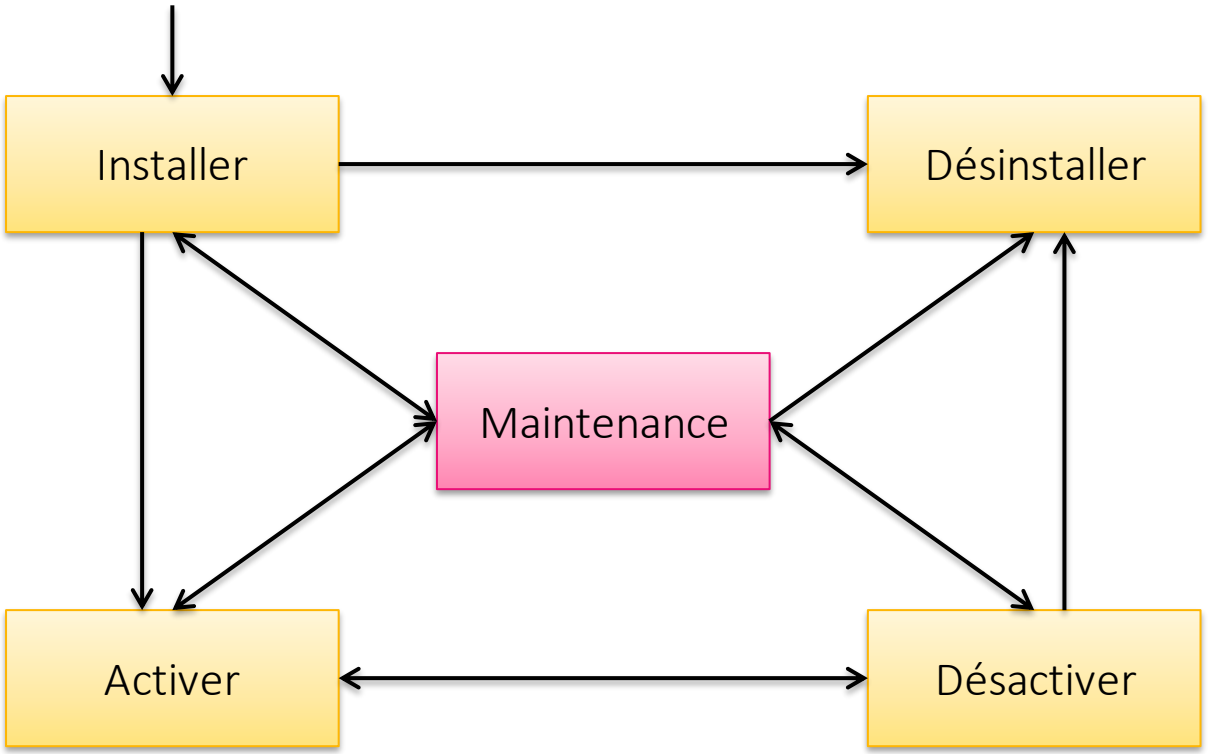
- Test

Evaluation du nombre de tests réalisés, acceptés, en défaut…

stephanie.chollet@grenoble-inp.fr

Intégration et déploiement

- Objectif :
  - Assembler les composants et vérifier le produit dans son ensemble
  - Installer le produit fini chez le client



stephanie.chollet@grenoble-inp.fr

Maintenance

- Objectifs :
  - Maintenir un logiciel pendant qu'il est en fonctionnement
  - Déterminer si le logiciel fonctionne encore correctement

- Types de maintenance :
  - Corrective : élimination d'un problème lors du fonctionnement
  - Prédictive : élimination d'un potentiel problème avant son apparition
  - Adaptative : évolution du logiciel due à un changement d'environnement
  - Evolutive : modification des fonctionnalités du logiciel

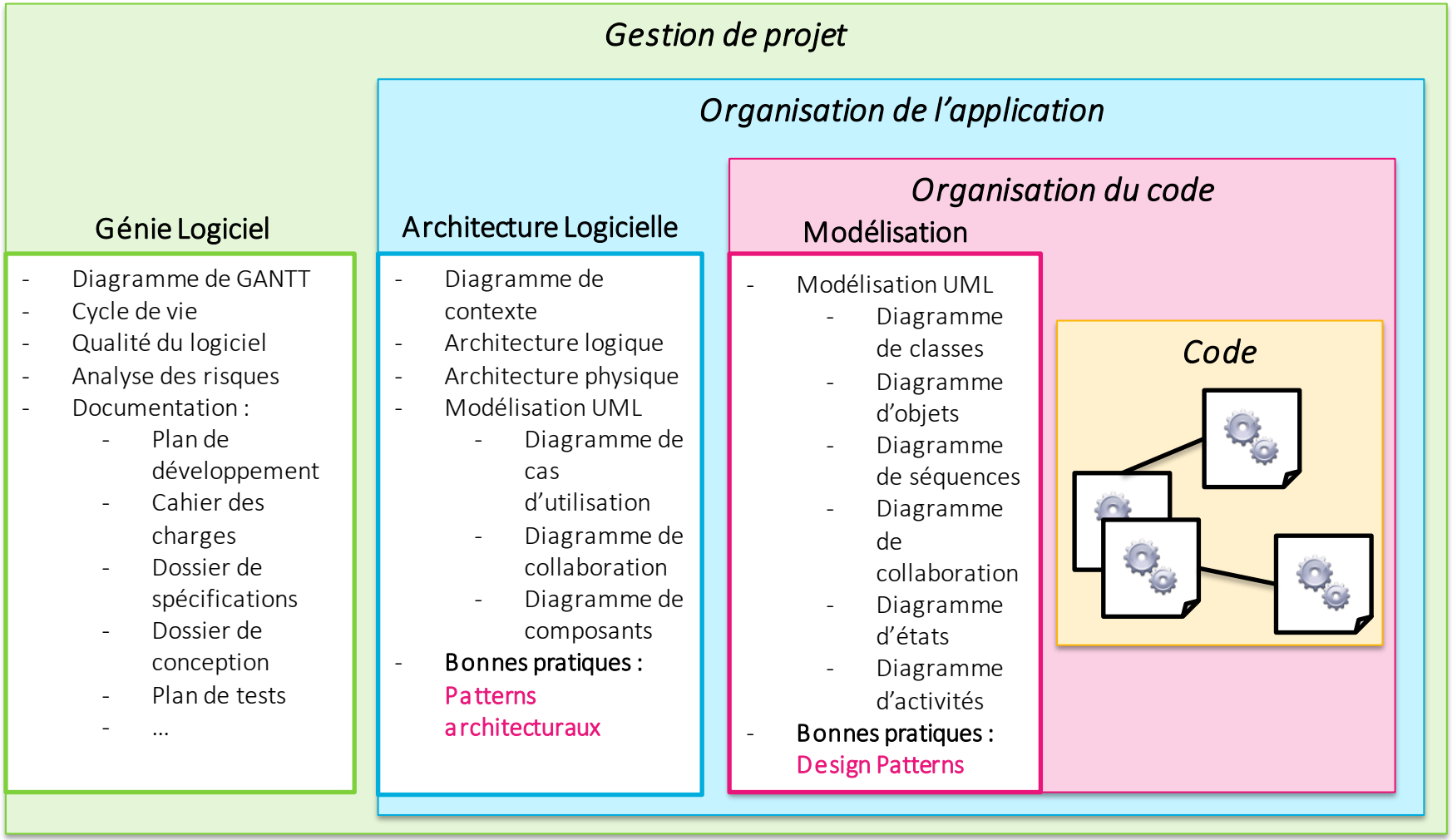
Conclusion

De la programmation logicielle au Génie Logiciel

Programmation logicielle	Génie Logiciel
• Un seul développeur	• Une équipe de développeurs • De nombreux rôles
• Une application « jouet »	• Un système complexe
• Courte durée de vie	• Durée de vie longue voire infinie
• Un seul ou plusieurs intervenants • Développeur = utilisateur	• De multiple intervenants • Développeur ≠ utilisateur • Utilisateur ≠ client
• Un système unique	• Une famille de systèmes
• Construit à partir de rien	• Réutilisation pour amortir les coûts
• Maintenance minimale	• Maintenance représente 60% du coût total de développement

stephanie.chollet@grenoble-inp.fr

Du projet au code

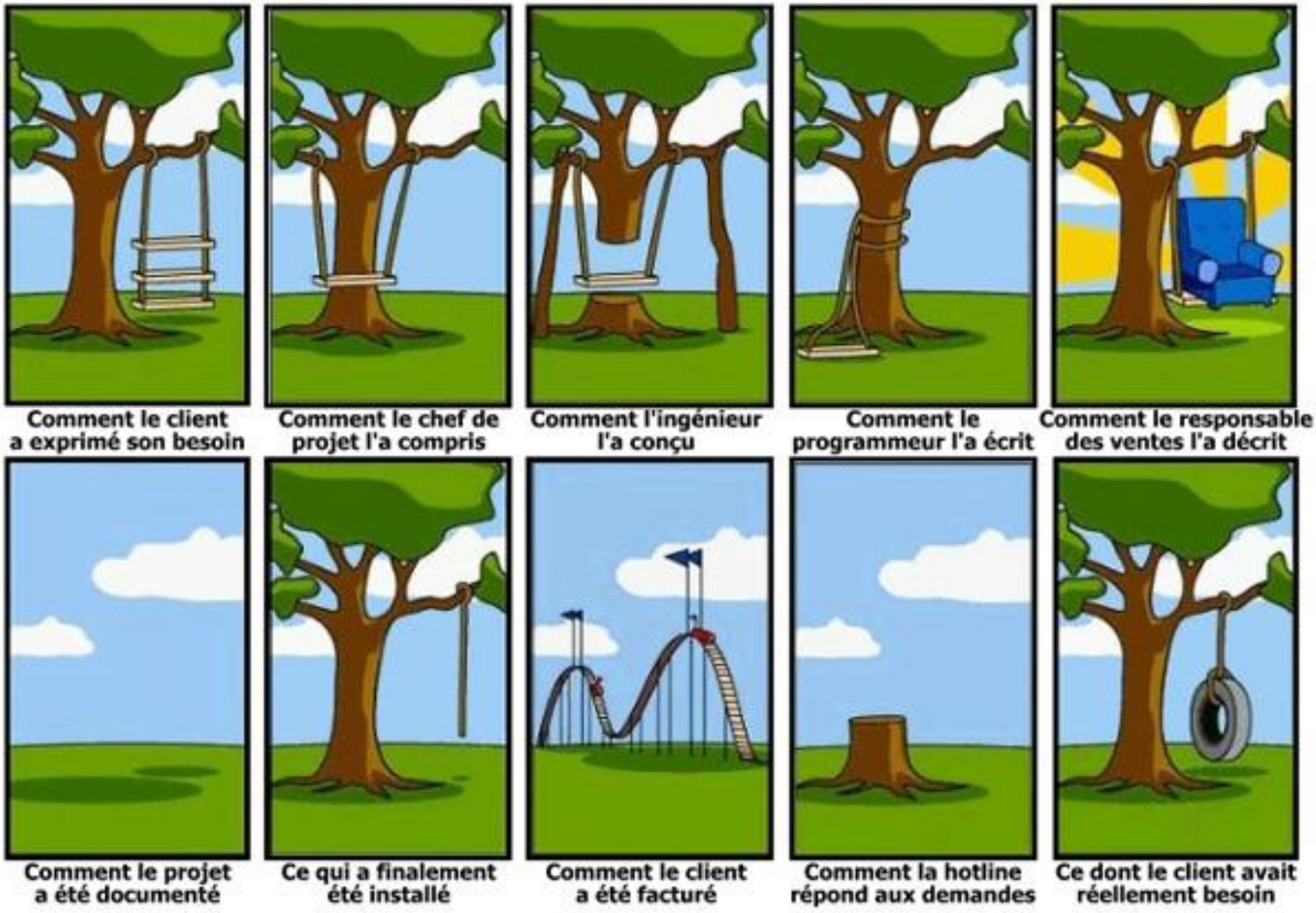


stephanie.chollet@grenoble-inp.fr

Synthèse

- Le génie logiciel est une démarche d'ingénierie qui poursuit les objectifs suivants :
  - Prédictible
  - Reproductible
  - Evaluable
- Le génie logiciel repose sur quatre grands principes :
  - La rigueur
  - La modularité
  - L'abstraction
  - La séparation des préoccupations

stephanie.chollet@grenoble-inp.fr



stephanie.chollet@grenoble-inp.fr