INGÉNIERIE DES SYSTÈMES

Systèmes complexes et logiciels sécuritaires

Objectif du cours

- ✓ Introduction au développement de logiciels sécuritaires
- ✓ Focus sur les tests
- ✓ Introduction à l'ingénierie des systèmes

Julien VANDERSTRAETEN

Systerel

julien.vanderstraeten.ups@gmail.com

UPSSITECH ÉCOLE D'INGÉNIEURS

UNIVERSITÉ TOULOUSE III PAUL SABATIER Université de Toulouse

Introduction au développement de logiciels sécuritaires

Qu’est ce qu’un logiciel sécuritaire ?

➔ La parole est à vous !

Logiciels sécuritaires

Logiciels sécuritaires = logiciels critiques

Mauvais fonctionnement d'un logiciel critique ➔ impact important sur la sécurité ou la vie des personnes, des entreprises ou des biens.

➔ Nécessité de limiter les risques de dysfonctionnements

L'industrie en France et à l'international

➢ Contexte industriel

- ✓ Tendu commercialement
  - Les grandes puissances mondiales se font une guerre à coup de règlementations et taxes douanières
- ✓ Guerre technologique
  - Courses aux nouvelles technologies

➢ Marché français, européen et international

- ✓ Concurrence entre les Etats-Unis et la Chine
- ✓ La place de l'Europe dans tout ça ?
- ✓ L'industrie demeure le premier moteur de l'activité économique européenne

➢ Les opportunités

- ✓ Fusions des industriels
- ✓ Pression du marché mondial

=> recherche de techniques pour améliorer la productivité

➢ Qu'est ce que c'est (au sens informatique) ?

✓ Système embarqué

- Système informatique fonctionnant sur des produits industriels ou destinés au grand public dont les moyens de calculs sont embarqués sur le procédé contrôlé (calculateurs, batterie, etc.).

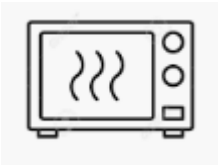
✓ Système temps réel

- Système informatique soumis à des contraintes de temps.
- L'exactitude des applications ne dépend pas seulement du résultat mais également du temps auquel ce résultat est produit.

✓ Système critique

- Système informatique pour lequel une défaillance, totale ou partielle, du système, peut entrainer des conséquences graves (économiques, humaines, écologiques, etc.).

Les applications



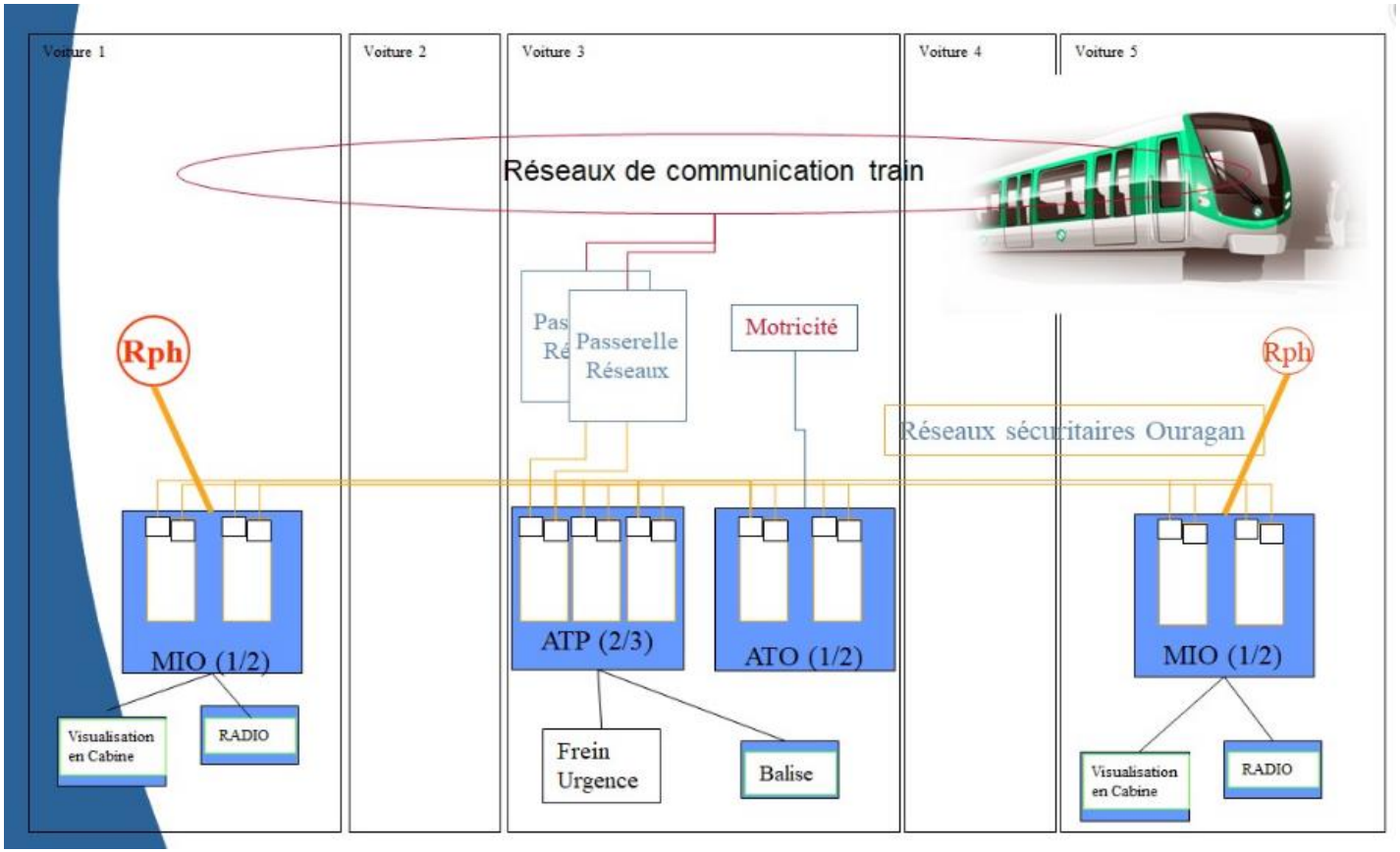



- ✓ Système de contrôle – commande
  - 1 - Évaluation d'un capteur ou contrôleur
  - 2 - Application d'un ordre sur un actionneur
- ✓ Drone VS planeur -> fréquence de contrôle de stabilité

Caractéristiques des systèmes de contrôle-commande

- ✓ **Diversité** des dispositifs d'entrées/sorties (capteur analogique, numérique, E/S TOR, bus de communication)
- ✓ Prise en compte des **comportements concurrents** (systèmes asynchrones)
- ✓ Respect des **contraintes temporelles** (temps de cycle, traitement des entrées, élaboration des sorties)
- ✓ **Sûreté de fonctionnement** : coût et vies humaines en jeu (respect des normes, démonstration de sûreté, etc.)

- Un exemple d'architecture (Pilote Automatique Embarqué dans un train type métro)



Les domaines industriels concernés

- Ferroviaire

- Aérospatiale

- Aéronautique

- Automobile

- Modèles classiques :
  - ✓ Le modèle (encore) très utilisé (et oui !) : cycle en V

LA MÉTHODE EN V

  - ✓ Autres méthodes : modèle en cascade, spirale, incrémental, prototypage, etc.

- Méthodes agiles
  - ✓ Scrum, Rad, XP, etc.

➢ Comparatif des méthodes « Classique » VS Agile

✓ Classique :

- + respect de la norme aisé
- + suivi des étapes du cycle en V
- - logiciel fonctionnel au bout du cycle seulement
- - gestion des reprises lourdes
- - pas ou peu d'interaction en cours de cycle

✓ Agile :

- + processus itératif et incrémental
- + adaptation au besoin si nécessaire en cours de développement
- + interactions fortes au sein des membres de l'équipe
- - nécessite une grande disponibilité de tous les acteurs
- Méthode dépendante de la taille de l'équipe (conseillée : 3 à 9)

➢ Les Normes et les niveaux d'intégrité

✓ **Norme** :

- Document établi par consensus et approuvé par un organisme reconnu, qui fournit, pour des **usages communs et répétés**, des règles, des lignes directrices ou des caractéristiques, pour des activités ou leurs résultats garantissant un **niveau d'ordre optimal** dans un contexte donné.

✓ **Niveaux d'intégrité** :

- Numéro de classification déterminant les techniques et mesures à appliquer de manière à réduire à un niveau convenable, les défauts résiduels du logiciel.

SSIL 4 : Très élevée	10-8 > THR > 10-9
SSIL 3 : Elevée	10-7 > THR > 10-8
SSIL 2 : Moyenne	10-6 > THR > 10-7
SSIL 1 : Faible	10-5 > THR > 10-6
SSIL 0 : Logiciel non lié à la sûreté	

- **THR (Tolerable Hazard Rate)** : taux maximum acceptable d'occurrence d'un danger par heure et par fonction

Des développements fortement cadrés

- Quelques exemples :
  - ✓ Ferroviaire : Norme EN 50128

  - ✓ Nucléaire : Norme CEI 60880
  - ✓ Avionique : Norme DO-178

Domaines :

- Aéronautique
- Défense
- Spatial
- Nucléaire
- Ferroviaire
- …

→ **Fiabilité, Disponibilité, Maintenabilité, Sécurité (FDMS)**

→ **Reliability, Availability, Maintainability, Safety (RAMS)**

Niveaux de criticité

Il existe différents niveaux de criticité d'un système.

Quel est l'impact d'un dysfonctionnement du système ? ➔ niveau de criticité

➔ Ces niveaux sont décrits par des normes (qui dépendent du domaine d'application) :

- Avionique (DO-178)
- Ferroviaire (EN 50128)
- Nucléaire (CEI 60880)
- …

Les normes citées précédemment (DO-178, EN 50128, CEI 60880) sont élaborées pour aider au développement de logiciels critiques.

Ces normes décrivent notamment, en fonction du niveau de criticité, les organisations à mettre en œuvre, la description des différentes phases de développement, les documents à produire…

Description des niveaux

Avionique (DO-178) : DAL (Development Assurance Level)

5 niveaux de criticité : DAL E (moins critique) à DAL A (plus critique)

- *DAL E* : aucun impact sur le fonctionnement de l'appareil ou la charge de travail du pilote.
- *DAL D* : un dysfonctionnement du logiciel provoquerait ou contribuerait à un dysfonctionnement mineur de l'appareil
- *DAL C* : un dysfonctionnement du logiciel provoquerait ou contribuerait à un dysfonctionnement majeur de l'appareil
- *DAL B* : un dysfonctionnement du logiciel provoquerait ou contribuerait à une condition dangereuse ou un dysfonctionnement sévère et majeur de l'appareil
- *DAL A* : un dysfonctionnement du logiciel provoquerait ou contribuerait à une condition de perte catastrophique de l'appareil

Le SIL (Safety Integrity Level) en détail :

1. Analyse de risques - Probabilité

Probabilité d'occurrence	Définition	Taux d'occurrence (défaillance par an)
Fréquent	Nombreuses fois dans la vie du système	$> 10^{-3}$
Probable	Plusieurs fois dans la vie du système	10^{-3} to 10^{-4}
Occasionnel	Une fois dans la vie du système	10^{-4} to 10^{-5}
Rare	Peu probable dans la vie du système	10^{-5} to 10^{-6}
Improbable	Très peu probable dans la vie du système	10^{-6} to 10^{-7}
Invraisemblable	Ne devrait jamais arriver dans la vie du système	$< 10^{-7}$

2. Analyse de risques – Conséquences / Gravité

Type de conséquence	Définition
Catastrophique	Pertes humaines multiples
Critique	Perte d'une vie humaine
Marginal	Blessures majeures à une ou plusieurs personnes
Insignifiant	Blessures mineures

3. Analyse de risque – Résultat / Criticité

	Type de conséquence			
Probabilité d'occurrence	Catastrophique	Critique	Marginal	Insignifiant
Fréquent	I	I	I	II
Probable	I	I	II	III
Occasionnel	I	II	III	III
Rare	II	III	III	IV
Improbable	III	III	IV	IV
Invraisemblable	IV	IV	IV	IV

Description des niveaux

Classe I : le risque est inacceptable en toute circonstance ;

Classe II : non désiré. Tolérable seulement si la réduction du risque n'est pas faisable ou si les coûts sont largement disproportionnés par rapport au gain de réduction du risque ;

Classe III : tolérable si le coût de réduction de risque dépasse le gain ;

Classe IV : acceptable, bien que nécessitant une surveillance.

Comment est défini le niveau de SIL

SIL	Faible sollicitation Probabilité moyenne de défaillance de la fonction (sur demande)	Forte sollicitation Probabilité de défaillance dangereuse (par heure)
1	$\geq 10^{-2}$ to $< 10^{-1}$	$\geq 10^{-6}$ to $< 10^{-5}$
2	$\geq 10^{-3}$ to $< 10^{-2}$	$\geq 10^{-7}$ to $< 10^{-6}$
3	$\geq 10^{-4}$ to $< 10^{-3}$	$\geq 10^{-8}$ to $< 10^{-7}$
4	$\geq 10^{-5}$ to $< 10^{-4}$	$\geq 10^{-9}$ to $< 10^{-8}$

Description des niveaux

Des choses analogues existent dans les autres domaines :

- Nucléaire (CEI 61513) : Niveau ou Level
- …

➔ Chaque niveau de criticité implique des contraintes particulières sur le système ou sur les parties du système concernées.

Contraintes

Plus le niveau de criticité du système est élevé, plus les contraintes sur le développement sont fortes.

Ces contraintes concernent notamment :

- Organisation
- Documentation
- Traçabilité
- Méthodes
- Vérification
- Tests

Organisation

Les différentes normes imposent d'adopter une organisation « spécifique ». Chaque norme a ses spécificités mais une chose est récurrente :

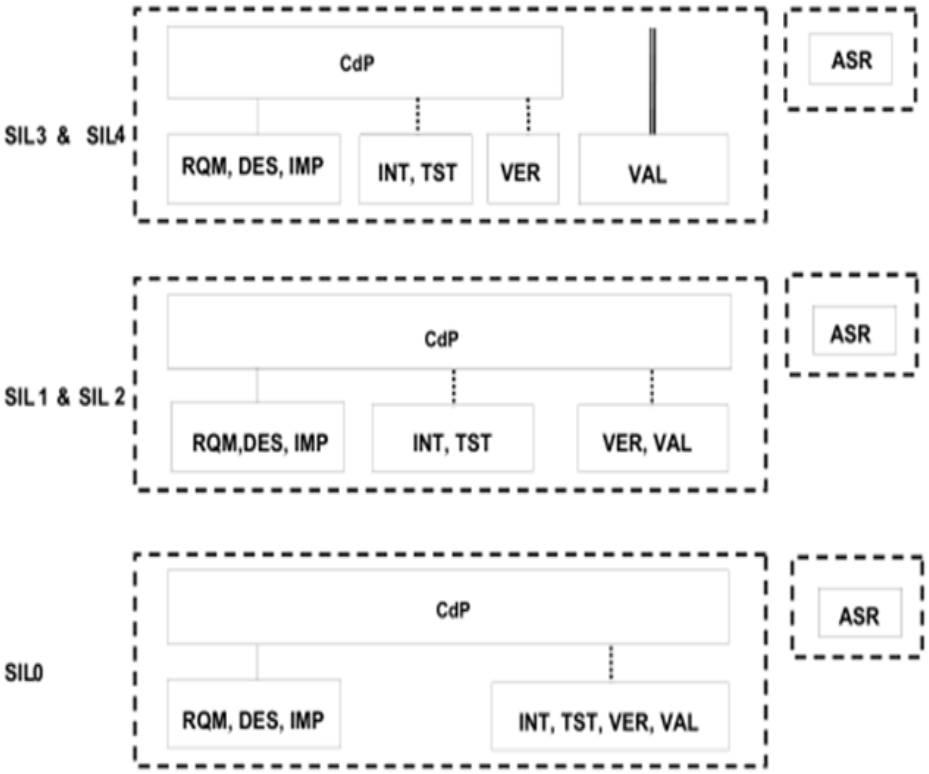
celui qui fait != celui qui vérifie

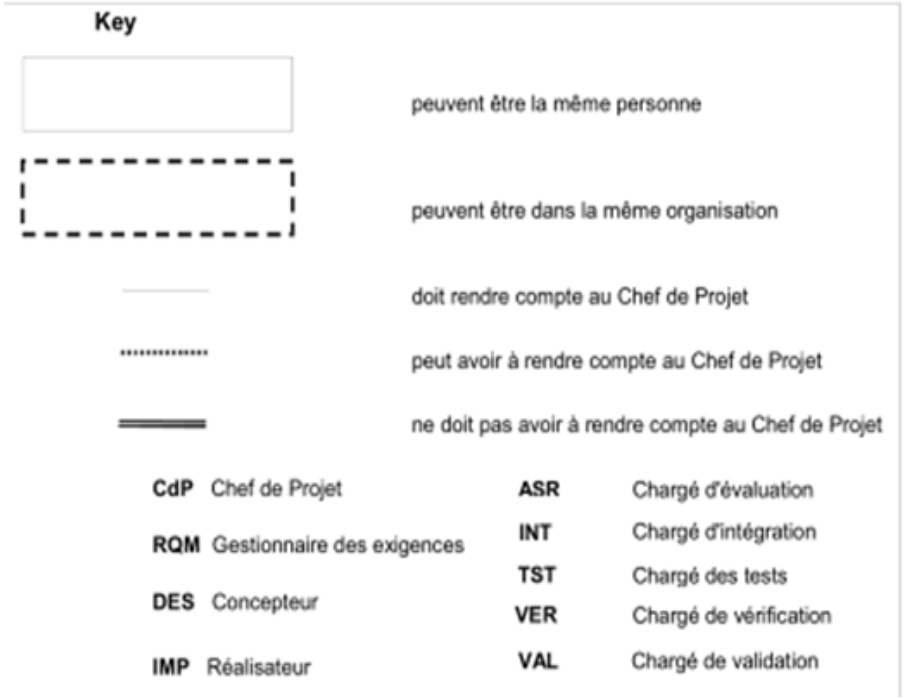
On trouve notamment :

- Des développeurs
- Des vérificateurs
- Des valideurs
- Un responsable qualité projet (RQP)

➔ Les normes imposent de démontrer l'adéquation entre le profil du personnel et la tâche qui leur est confiée

➢ Séparation des équipes





A chaque étape du développement ➔ phase de vérification associée

Spécification	➔	vérification de la spécification
Conception	➔	vérification de la conception
Codage	➔	vérification du codage
Intégration	➔	vérification de l'intégration

…

La vérification n'est pas « informelle » ➔ preuves de vérifications

Outils de développement

Modélisation de la spécification et/ou de la conception puis génération automatique de code :

→ Atelier B, SCADE, Control Build, …

Analyse statique de code / interprétation abstraite :

- Absence d'erreurs à l'exécution
- Absence d'erreurs arithmétiques
- …

→ Polyspace, Klocwork, Frama-C, …

Documentation

TOUT doit être documenté !

Encore plus que dans le développement de logiciels traditionnels, dans le cas de développement de logiciels critiques, la documentation est primordiale :

- Plan de développement
- Plan qualité
- Analyse des risques
- AMDEC
- Documentation des différentes phases du projet
- …

➔ Tout est tracé, tout est documenté, tout est justifié

Traçabilité

La traçabilité fait partie de la documentation !

2 types de traçabilité :

- Exigences amont → exigences aval
- Exigences aval → exigences amont

Exemple :

- Exigences de spécification → exigences de conception
- Exigences de conception → exigences de spécification

La traçabilité doit permettre de s'assurer que :

- rien n'a été oublié
- tout a une raison d'être

La traçabilité concerne aussi les étapes d'intégration et de validation.

Traçabilité

Un logiciel sécuritaire ne comporte pas forcément que des exigences de sécurité.

Les exigences de sécurité sont clairement identifiées et il est nécessaire de démontrer leur maîtrise et leur implémentation

➔ dossier de sécurité

Tout cela nécessite donc de mettre en place une architecture appropriée selon la criticité du logiciel

Pour faciliter le travail de gestion des exigences, de nombreux outils existent :

- IBM Rational DOORS
- Reqtify
- ReqFlow
- GenSpec
- ….

Ces outils ont chacun leur mode de fonctionnement et nécessite d'adopter une démarche appropriée

Langages de programmation utilisés :

De manière générale, les technologies utilisées dans le développement de logiciels critiques, sont souvent des « vieilles » technologies (C, ADA, …).

Règles de programmation :

- Limiter le risque d'erreurs de codage
- Facilité la maintenance du code
- Homogénéiser le code produit
- …

Programmation défensive :

La programmation défensive consiste à écrire le code en s'attendant au pire

Exemple :

- « codage » des booléens sur un octet → être capable de détecter un SEU (Single Event Upset)

Les méthodes dites « formelles » sont de plus en plus utilisées pour le développement de logiciel critiques

→ Modélisation de la spécification et/ou de la conception

→ Génération automatique de code (Atelier B, SCADE, Control Build, …)

→ Analyse statique de code / interprétation abstraite (Astrée, Polyspace, Frama-C, …)

- Absence d'erreurs à l'exécution (accès de tableaux en dehors des bornes, pointeurs NULL, …)
- Absence d'erreurs arithmétiques (division par 0, accès aux variables partagées, code mort, …)

Atouts des méthodes formelles

- Limitation des activités de tests de bas niveau (Tests Unitaires et d'Intégration Logiciel / Logiciel)
- Détection plus rapide et plus exhaustive des défauts
- Démonstration formelle que le logiciel répond aux besoins

Divers :

- Pas de changement concernant l'intégration Logiciel, Matériel et les Tests de Validation
- Qualification des outils
- Reconnaissance du formelle (DO-178, EN 50128)

Niveaux Vs Coût

Niveau de criticité élevé ➔ fortes contraintes

Fortes contraintes ➔ coût élevé

➔Le niveau de criticité n'est pas fixé « au hasard » !

Sous-estimer la criticité du logiciel ➔ danger

Sur-estimer la criticité du logiciel ➔ coûteux

Certification

Système critique ➔ certification

La certification est attribuée par un organisme indépendant qualifié et agrée dit « EOQA » (Expert ou Organisme Qualifié Indépendant)

Rôle de l'EOQA :

- Evaluation de la conformité du système aux règlements, normes et référentiels techniques en vigueur
- Evaluation de l'atteinte du niveau de sécurité requis pour le système dans son ensemble

Certification

Système critique ➔ certification

La certification est attribuée par un organisme indépendant qualifié et agrée dit « EOQA » (Expert ou Organisme Qualifié Indépendant)

Rôle de l'EOQA :

- Evaluation de la conformité du système aux règlements, normes et référentiels techniques en vigueur
- Evaluation de l'atteinte du niveau de sécurité requis pour le système dans son ensemble

Certification

- Pour quoi faire ?
  - ✓ Harmoniser les pratiques.
  - ✓ Assurer un niveau de qualité supérieur.
  - ✓ Portée nationale, européenne ou internationale.
  - ✓ Prérequis pour répondre à un appel d'offre.

- Démonstration du respect des normes, du process qualité, des développements et de la sureté de fonctionnement
  - ✓ Démontrer que le système est apte à remplir une ou plusieurs fonctions requises dans des conditions données.
  - ✓ 4 composantes : fiabilité, disponibilité, maintenabilité et sécurité (FDMS ou RAMS).

- Mise en service
  - ✓ OK de l'organisme d'évaluation et de certification.
  - ✓ Niveau de sûreté assuré.

Focus sur les tests

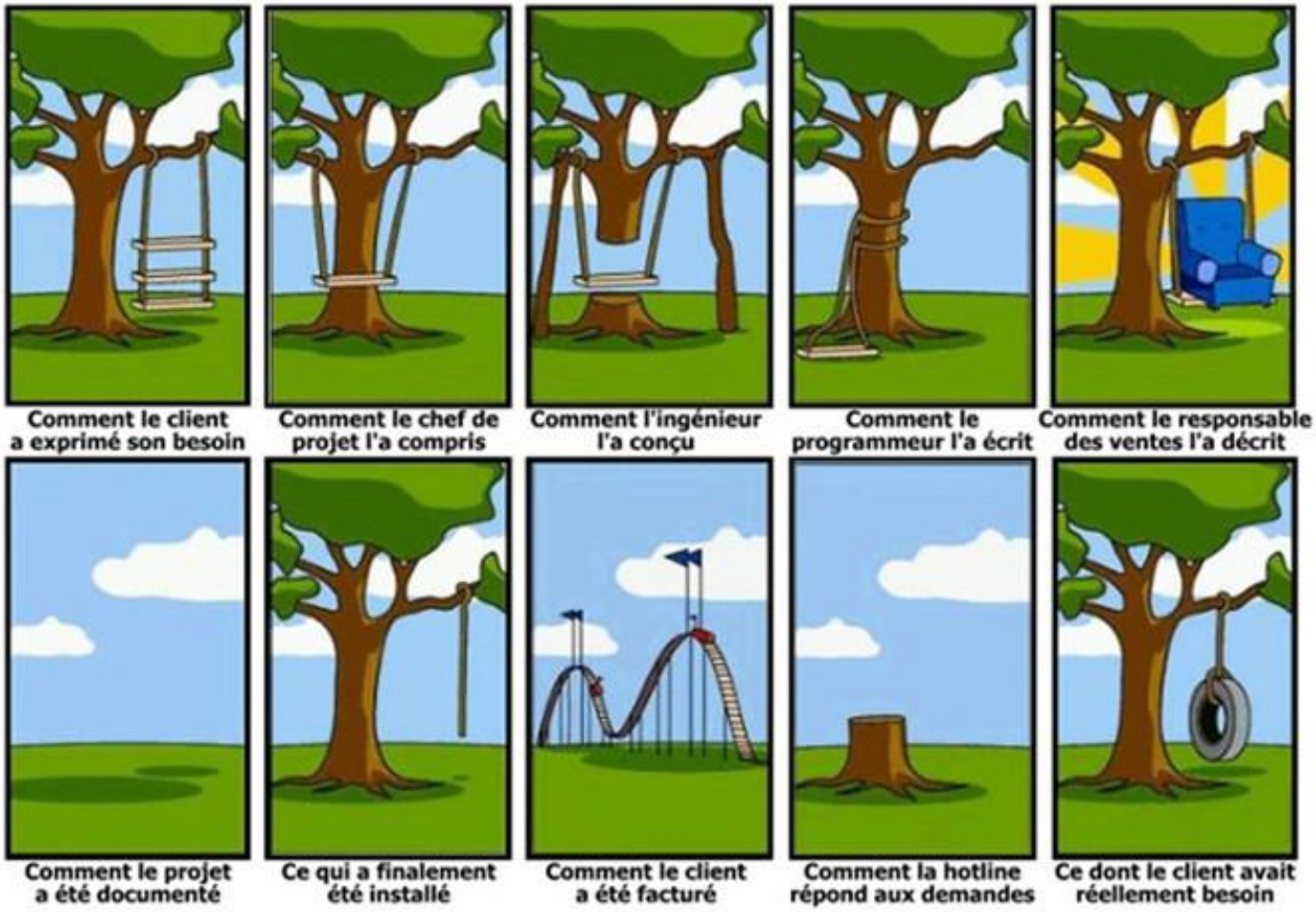
Pourquoi des tests ?

- Détecter des bugs

- Vérifier le bon fonctionnement du logiciel (conformité aux spécifications)

- Démontrer le respect des process (normes et qualité)
  - ✓ Couverture
  - ✓ Détection code mort
  - ✓ Documentation
  - ✓ Gestion de configuration
  - ✓ Etc.

La compréhension du besoin



Les 7 principes de test

1 – Les tests montrent la présence de défauts…

- ✓ …mais ne peuvent pas en prouver l'absence.

2 – Les tests exhaustifs sont impossibles

- ✓ Tout tester n'est pas faisable sauf pour des cas triviaux. Plutôt que des tests exhaustifs, utiliser l'analyse des risques et des priorités.

3 – Tester tôt

- ✓ Corriger une erreur dans les spécifications coûte moins que de la corriger après son développement.

4 – Regroupement des défauts

- ✓ Un petit nombre de modules contiennent généralement la majorité des défauts détectés. L'effort de test est à diriger en ce sens.

5 – Paradoxe du pesticide

- ✓ Si les mêmes tests sont répétés de nombreuses fois, il arrivera que le même ensemble de cas de tests ne trouvera plus de nouveaux défauts.
- ✓ Pour prévenir ce “paradoxe du pesticide”, les cas de tests doivent être régulièrement revus et révisés, et de nouveaux tests, différents, doivent être écrits pour couvrir d’autres chemins dans le logiciel ou le système

6 – Les tests dépendent du contexte

- ✓ Le contexte métier, le moment de l'année, le nombre et les caractéristiques des utilisateurs, le type de logiciel (logiciels de sécurité critique ou site web) sont autant d'éléments précieux qui permettent d'identifier les types de tests à prévoir.

7 – Illusion de l’absence d’erreurs

- ✓ L'exigence de qualité nécessite de comprendre en profondeur le besoin des utilisateurs finaux, c'est pourquoi le testeur doit endosser un rôle proactif d'interface entre équipe technique et équipe métier. Sans cela, on aboutira à un produit bien fait, qui ne sera pourtant pas le bon produit.

Les bons réflexes

- Indépendance
  - ✓ Un programmeur ne doit pas tester ses propres programmes !
  - ✓ Il peut / doit faire des tests sommaires pour vérifier son développement…
- Paranoïa
  - ✓ Ne pas faire de tests avec l'hypothèse qu'il n'y a pas d'erreur (code trivial, déjà vu, ...) ⇒ bug assuré !
- Prédiction
  - ✓ La définition des sorties/résultats attendus doit être effectuée avant l'exécution des tests. C'est un produit de la spécification.

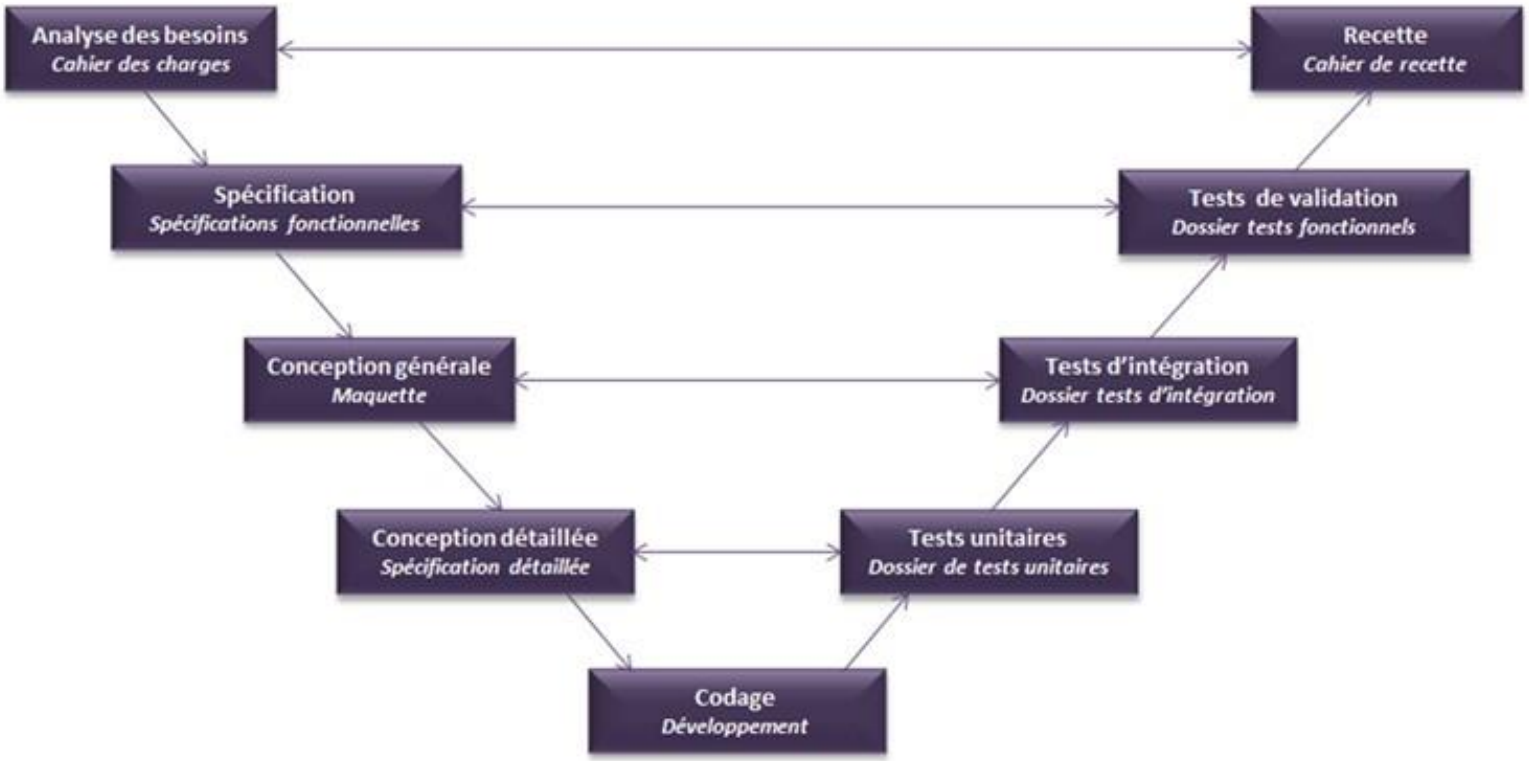
Les bons réflexes

- Vérification
  - ✓ Il faut inspecter :
    - La pertinence de chaque test
    - Les données d'entrées du test ainsi que les résultats obtenus

- Robustesse
  - ✓ Les jeux de tests doivent être écrits avec des jeux valides, mais aussi invalides ou incohérents : cas de défaut sur un capteur par exemple

Les niveaux de tests

- Cycle en V

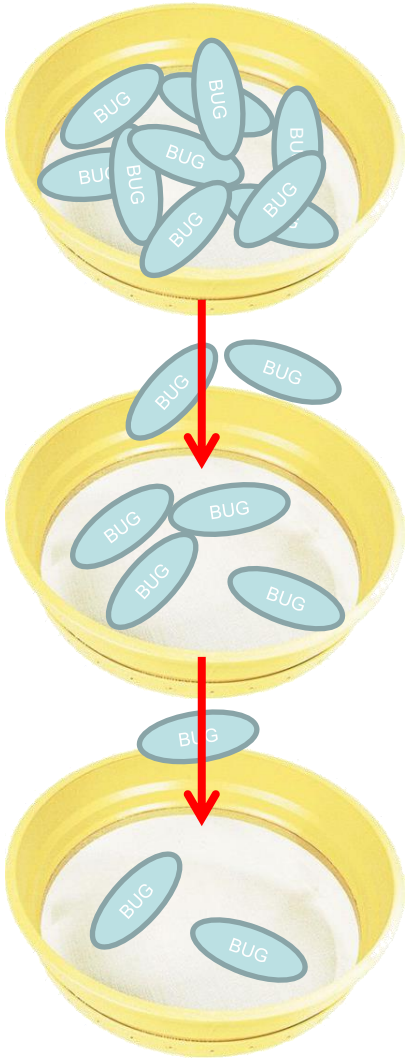


Les niveaux de tests

- Tests de recette :
  - ✓ Tests de réception du logiciel chez le client final
- Tests de validation (logiciel ou système) :
  - ✓ Tests d'acception du logiciel avant livraison
- Tests d'intégration (logiciel ou système) :
  - ✓ Tests de l'intégration des différents composants (avec ou sans hardware)
- Tests unitaires:
  - ✓ Tests élémentaires des composants logiciels (une fonction, un module, ...)
- Tests de non-régression
  - ✓ Tests de vérification que les fonctions non « touchées » n'ont pas régressées fonctionnellement
- "Smoke-tests" ou « tests développeur »:
  - ✓ Jeu réduit de tests pour valider un build avant de passer à la validation

Les niveaux de tests

- Tests unitaires
- Tests d'intégration
- Tests de validation



Les types de test

- Fonctionnels (boite noire)
  - ✓ Objectif : analyse du comportement.
    - Vérifie que le logiciel réalise ce qui est décrit dans les documents de spécification, conception, etc.
    - Conformité, comportement nominal, aux limites, robustesse.

- Non-Fonctionnels (boite noire)
  - ✓ Objectif : évaluer “comment” le système fonctionne
    - Les tests non-fonctionnels incluent, mais pas uniquement, les tests de performances, tests de charge, tests de stress, tests d’utilisabilité, tests de maintenabilité, tests de fiabilité et les tests de portabilité.

- Structurels (boite blanche)
  - ✓ Objectif: détecter les fautes d’implémentation
    - Les tests structurels (boîte blanche) peuvent être effectués à tous les niveaux de tests. Ces tests sont réalisés généralement après les techniques basées sur les spécifications, pour aider à mesurer l’ampleur des tests via l’évaluation de la couverture d’un type de structure.

Techniques statiques

- ✓ Les techniques de tests statiques reposent sur l'examen manuel (revues) ou l'analyse(analyse statique) du code ou de la documentation du projet sans exécution du code.
- ✓ Possible outillage pour l'analyse de règles de codage ou de métriques par exemple.

Techniques dynamiques

- ✓ Tests nécessitant l'exécution du logiciel.
- ✓ Tester et déboguer sont différents :
  - Les tests dynamiques peuvent montrer des défaillances causées par des défauts.
  - Déboguer est l'activité de développement qui trouve, analyse et supprime les causes de la défaillance. Les tests de confirmation ultérieurs effectués par un testeur assurent que la correction résout effectivement la défaillance.
- ✓ La responsabilité de chaque activité est différente : les testeurs testent, les développeurs déboguent.

Gestion des risques

- Définir sa stratégie de test en fonction des risques associés
  - ✓ Définir les risques liés à l'application
  - ✓ Les quantifier
  - ✓ Les classer
  - ✓ Définir sa stratégie de test ainsi que le suivi du projet

- Exemple basique : tester un « ET logique » dans la conception

Exigence 1 (test du **AND**) :	IF intput1 **AND** input2 **AND** input3
	THEN
	output1 = VRAI
	ELSE
	output1 = FAUX
	ENDIF

qui peut s'écrire également :

Exigence 1 (test du **AND**) :	output1 = intput1 **AND** input2 **AND** input3

➢ Rédiger des cas de test :

✓ Stratégie 1 : couverture des branches

Test exigence 1 : AND				
	input1	input2	input3	output1
Cas de test 1	V	V	V	V
Cas de test 2	F	V	V	F

✓ Stratégie 2 : classe d'équivalence sur les entrées

Test exigence 1 : AND				
	input1	input2	input3	output1
Cas de test 1	V	V	V	V
Cas de test 2	F	V	V	F
Cas de test 3	V	F	V	F
Cas de test 4	V	V	F	F

➢ Rédiger un scénario de test :

✓ Stratégie 1 : couverture des branches

	Test exigence 1 : AND			
	input1	input2	input3	output1
Etape 1 : Cas de test 1	V	V	V	V
Etape 2 : Cas de test 2	F	V	V	F

=> scénario en 2 étapes minimum : 2 combinatoires ne sont pas testées

✓ Stratégie 2 : classe d'équivalence sur les entrées

	Test exigence 1 : AND			
	input1	input2	input3	output1
Etape 1 : Cas de test 1	V	V	V	V
Etape 2 : Cas de test 2	F	V	V	F
Etape 3	V	V	V	V
Etape 4 : Cas de test 3	V	F	V	F
Etape 5	V	V	V	V
Etape 6 : Cas de test 4	V	V	F	F

=> scénario en 6 étapes minimum : tout est testé

➢ Comparaison des 2 méthodes :

- ✓ Stratégie 1 : couverture des branches
  - Avantages :
    - facile à définir
    - peu de tests à écrire
  - Inconvénients :
    - Peu de combinatoires testées
    - Faible confiance dans le logiciel

- ✓ Stratégie 2 : classe d'équivalence sur les entrées
  - Avantages :
    - Bonne confiance dans le logiciel
    - Détecte plus de bugs
  - Inconvénients :
    - Plus de cas de tests à décrire et rédiger
    - Plus coûteux

➢ Conclusion :

- ✓ La stratégie est définie en fonction de plusieurs critères
  - L'analyse des risques et le niveau de criticité du logiciel
  - Les normes dont dépendent la réalisation du logiciel

- ✓ La stratégie de test pour un logiciel sécuritaire ne dépend pas :
  - Du nombre de personnes disponibles !
  - Du délai pour sortir le logiciel !
  - Du budget disponible !

Critère d'arrêt des tests

- L'objectif est de définir quand arrêter le test
  - ✓ C'est-à-dire quand on estime qu'on a suffisamment confiance

- Exemples de critères de sortie :
  - ✓ Mesure d'exhaustivité (couverture de code, de fonctionnalités ou de risques)
  - ✓ Estimation de la densité des anomalies ou des mesures de fiabilité
  - ✓ Les risque résiduels (anomalies non corrigées, manque de couverture, etc.)

➢ Exemple de 2 approches :

1 – **Métriques** : estimation de l'effort de test basée sur des mesures issues de projets antérieurs ou similaires ou basée sur des valeurs typiques.

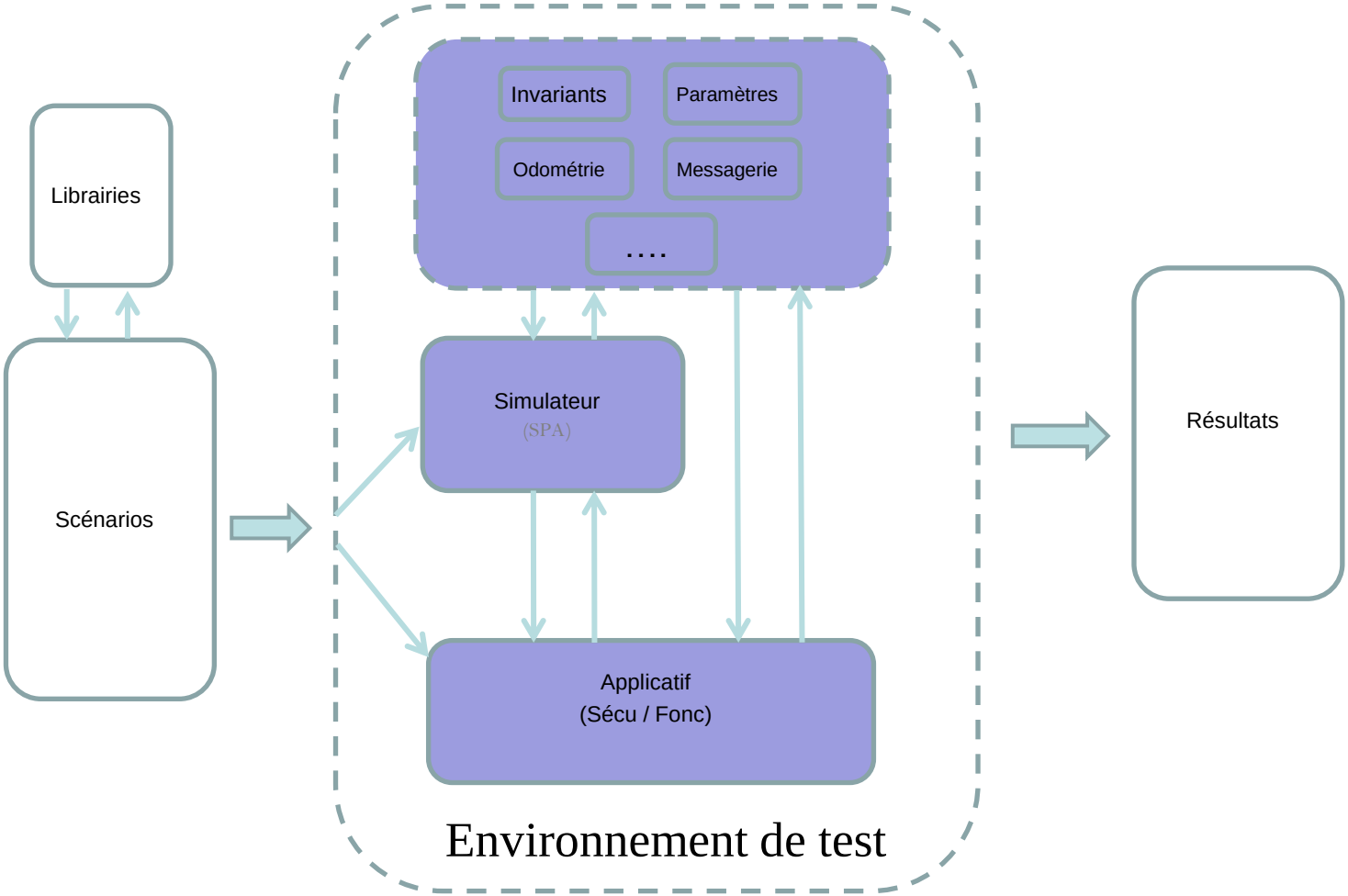
2 - **L'approche experte** : estimation des tâches par le détenteur de ces tâches ou par des experts.

✓ Une fois que l'effort de test a été estimé, il est possible :
- D'identifier les ressources nécessaires.
- D'établir un échéancier.

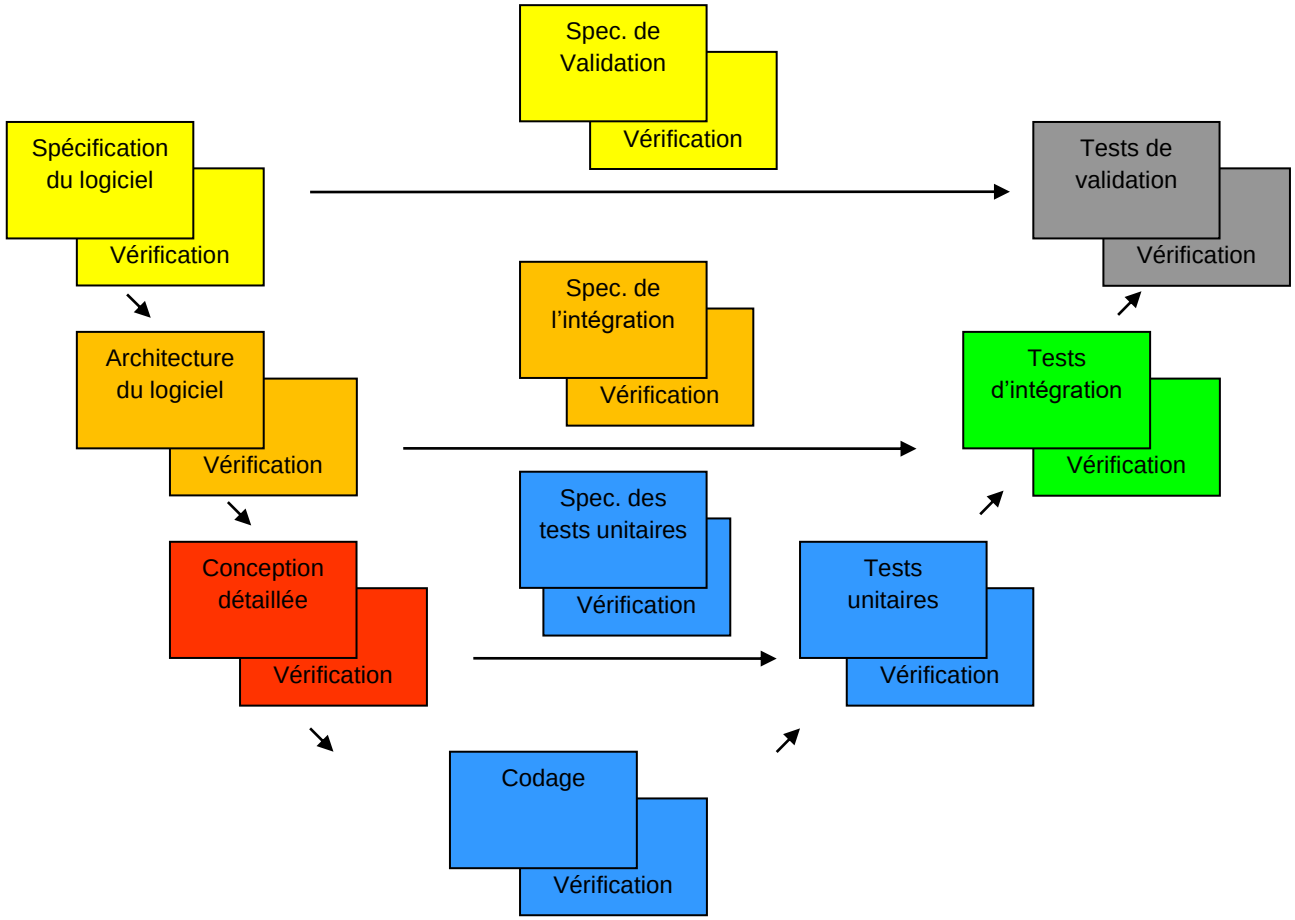
✓ Les critères d'entrées et de sortie des tests ainsi que la stratégie de test devront être définis dans cette étape.

- Démarche de test

Mise en place des tests



➢ Rappel cycle en V et documents associés



Gestion particulières des bugs

- Planification de la prise en compte des bugs détectés
  - ✓ Chaque anomalie est tracée
  - ✓ Analyse et études d'impacts de l'anomalie
- Justification (restriction pour la mise en service ?)
  - ✓ Exemple sur un train : Problème sur la fonction de retournement du train sur une aiguille
- Démonstration de non opposition à la sureté de fonctionnement
  - ✓ Exemple : anomalie sur une fonction de confort

Cas de l'interopérabilité

- L'interopérabilité
  - ✓ C'est quoi ?
  - ✓ Comment cela fonctionne ?
  - ✓ Un exemple sur Octys L5 métro Parisien
- Des tests communs entre les industriels

Bilan

- Partie VnV à ne pas négliger
- Vision plus globale du fonctionnement du système
- Aussi voire plus coûteuse que le développement
- Démonstration de bon fonctionnement et de sûreté
- Cyber-sécurité dans tous les esprits

Introduction à l'ingénierie des systèmes

Un système ?

*« Un système est un **ensemble d'éléments** interagissant entre eux selon certains principes ou règles. Un système est déterminé par :*

- *la nature de ses éléments constitutifs,*
- *les interactions entre ces derniers,*
- *sa frontière, c'est-à-dire le critère d'appartenance au système (déterminant si une entité appartient au système ou fait au contraire partie de son environnement),*
- *ses interactions avec son environnement. »*

Source Wikipédia

Ingénierie des systèmes

L'Ingénierie Système (IS) s'appuie sur une démarche et des moyens **multi-discipline** pour réaliser avec succès des systèmes. Elle se focalise sur la **définition au plus tôt** dans le cycle développement, des **besoins client** et des **fonctionnalités** demandées, sur la documentation des **exigences**, et ensuite sur le traitement synthétique de la **conception** et de la **validation** du système en considérant l'**ensemble du problème** (i.e. opérations, coûts & délais, performance, formation & assistance, test, fabrication, retrait de service).

L'IS intègre toutes les disciplines et spécialités dans un travail d'équipe en utilisant un **processus de développement structuré** qui intervient des concepts de production jusqu'aux opérations. L'IS prend en compte les besoins techniques et de business de tous les clients dans le but de **fournir un produit de qualité** respectant les besoins des utilisateurs.

(INCOSE- International Council of Systems Engineering)

Ingénierie des systèmes

L'ingénierie des systèmes ou ingénierie système est une **approche scientifique interdisciplinaire**, dont le but est de formaliser et d'appréhender la conception de **systèmes complexes** avec succès.

L'ingénierie des systèmes a pour objectif de maîtriser et de contrôler la conception de systèmes dont la complexité ne permet pas le pilotage simple. Par système, on entend un ensemble d'éléments humains ou matériels en interdépendance les uns les autres et qui inter-opèrent à l'intérieur de frontières ouvertes ou non sur l'environnement. Les éléments matériels sont composés de sous-ensembles de technologies variées : mécanique, électrique, électronique, matériels informatiques, logiciels, réseaux de communication, etc.

(Wikipédia)

Un système complexe ?

Définition :

- Ensemble des éléments en interaction dynamique organisés en vue d'une mission, de services, de fonctions,
- La complexité est due à l'ensemble à prendre en compte et à l'évolution de l'environnement.

Caractéristiques :

- Nombre d'éléments,
- Nature des interactions entre ces éléments,
- Nombre et variété des liaisons qui relient ces éléments entre eux.

Pourquoi l'IS ?

- Système de qualité (fonctionnalités conformes aux besoins du client, réutilisabilité, maintenabilité, …)
- Réduction des risques liés aux coûts
- Réduction des risques liés aux délais

→ Démarche QCD (Qualité/Coût/Délai) (entre autres) et recherche du meilleur compromis

Quels autres bénéfices ?

- Documentation système de meilleure qualité
- Amélioration de la participation des parties prenantes
- Fonctionnalités système qui respectent les attentes des parties prenantes
- Réduction du temps de cycle du projet
- Limitation de la réingénierie et des coûts pour les évolutions
- Réduction des risques liés aux délais
- Meilleure possibilité de réutilisation
- Meilleure prédiction des projets

Mais encore ?

L'ingénierie des systèmes doit permettre de trouver la meilleure solution et ainsi optimiser la performance du projet concerné en terme de :

- Durée,
- Coût (étude),
- Performance du produit,
- Coût (production).

Les dilemnes

Réduire les coûts à risques constants

→ Réduire les performances du produit

Réduire le risque à coûts constants

→ Réduire les performances du produit

Réduire les coûts à performances constantes

→ Augmenter les risques

Réduire les risques à performances constantes

→ Augmenter les coûts

Les raisons d'échec

Facteurs d'échec	Causes racines	Facteurs de réussite
37%	Besoins et exigences	40%
9%	Projet, ressources	23%
8%	Gestion des données techniques	14%
11%	Technique, technologique	9%

Données Standish Group

Les raisons d'échec

Manque de rigueur :

- Recherche de l'innovation technologique sans mesurer les impacts au plan industriel et utilisation opérationnelle,
- Absence de vision globale sur le comportement et les défaillances,
- Peu ou pas d'analyse fonctionnelle,
- Faible maîtrise des risques,
- Cohérence système insuffisamment maîtrisée (volume important et grande diversité d'informations et relation entre ces informations).

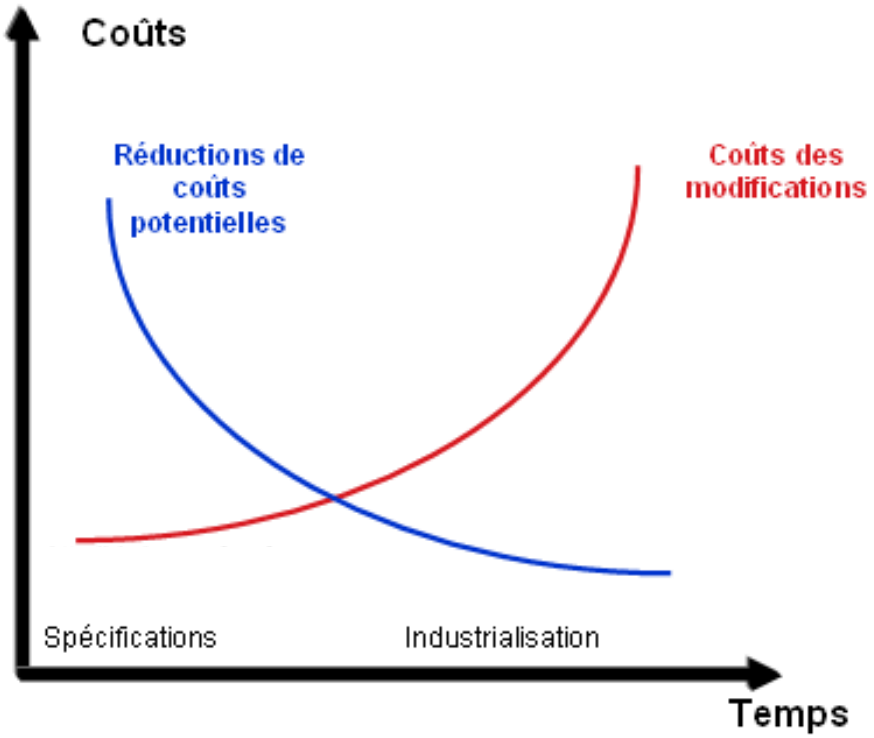
Les erreurs ne pardonnent pas !!!

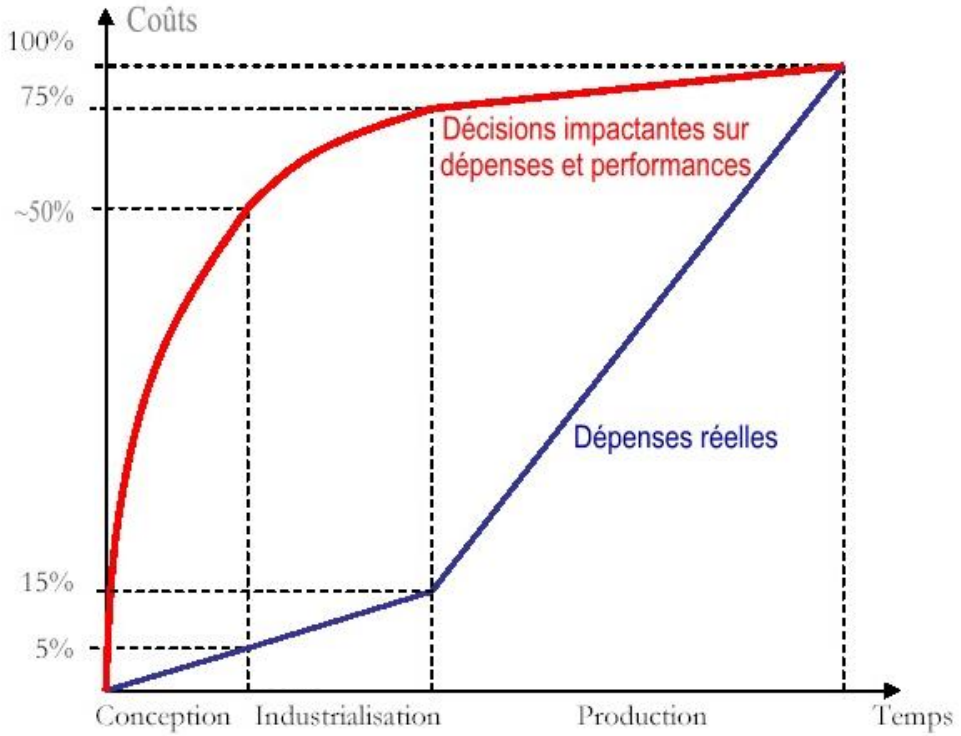
Coût relatif d'une correction (selon l'avancement du projet) :

- 1 si correction en spécification,
- 2 si correction en conception préliminaire,
- 5 si correction en conception détaillée/développement,
- 10 si correction en phase de test,
- 30 et + si correction en phase opérationnelle.

Données INCOSE (International Council on Systems Engineering)

Quelques chiffres





→ Importance des décisions initiales et donc nécessité de maîtriser les phases amonts (spécification et conception).

Comment ?

- Démarche orientée processus,
- Marier Approche analytique / Approche systémique,
- Prise en compte de TOUTES les exigences et contraintes,
- Prendre en compte les différents points de vue,
- Décomposer le système,
- Anticiper au maximum les phases avales,
- Réutiliser,
- Structure le développement en projets
- Traquer les non conformités,
- …

Processus - Définition

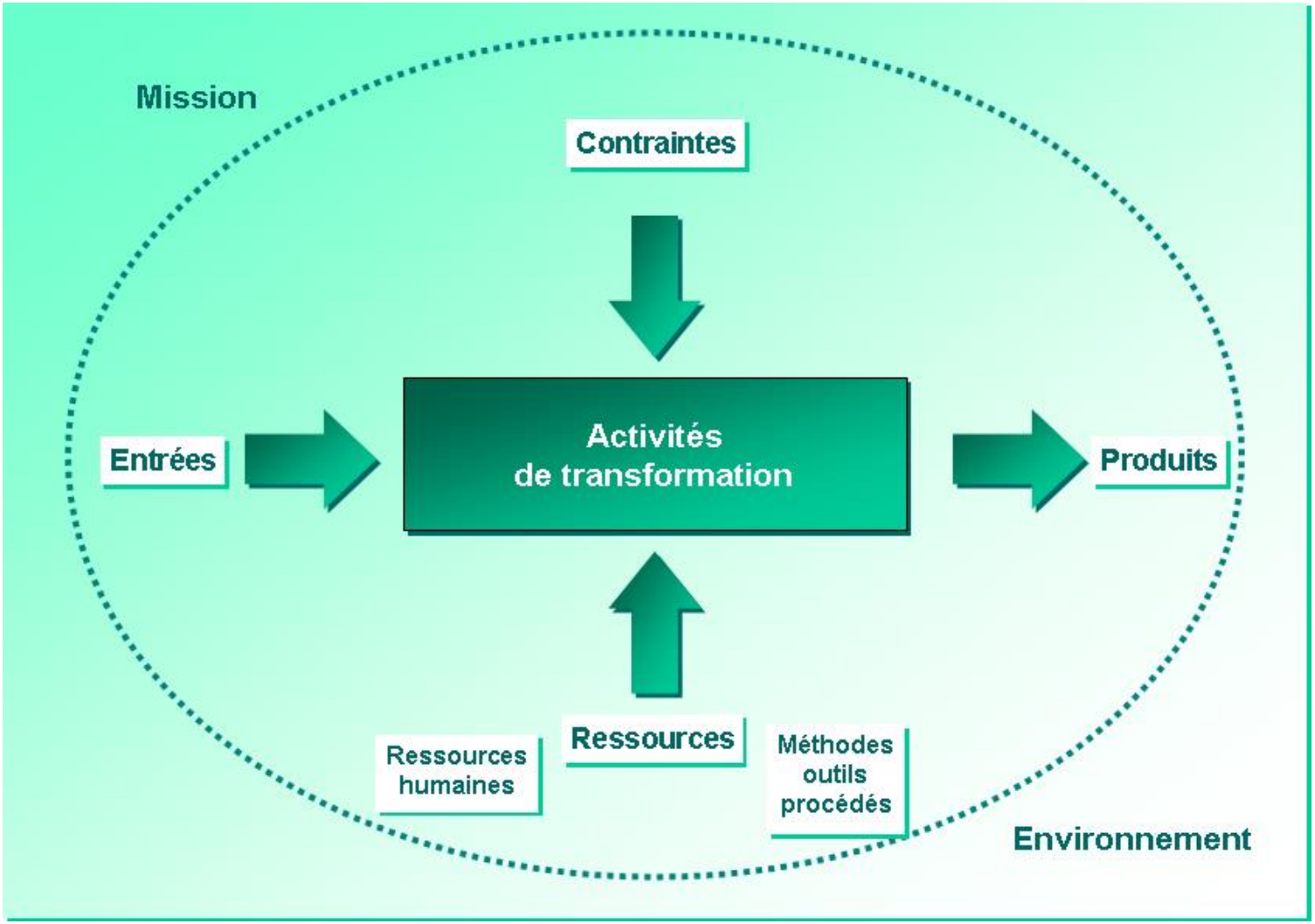
Norme ISO 9001 :

Processus = Ensemble d'activités corrélées ou interactives qui transforment des éléments d'entrée en éléments de sortie.

Les moyens mis à disposition de ce processus peuvent inclure le personnel, les finances, les installations, les équipements, les techniques et méthodes.

Il est nécessaire d'intégrer ces processus à leur environnement.

Processus - Définition



Processus – Intérêt

- Planifier :
  - Activités à réaliser,
  - Enchaînement de ces activités (cycle de vie, entrées/sorties),
  - Surveillance de l'avancement (revues/jalons, indicateurs).

- Capitaliser le savoir faire (processus reproductibles) :
  - Référentiel société,
  - Outillage.

→ Nécessité d'ajuster le processus avant et pendant le projet

Approche analytique

Découpage de la complexité du système en éléments distincts. Le découpage est fait de manière à ce que chaque élément soit traité par une discipline appropriée.

→ Problème : Fragmentation et cloisonnement du savoir

→ Challenge : Amener les disciplines à concourir au même objectif final

Approche systémique

Décomposition du système en éléments interdépendants qui évoluent dans le temps. Le système est organisé en niveaux (poupées russes).

→ Problème : Besoin d'activités dédiées pour maîtriser le système

→ Challenge : Optimiser les activités de maîtrise de l'ensemble

Exigences et contraintes

- **Besoins clients**
- Contraintes techniques (normes, standards, état de l'art…)
- Contrainte d'exploitation (coût d'exploitation et de maintenance, …)
- Contraintes industrielles (sous-traitance, partenariat, réutilisation, …)
- Contraintes contractuelles
- Contraintes humaines
- Contraintes de l'environnement (législation, politique, …)
- Contraintes programmatiques (coût, délai)

Traquer les non-conformités

Eliminer tout risque qu'un client refuse d'accepter le produit :

1. Quel engagement du fournisseur par rapport a son client ?
2. Identification des non-conformités par rapport à cet engagement
3. Mise en œuvre, au plus tôt, de tout ce qui est possible pour éliminer chaque non-conformité

Synthèse (1)

Ingénierie des systèmes → ingénieur système ou « systémier »

Cet ingénieur système doit :

- Considérer le système du point de vue des différentes parties prenantes (analyse des besoins, participation de ces parties prenantes, …)
- Démarrer en définissant les sorties du système
- Prendre en compte les risques le plus tôt possible (coûts plus faibles

Synthèse (2)

Cet ingénieur système doit :

- Faire les choix technologiques le plus tard possible (définir ce qui est à faire avant le « comment »)
- Détailler les interfaces du système pendant la phase de conception (description précise des interfaces entre les éléments et gestion des évolutions)
- Comprendre et prendre en compte l'environnement du système