Software testing

Emmanuel Grolleau

Observatoire de Paris – LESIA – Scientific I.T. department

Master 2 « Outils et Systèmes de l'Astronomie et de l'Espace »

Summary

Introduction

- Why test ?
- Software quality
- Systems development life cycle (reminder)
- Agile software development

Software testing

- Introduction
- Types of Software Testing
- Test Strategy, Test Plan
- Anomaly
- Coverage matrix (Matrice de couverture)
- Continuous integration and test frameworks

Summary

Introduction

- **Why test ?**
- Software quality
- Systems development life cycle (reminder)
- Agile software development

Software testing

- Introduction
- Types of Software Testing
- Test Strategy, Test Plan
- Anomaly
- Coverage matrix (Matrice de couverture)
- Continuous integration and test frameworks

Test activities

- Often overlooked
- View as less rewarding
- The cost of test is often more than 50% of the total cost of a software
- Very few courses at university

Why test?

- **Ariane 5 Bug (1996)**
  - 370 million dollars
- **French social security bug (2009)**
  - between 900 million and 2,5 billion euros
- **Mars Climate Orbiter (1999)**
  - 300 million dollars.
- A 2013 study from Cambridge University evaluated to 312 billion dollar a year the software bug cost.

Summary

Introduction

- Why test ?
- **Software quality**
- Systems development life cycle (reminder)
- Agile software development

Software testing

- Introduction
- Types of Software Testing
- Test Strategy, Test Plan
- Anomaly
- Coverage matrix (Matrice de couverture)
- Continuous integration and test frameworks

Software quality concepts
ISO 9126 standard

- How to define the quality of a software?
  - We have to base our assessment on indicators

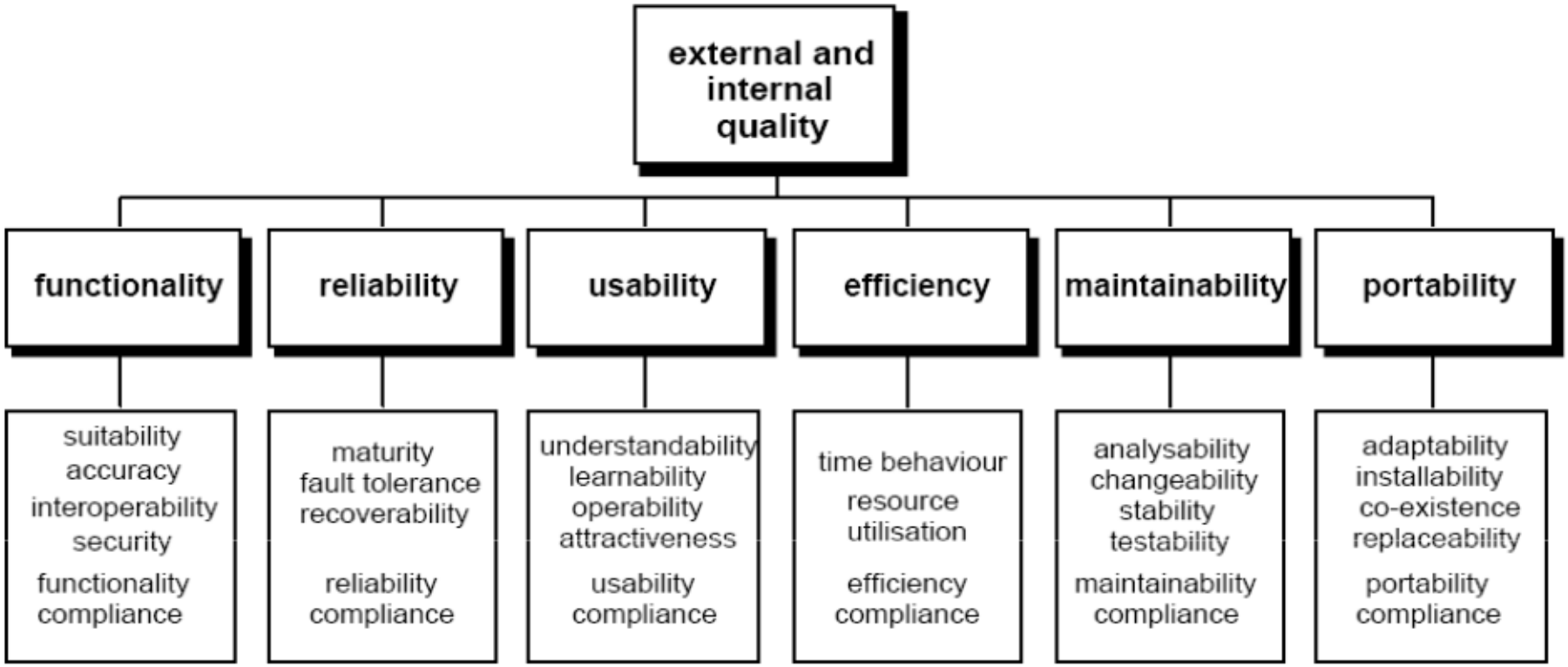
- The standard **ISO 9126** (Software quality) defined 6 groups of software **quality characteristics** (facteurs de qualité) : **FURPSE**

  - F (Functionality): meet the requirements
  - U (Usability): the ease of use
  - R (Reliability): to maintain service under defined conditions for defined periods of time
  - P (Performance, efficiency): system resources used when providing the required functionality
  - S (System Maintainability): ability to identify and fix a fault within a software component
  - E (Portability (Evolutivity)): how well the software can adapt to changes in its environment

Software quality concepts
ISO 9126 standard

- **Quality Characteristics (Facteurs Qualités)** :
  - this is the user view, the external view
- **Quality Sub-Characteristics (Critères Qualités)** :
  - this is the developer view, the internal view

Software quality concepts
ISO 9126 standard



Sub-characteristics ISO 9126 (1/3)

- F : **Functionality**
  - Suitability / match the requirements
  - Accuracy / correctness of the functions
  - Interoperability / interact with other components or systems
  - Security / manage access to the software

- U : **Usability**
  - Understandability / ease of which the systems functions can be understood
  - Learnability / learning effort for different users
  - Operability / ability of the software to be easily operated by a given user

Sub-characteristics ISO 9126 (2/3)

- R : **Reliability**
  - Maturity / frequency of failure of the software
  - Fault tolerance / ability of software to withstand from failure
  - Recoverability / bring back a failed system to full operation

- P : **Efficiency**
  - Time behaviour / response times
  - Resource behaviour / resources used, i.e. memory, cpu, disk and network usage

Sub-characteristics ISO 9126 (3/3)

- S : **Maintainability**
  - Analysability / ability to identify the root cause of a failure
  - Changeability / amount of effort to change a system
  - Stability / sensitivity to change of a given system
  - Testability / effort needed to verify (test) a system change

- E : **Portability**
  - Adaptability / ability to change to new specifications
  - Installability / the effort required to install the software
  - Conformance / conformity of interfaces (example : API)
  - Replaceability / how easy is it to exchange a given software component

Requirements

- Functional requirement (Exigences fonctionnelles)
  - **Business rules** (règles métiers)

- Non-functional requirement (Exigences non fonctionnelles)
  - **Usability**
  - **Reliability**
  - **Efficiency**
  - **Maintainability**
  - **Security**

- Constraints
  - Operating System
  - Programming language

How the customer explained it

How the team designed it

What the customer really needed

Summary

Introduction

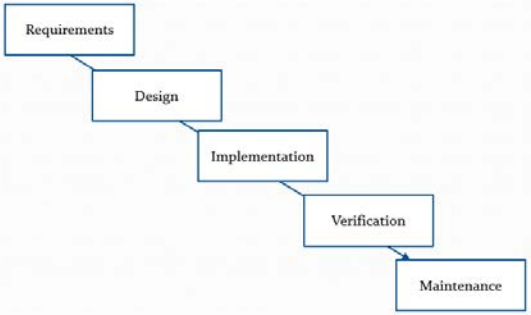
- Why test ?
- Software quality
- **Systems development life cycle (reminder)**
- Agile software development

Software testing

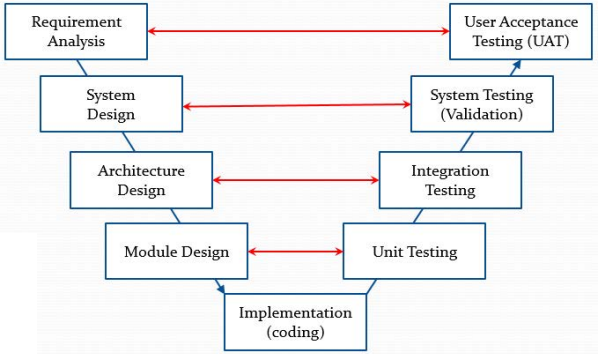
- Introduction
- Types of Software Testing
- Test Strategy, Test Plan
- Anomaly
- Coverage matrix (Matrice de couverture)
- Continuous integration and test frameworks

Systems Development Life Cycle

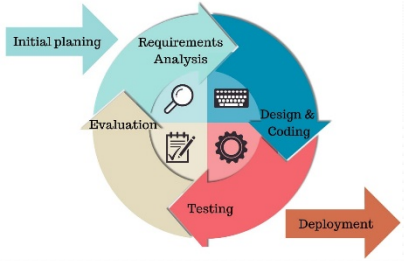
- **Waterfall model**..



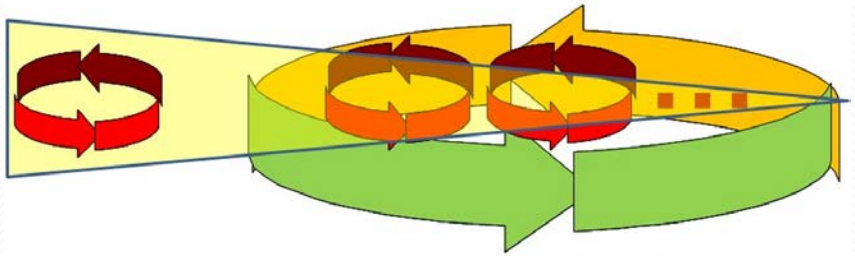
- V-model .....



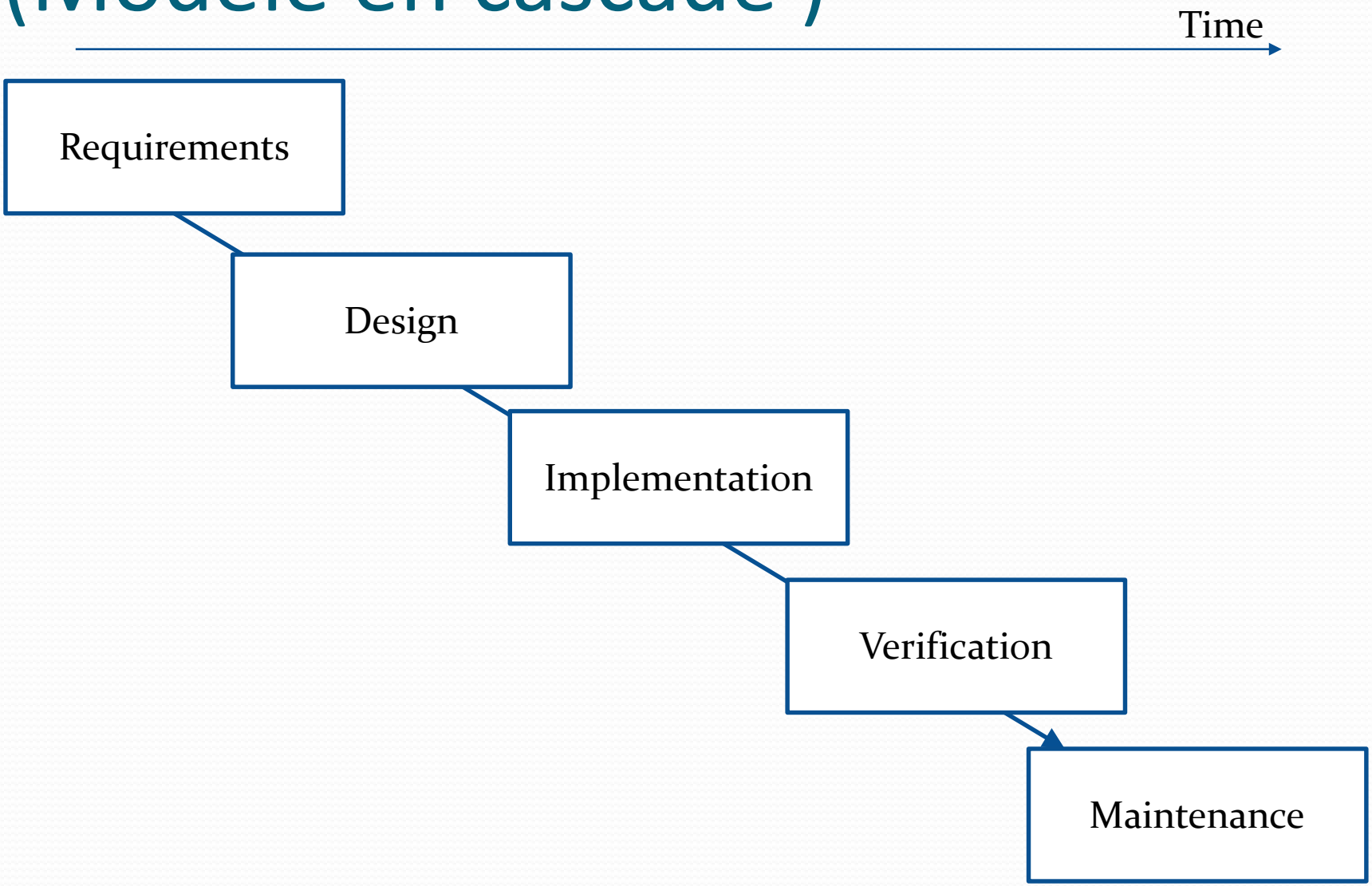
- Iterative development .....



- Agile software development.....

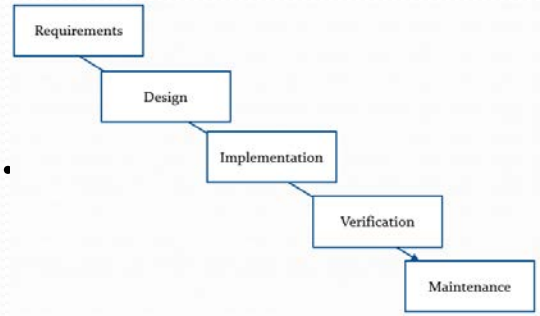


Waterfall model (Modèle en cascade)

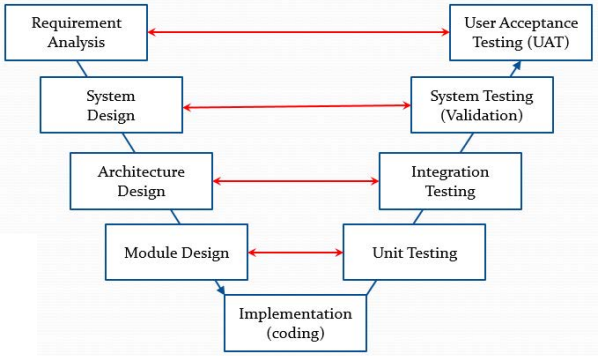


Systems Development Life Cycle

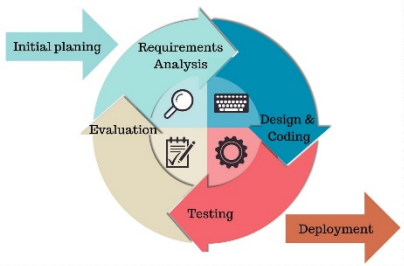
- Waterfall model.....



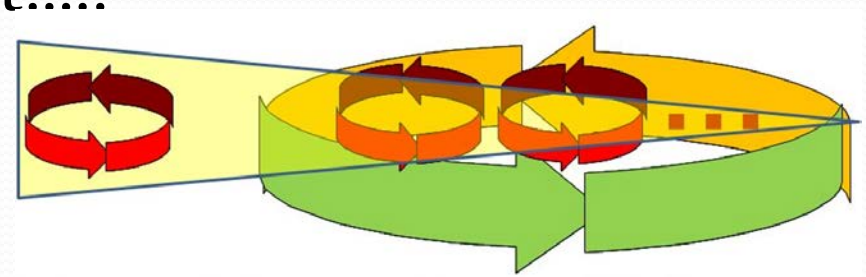
- **V-model** .....



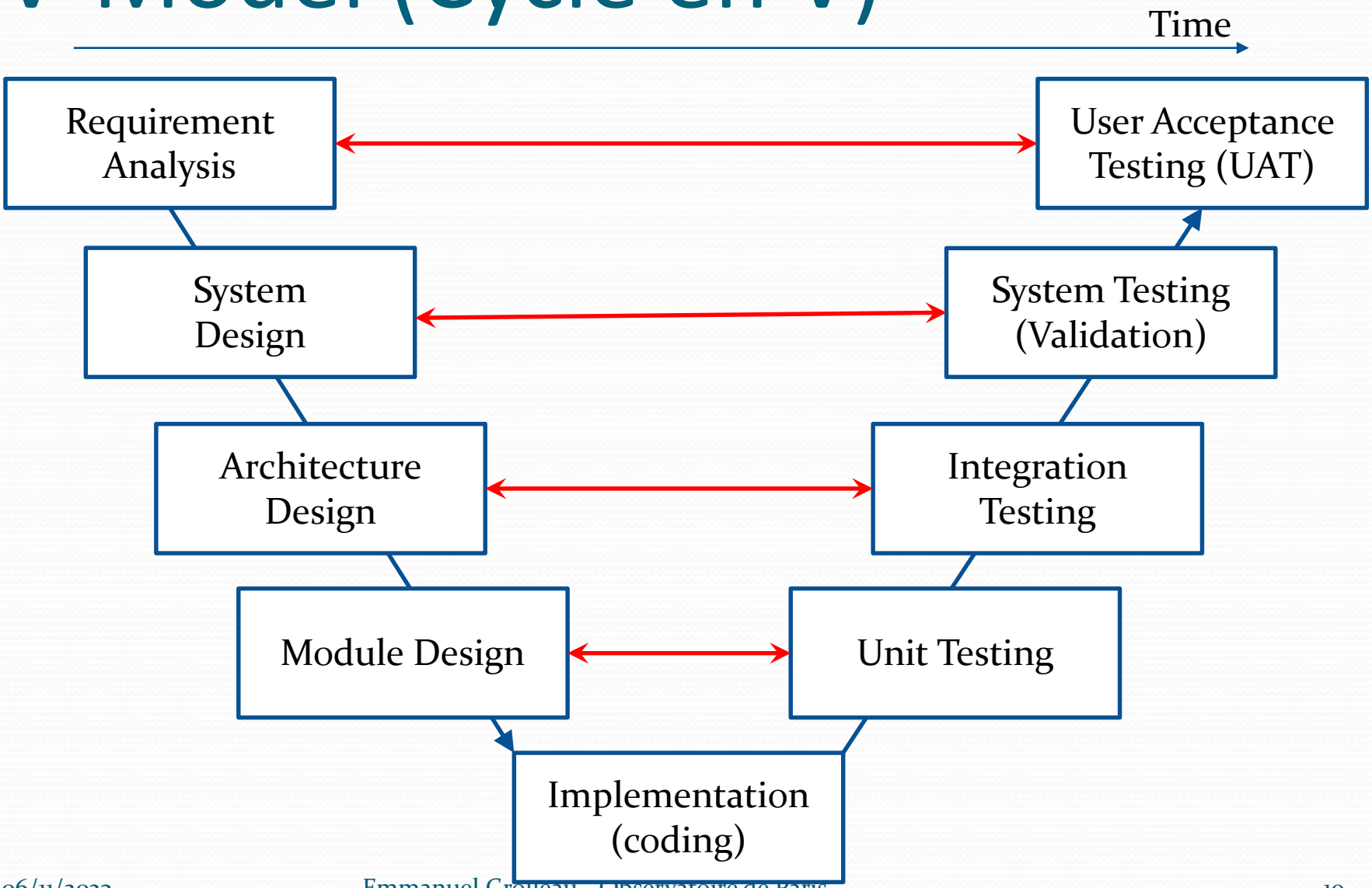
- Iterative development .....



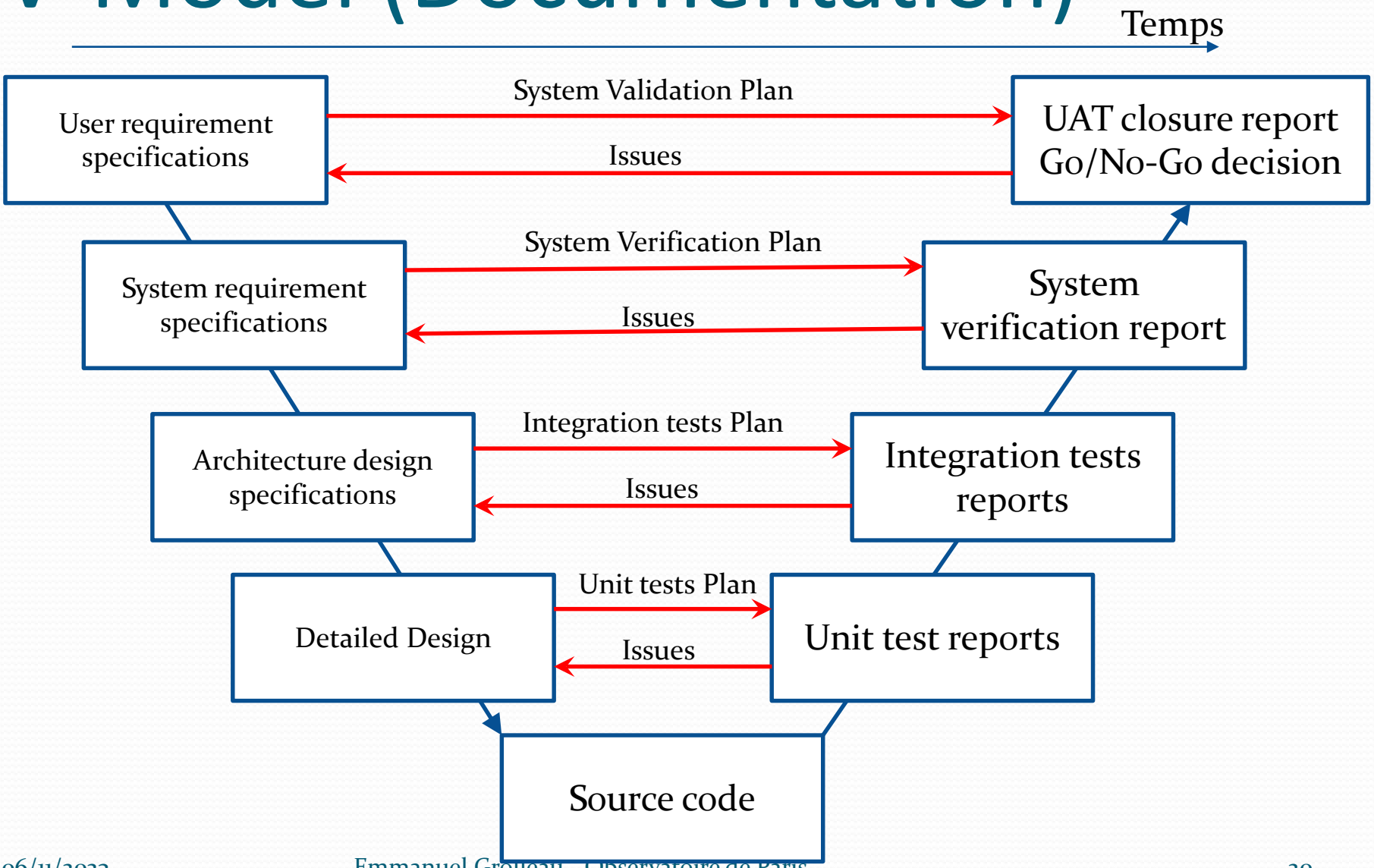
- Agile software development.....



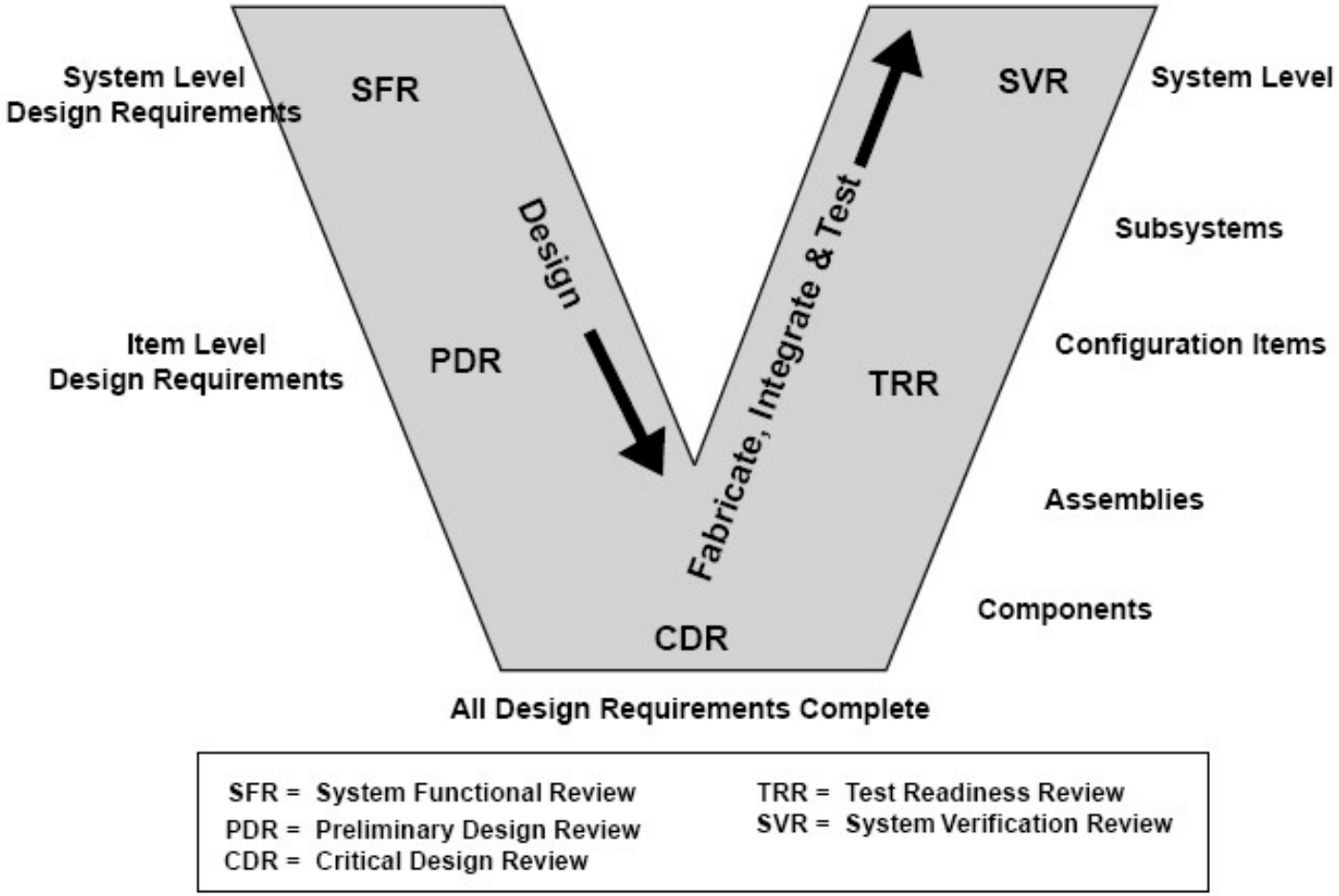
V-Model (Cycle en V)



V-Model (Documentation)



V-Model (Milestones)

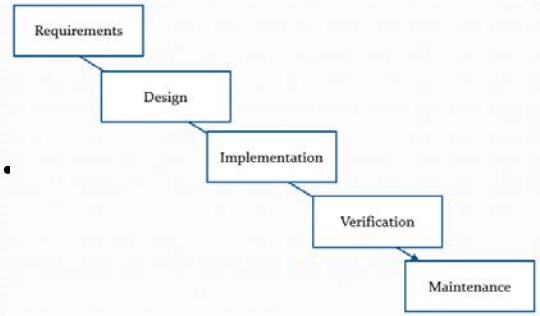


V-Model disadvantages

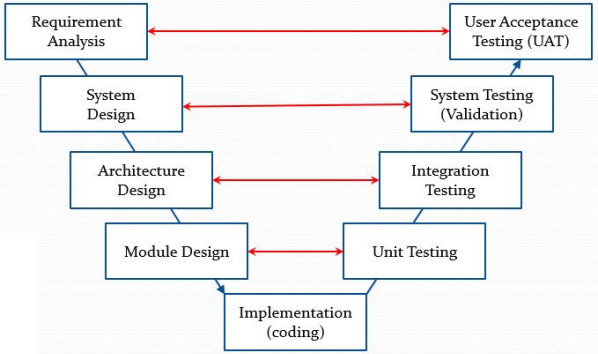
- It is not suitable for projects where requirements are not clear and contains high risk of changing.
- This model does not support iteration of phases.

Systems Development Life Cycle

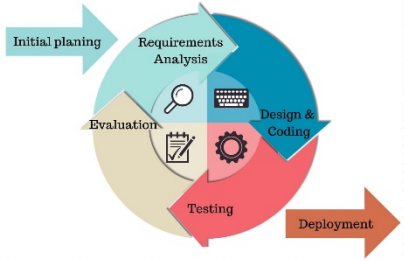
- Waterfall model.....



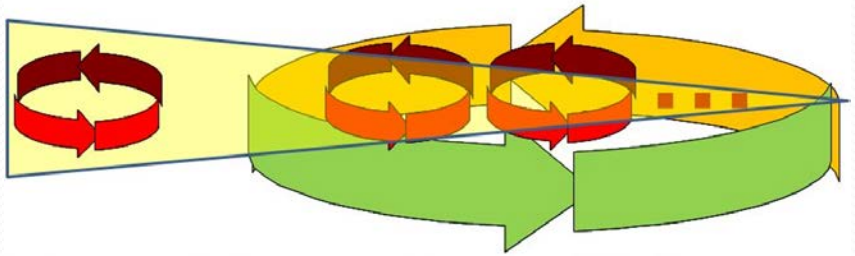
- V-model .....



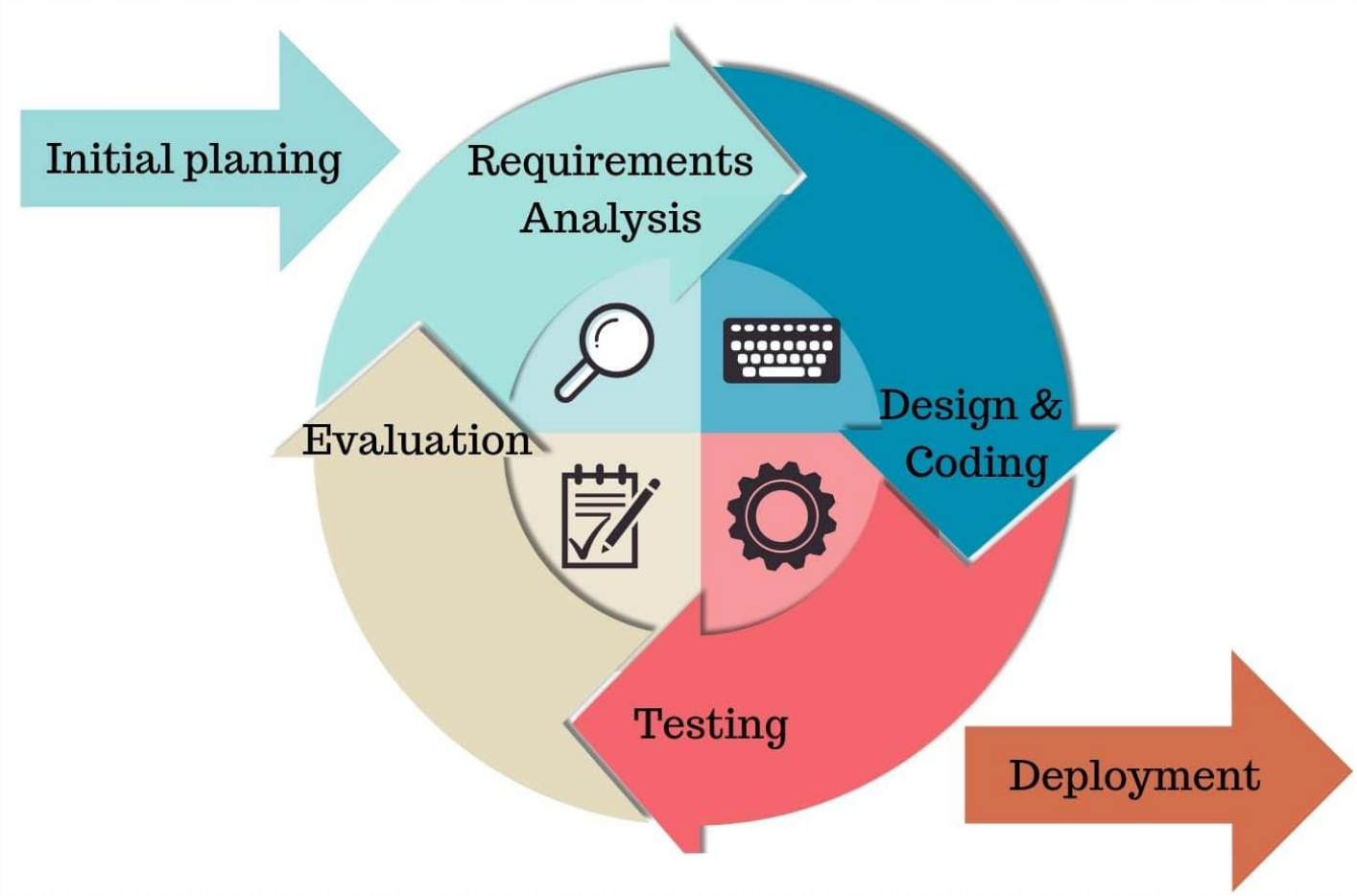
- **Iterative development** ..



- Agile software development.....

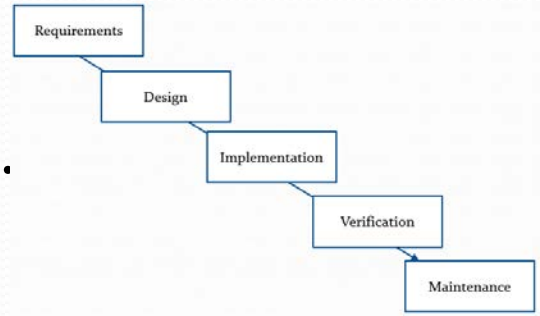


Iterative development (cycle itératif)

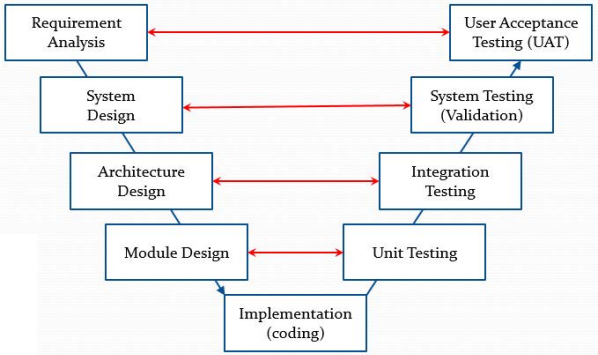


Systems Development Life Cycle

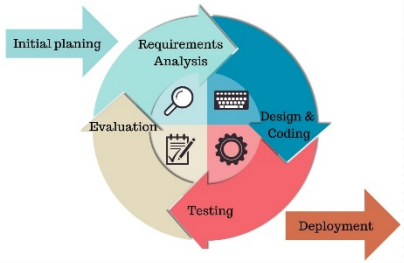
- Waterfall model.....



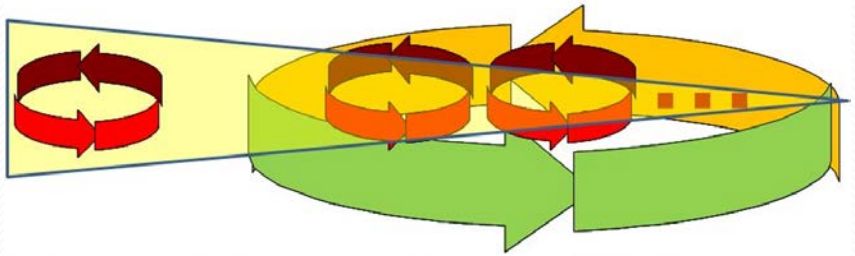
- V-model .....



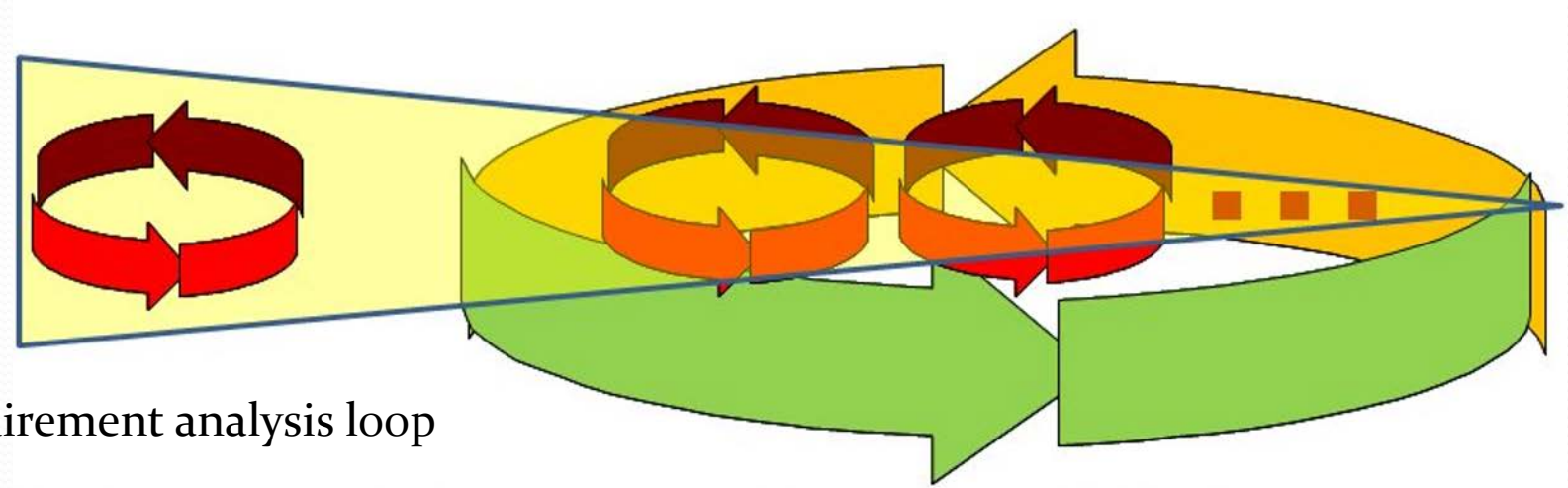
- Iterative development .....



- **Agile software development**.....



Agile software development



Requirement analysis loop

Building and validation loop

Summary

Introduction

- Why test ?
- Software quality
- Systems development life cycle (reminder)
- **Agile software development**

Software testing

- Introduction
- Types of Software Testing
- Test Strategy, Test Plan
- Anomaly
- Coverage matrix (Matrice de couverture)
- Continuous integration and test frameworks

Manifesto for Agile Software Development

- **4 values** and **12 principles**.

We are uncovering better ways of developing software by doing it and helping others do it. Through this work we have come to value:

- **Individuals and interactions** over processes and tools
- **Working software** over comprehensive documentation
- **Customer collaboration** over contract negotiation
- **Responding to change** over following a plan

Manifesto (12 principles)

- Our highest priority is to **satisfy the customer** through early and continuous delivery of valuable software.
- **Welcome changing requirements**, even late in development. Agile processes harness change for the customer's competitive advantage.
- **Deliver working software frequently**, from a couple of weeks to a couple of months, with a preference to the shorter timescale.
- **Business people and developers must work together** daily throughout the project.

Manifesto (12 principles)

- **Build projects around motivated individuals**. Give them the environment and support they need, and trust them to get the job done.
- The most efficient and effective method of conveying information to and within a development team **is face-to-face conversation**.
- **Working software** is the primary measure of progress.
- Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely.

Manifesto (12 principles)

- Continuous attention to **technical excellence** and good design enhances agility.
- **Simplicity**--the art of maximizing the amount of work not done--is essential.
- The best architectures, requirements, and designs emerge from **self-organizing teams**.
- At regular intervals, the team reflects on how to become more effective, then **tunes and adjusts** its behavior accordingly.

Summary

Introduction

- Why test ?
- Software quality
- Systems development life cycle (reminder)
- Agile software development

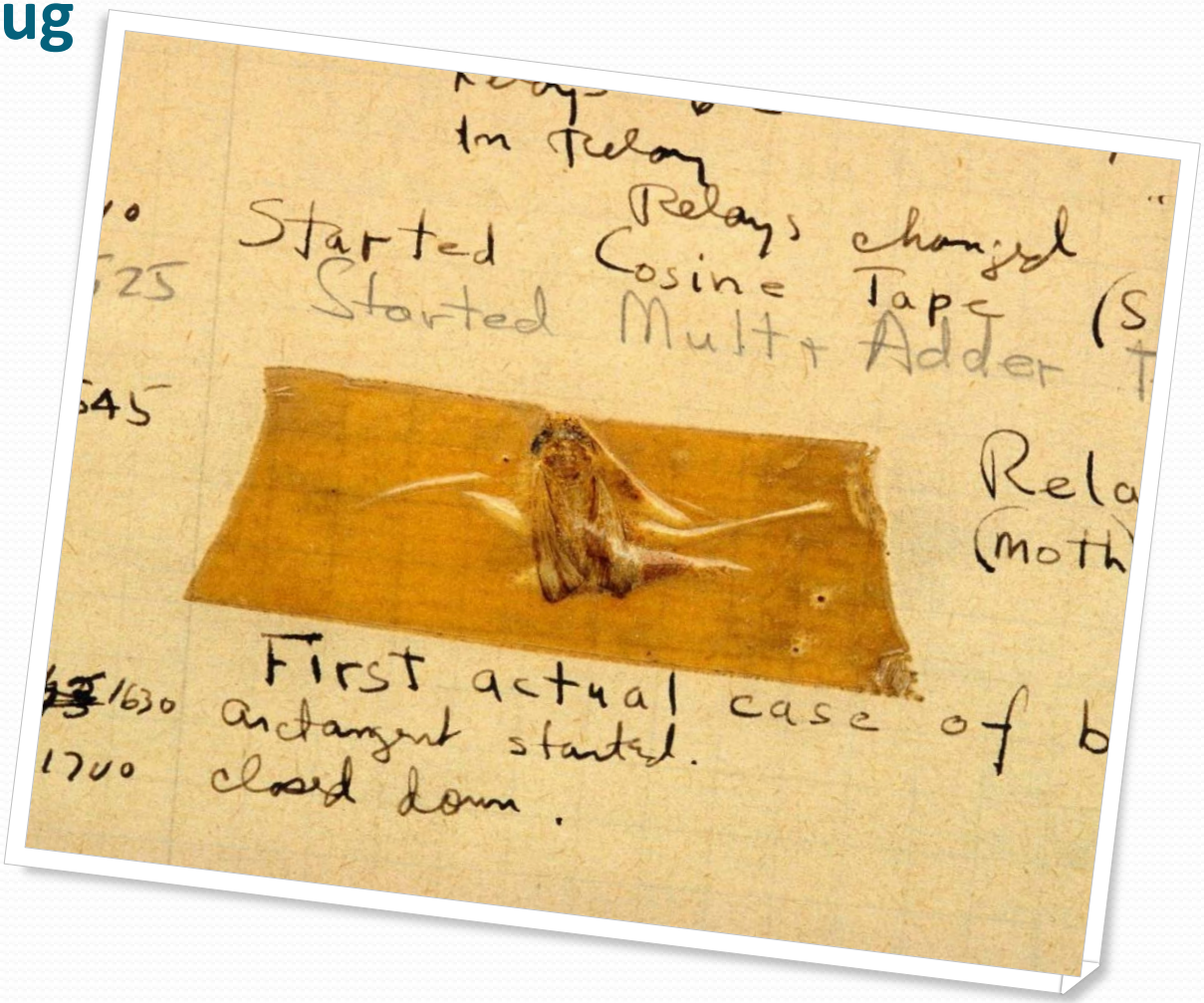
Software testing

- Introduction
- Types of Software Testing
- Test Strategy, Test Plan
- Anomaly
- Coverage matrix (Matrice de couverture)
- Continuous integration and test frameworks

Software testing

First computer bug report (1947)

Grace Hopper's operational logbook for the Harvard Mark II computer



Some terminologies

Error: A human being can make an **error** (*mistake*).

Defect (défaut): error produce a **defect** (*bug*) in the program code, or in a document. It may result in failure or not.

Failure (anomalie): If a defect in code is executed, the system may fail to do what it should do (or do something it shouldn't), causing a **failure**. Defects in software, systems or documents may result in failures, but not all defects do so.

Bug (bogue) : definition not very clear, synonym for both defect & failure.

Error → Defect → Failure

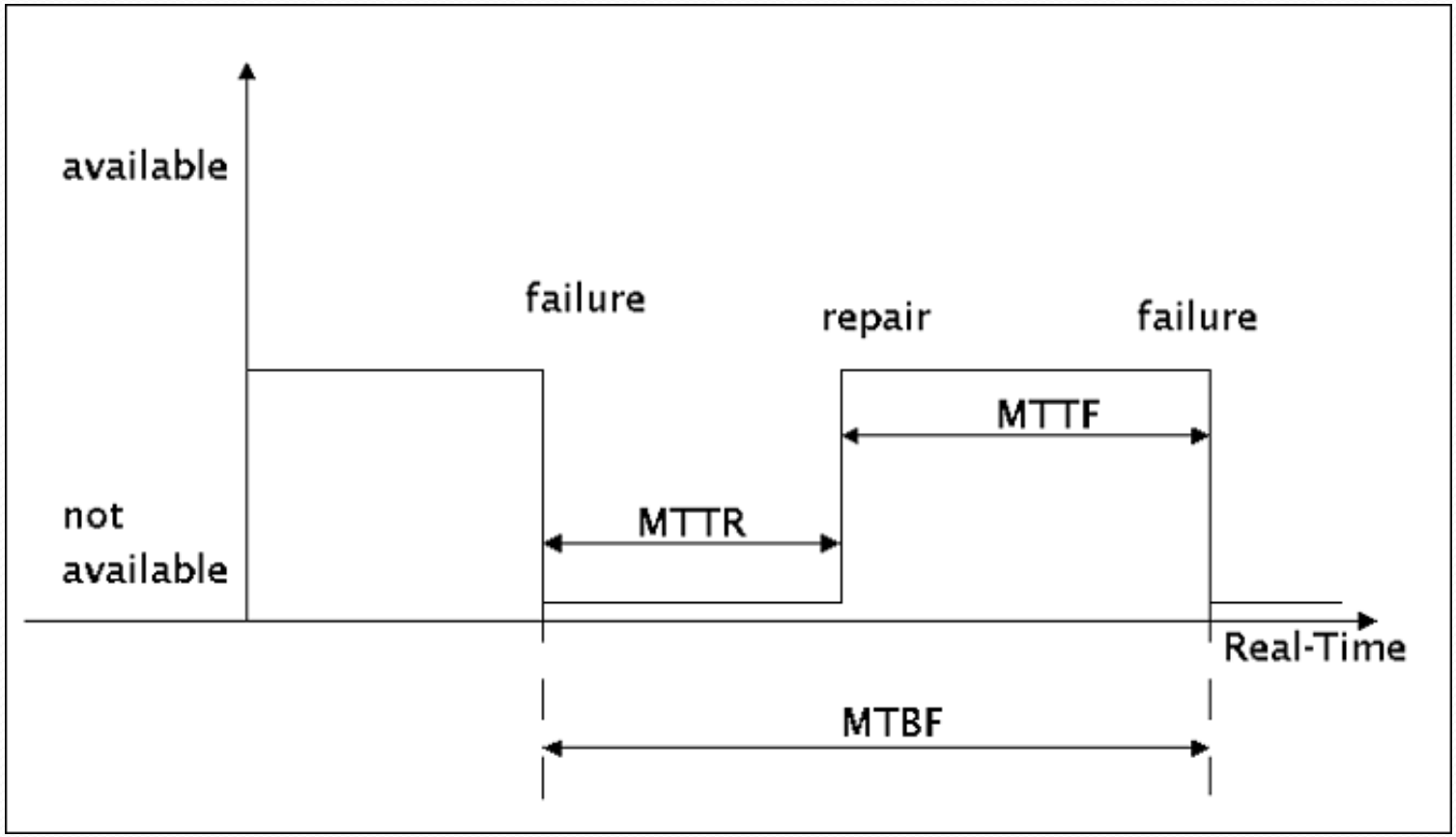
One Statistic

A good developer writes about:

50

defects per 1000 lines of code.

Failure metrics



$$Availability = \frac{MTBF}{MTBF + MTTR}$$

Software testing

- Whatever the software development life cycle you use, testing is essential
- In agile software development the main goal is to provide customer with working software asap, that means testing is a **continuous process !**

Definition

- What is software testing ?

Testing is the **process** of evaluating a system with the intent to **find defect**.

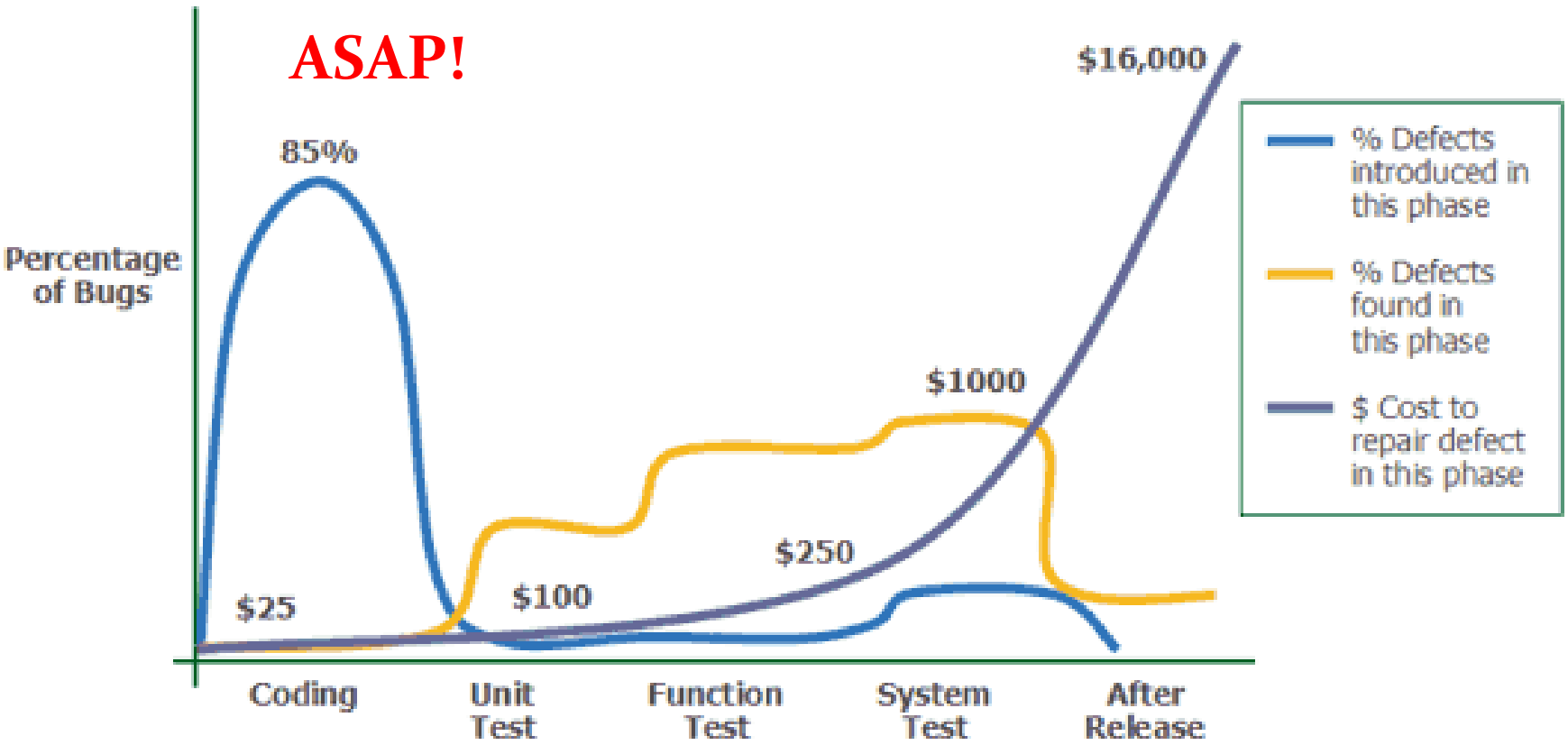
- Defect compared to what?
- **Compared to specifications**

- Therefore, we need exhaustive and accurate specifications, so that:
  - Programmer can develop the software without defect
  - Tester can find defects with regard to specifications

Program testing can be a very effective way to show the presence of bugs, but it is hopelessly inadequate for showing their absence.

Edsger W. Dijkstra , The Humble Programmer (1972), Communications of the ACM 15 (10), octobre 1972

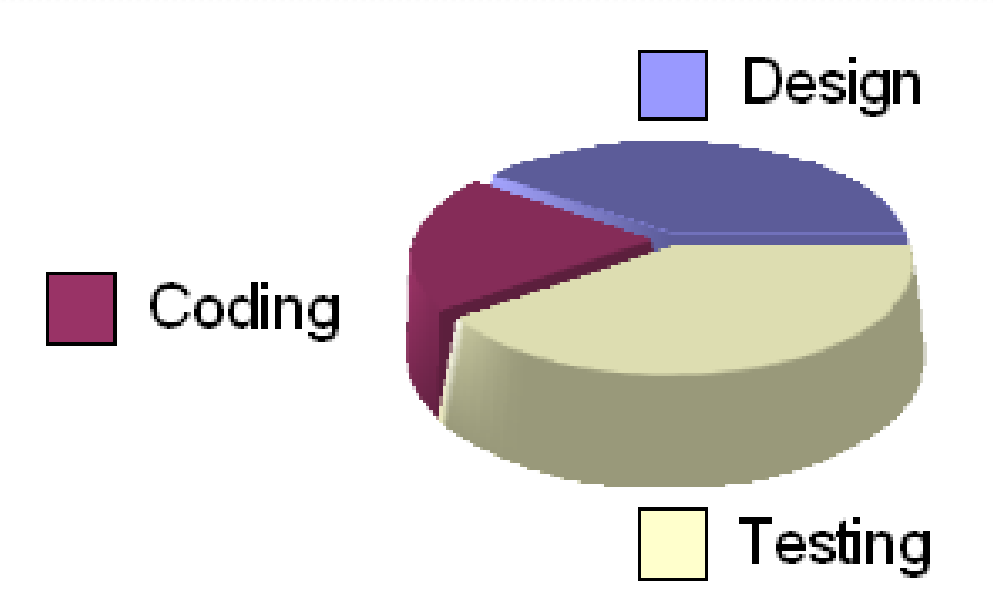
When should software be tested?



Source: Applied Software Measurement, Capers Jones, 1996

The 40 - 20 - 40 rule

- developer should spend 40% of total time for project analysis and design,
- 20% for programming
- **40% for testing**



The 40 - 20 - 40 rule

Focus only on coding phase certainly leads to a project failure.

After coding, we only reach around 50 % of the project length (that 's not so true for agile software development since testing phase is a continuous process).

Misconceptions about software testing

- We should find all defects
  - The truth: We should find all defects which don't meet the specifications
- I develop without any error, it is not useful to test…
- Testing is simple and easy
- Testing happens at the end
  - The truth testing happens all the time
- Anyone can test software
  - The truth: Testing takes skill and training

Software testability

- **Testability**: degree to which a software supports testing in a given test context

- **Testability factors**:
  - Accuracy and exhaustiveness of specifications
  - Simplicity of the architecture, modularity
    - A component under test can be tested in isolation?
  - Possibility to observe (intermediate and final) test results
  - Diversity in used technologies requires to use diverse test methods and tools in parallel.
  - …

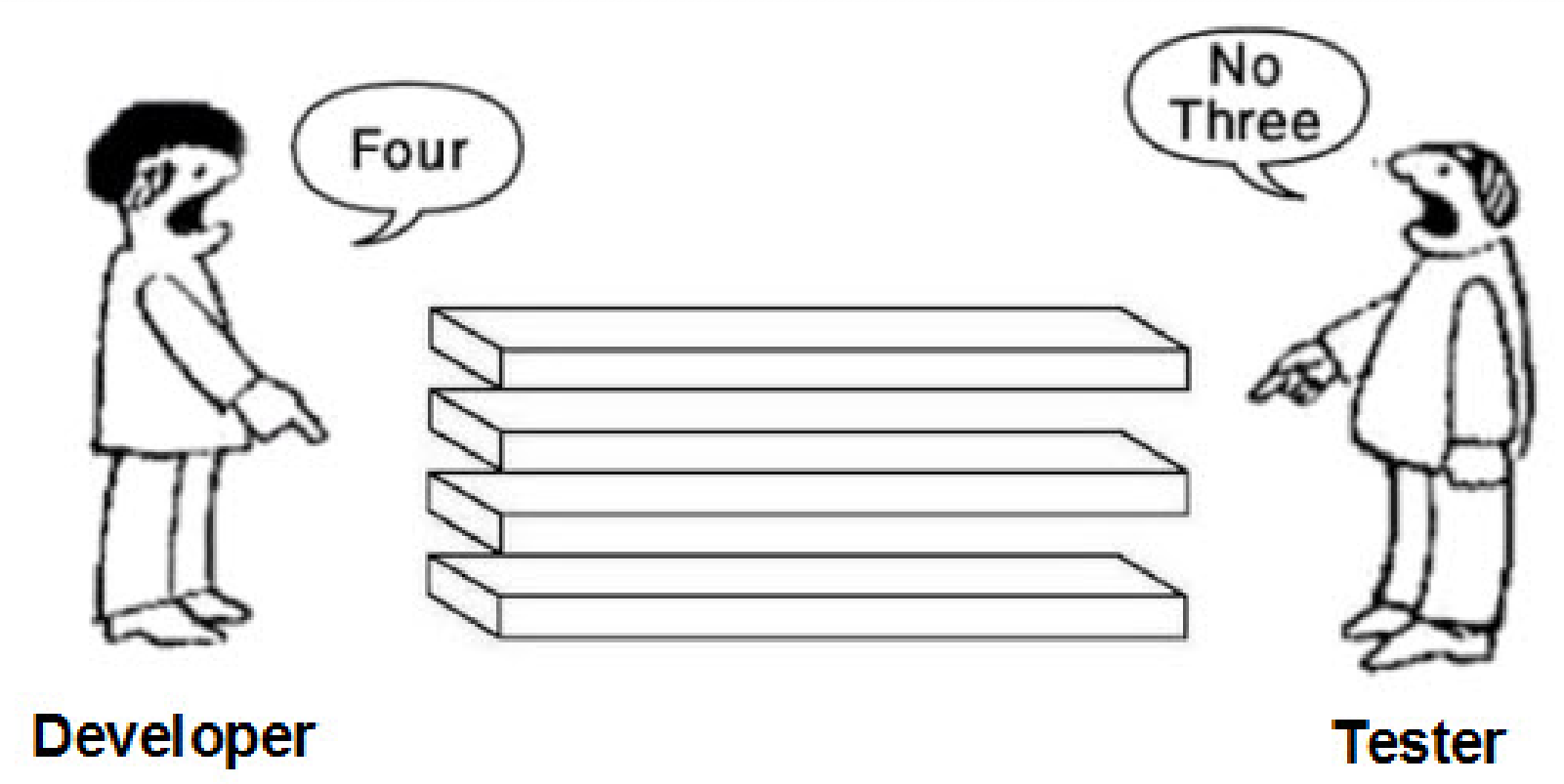
Verification & Validation (V&V)

- Verification
  - *Are you building it right?*
- Validation
  - *Are you building the right thing?*

6 principles

F. Xavier. Fornari., "Introduction to Software Testing", Esterel Technologies

- P1: **Independence**
  - A developer should not test his own program.
- P2: **Paranoia**
  - Do test thinking there is obviously some defects.
- P3: **Prediction**
  - Before executing the test, expected results should be defined.


Four
No Three
Developer
Tester

6 principles

F. Xavier. Fornari., "Introduction to Software Testing", Esterel Technologies

- P4: **Verification**
  - Each test result should be meticulously analysed.
- P5: **Reliability**
  - Test cases should be written for unexpected input values as much as expected input values.
- P6: **Completeness**
  - We have to check that:
    - Software does what it should do
    - Software doesn't do what it should not do

Summary

Introduction

- Why test ?
- Software quality
- Systems development life cycle (reminder)
- Agile software development

Software testing

- Introduction
- **Types of Software Testing**
- Test Strategy, Test Plan
- Anomaly
- Coverage matrix (Matrice de couverture)
- Continuous integration and test frameworks

Types of software testing

- Do several types of software testing exist?

Types of software testing

Static / Dynamic approach

- **Static** testing
  - Testing is done without executing the program
  - Before compilation
  - Somebody reads the code to prevent defects

- **Dynamic** testing
  - Testing is done by executing the program
  - After compilation
  - Involves test cases for execution with expected results

Types of software testing

Functional / Structural approach

- **Functional** testing
  - Does the software meet specifications.

- **Structural** testing
  - Test structure of code to detect defect.
  - Check that the software won't crash (overflow, not initialised variable, …)

Types of software testing

Functional / Structural approach

- **Functional** testing (**Black box testing**)
  - Based on customer's requirements
  - Specific knowledge of the application's code, internal structure is not required
  - Test cases may be written before code
  - Testing is not performed by the software developer (eg. QA Engineer)

- **Structural** testing (**White box testing**)
  - Tester knows the code and chooses inputs to walk through various paths and determine the expected outputs.
  - Tests internal structures of an application (control flow, …)
  - Test cases can't be written before code
  - Testing is often performed by the software developers

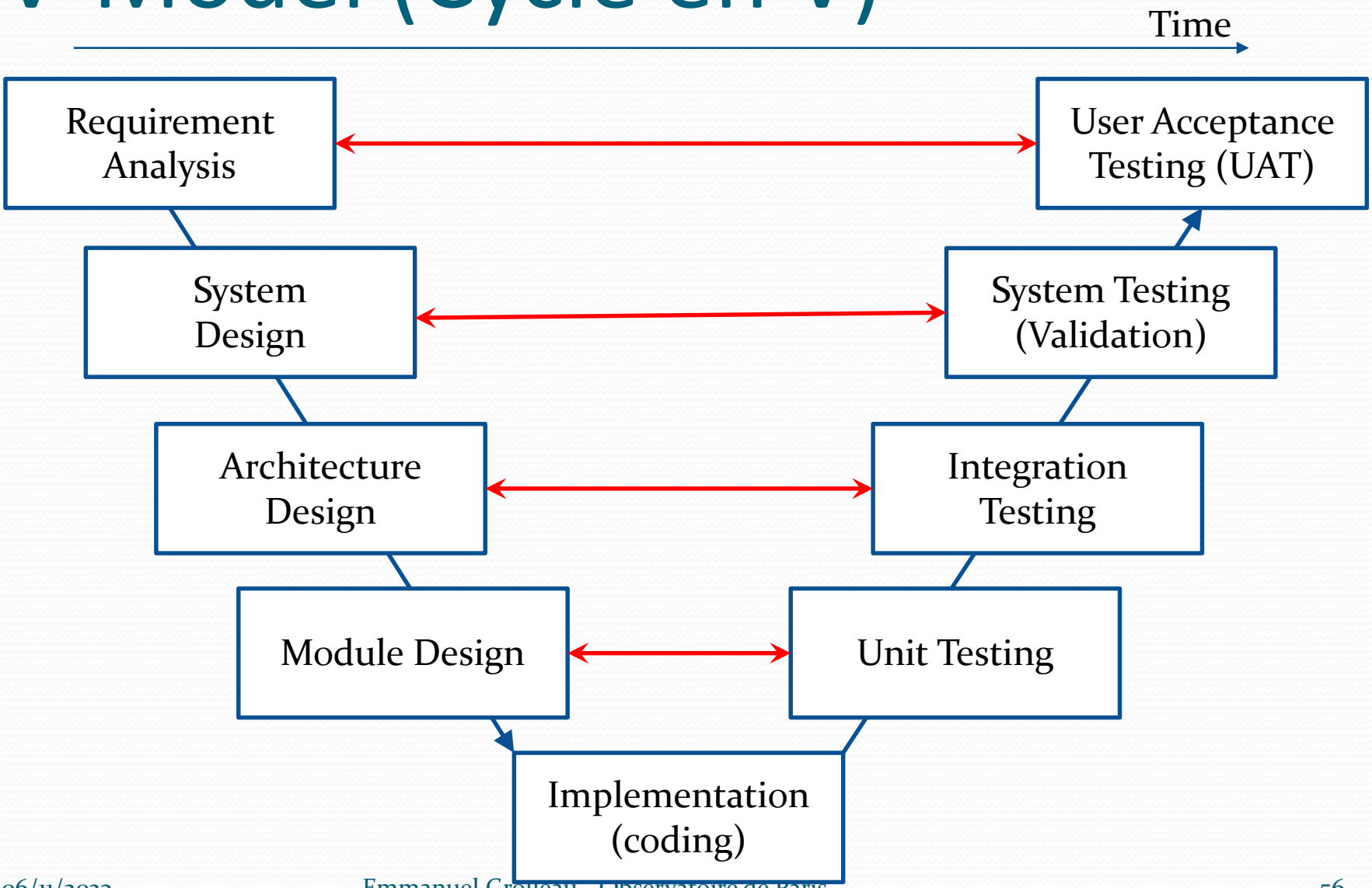
Types of software testing

Manual / Automation approach

- **Manual** testing
  - Tests are executed manually by developer or Q.A. engineer
  - Software testers execute the test cases and generate the test reports without the help of any automation software testing tools
  - He compare the results with expected results
  - Tedious task, possibility of human errors
  - Not possible for big software

- **Automation** testing
  - Use of automation software testing tools to :
    - Execute the test
    - Save the results
    - Compare results to expected results automatically
    - More reliable

V-Model (Cycle en V)



Types of software testing

Systems development life cycle approach

Based on V-model we can identify 4 testing levels:

- **Unit** testing
  - Test of small component separately
- **Integration** testing
  - Test of several component which have interactions Actually test the interface between components.
- **System** testing
  - Test the functional and non-functional requirements of the whole system .
- **User acceptance** *testing* (UAT)
  - *Quite similar to system testing but:*
    - Performed in a user environment close to the production environment
    - Performed by the customer

Tiny project

<u>Functional specifications</u>

- Our customer wants a software intended to compute a mean, and/or a median and/or a standard deviation based on a list of values saved within a text file.

- Moreover, results (mean, median, standard deviation) should be written in a result file.

Tiny project

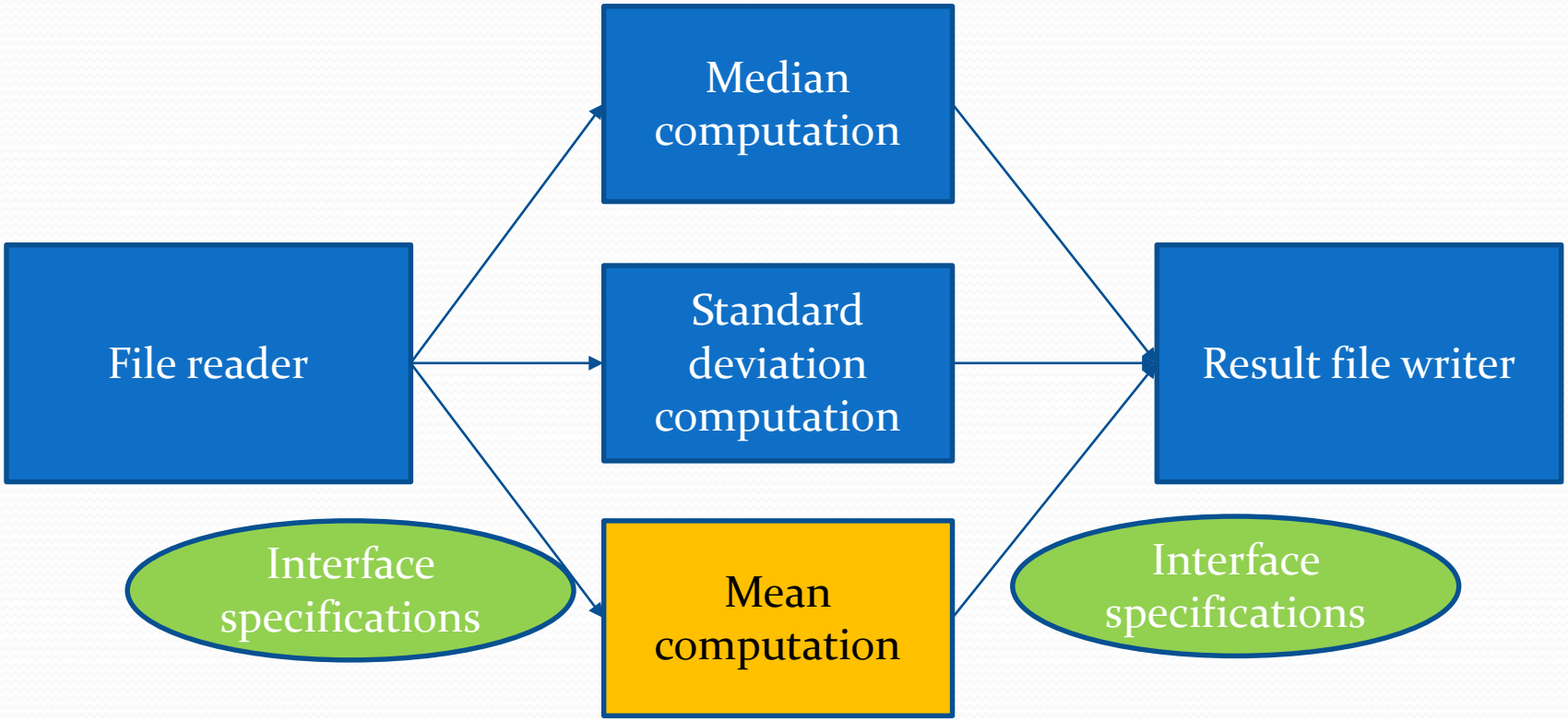
- Could you give some examples of **system testing** for this software?
  - Using several input values within a file, compute:
    - A mean
    - A median
    - A standard deviation
    - A mean and a standard deviation
    - …
  - Compare result file value(s) to expected results

Tiny project

Existing module (To be integrated)

New module

Architectural design

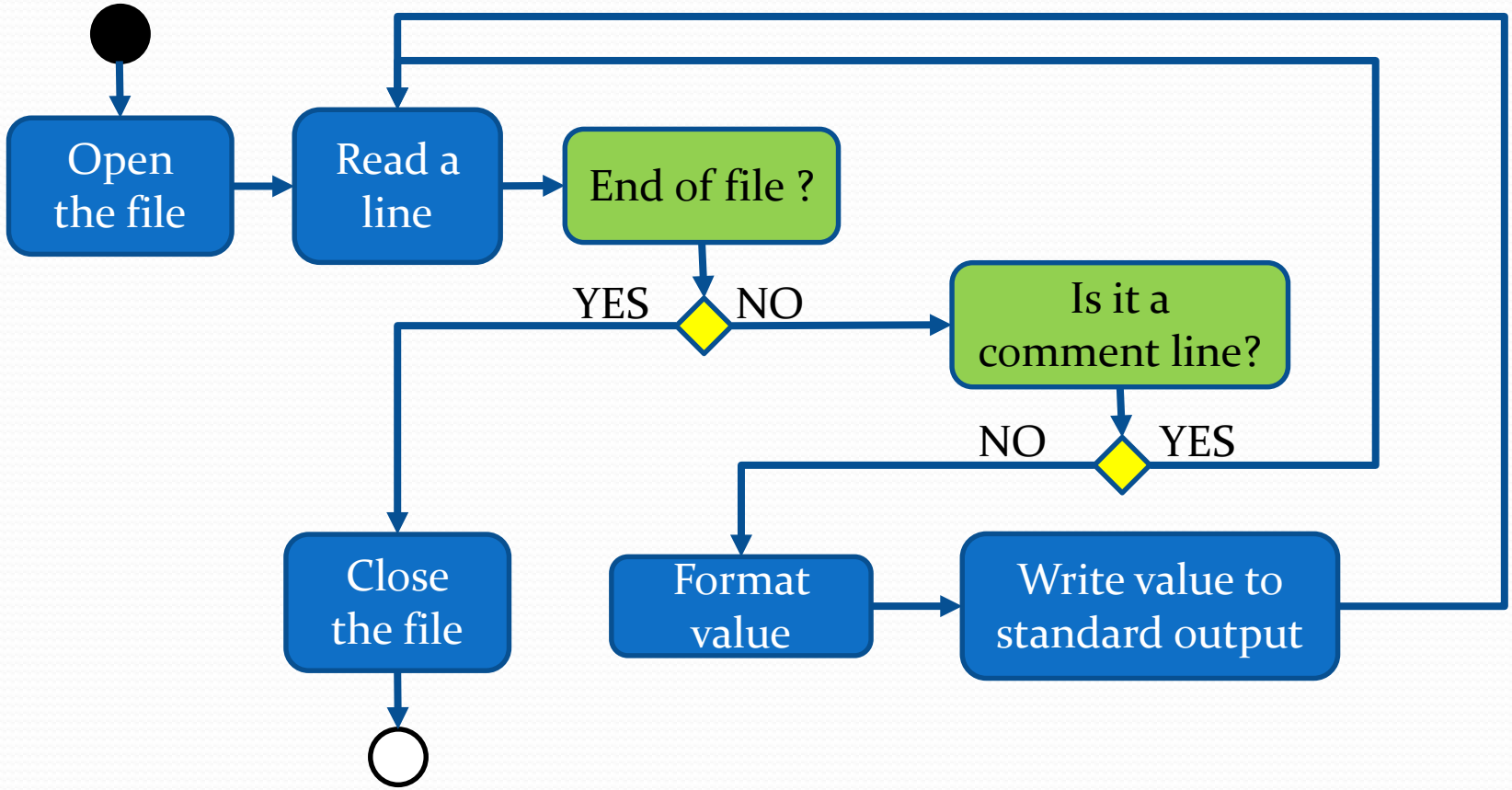


Tiny project

- Could you give some examples of **integration testing** for this software?
  - Perform reading of the input file and computation of the mean sequence
  - Perform reading of the input file, computation of the mean, computation of the median sequence
  - Perform median computation and file writer module sequence using cat command to push data into the pipeline
  - ….

Tiny project

<u>Module design,</u> File reader



Tiny project

- Could you give some examples of **unit testing** for this software?
  - Use cat <file> command to write data into standard input of deviation computation module, perform the module then read the results on standard output
  - Perform file reader module and read the results on the standard output
  - …

Tiny project

Same way for other modules

<u>Coding,</u> File reader module

OPEN file
WHILE NOT end of file
    READ line
    IF line is a comment, **CONTINUE**
    FORMAT value
    WRITE value to standard output
CLOSE file

Software quality concepts
ISO 9126 standard

- The standard **ISO 9126** (Software quality) defined 6 groups of software **quality characteristics** (facteurs de qualité) : **FURPSE**
  - F (Functionality): meet the requirements
  - U (Usability): the ease of use
  - R (Reliability): to maintain service under defined conditions for defined periods of time
  - P (Performance, efficiency): system resources used when providing the required functionality
  - S (System Maintainability): ability to identify and fix a fault within a software component
  - E (Portability (Evolutivity)): how well the software can adapt to changes in its environment

Types of software testing

Quality characteristics approach

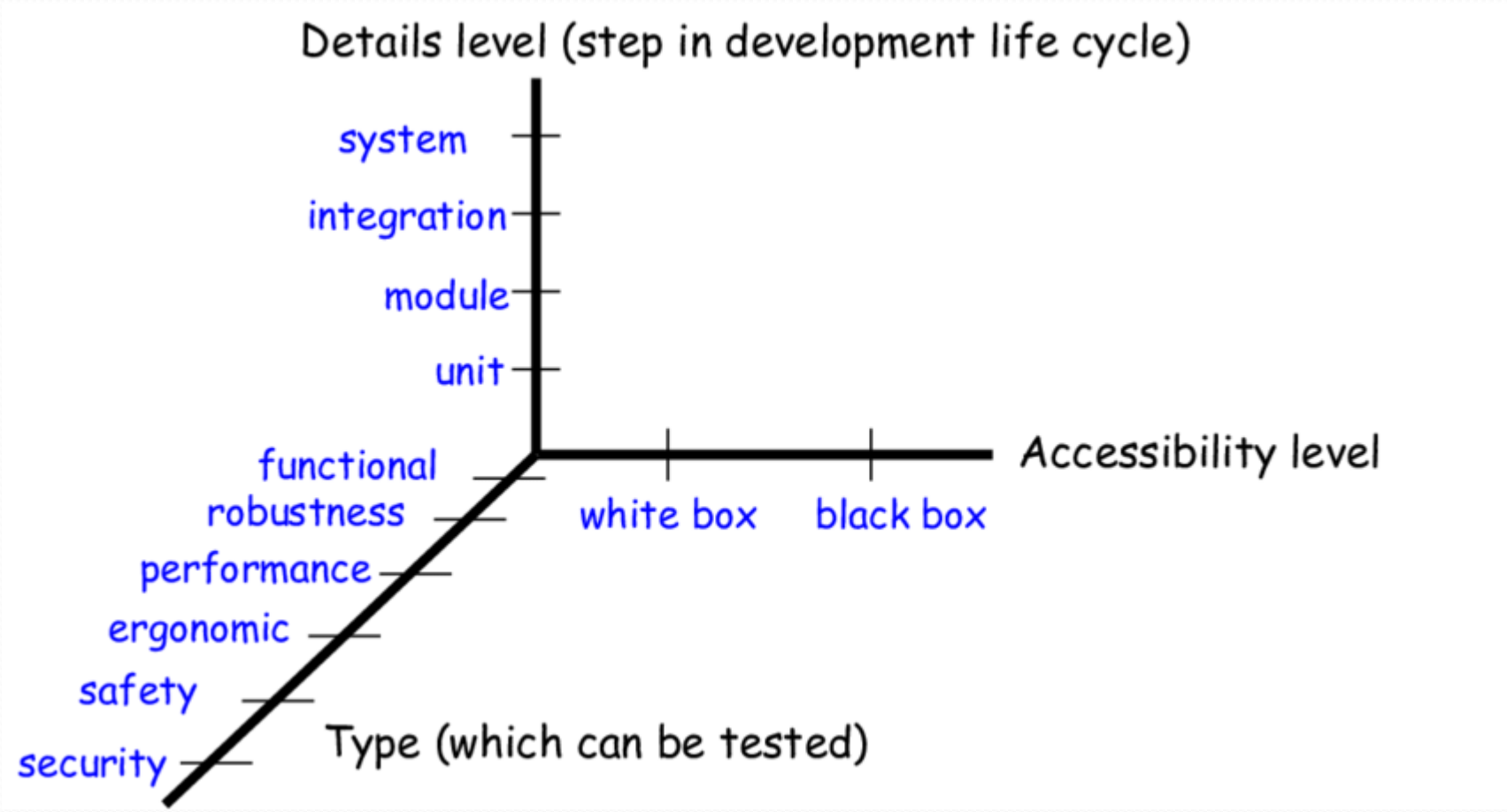
- F : Functionality
  - **Functional testing**: check that functionalities meet the requirements
  - **Vulnerability testing**: check the software security
- U: Usability
  - **User testing:** a real user perform test cases in realistic conditions
- R : Reliability
  - **Reliability testing**: software's ability to function, given environmental conditions, for a particular amount of time

Types of software testing

Quality characteristics approach

- P : Performance
  - **Performance testing**: check that the performance meet the performance requirements (how a software performs in terms of responsiveness and stability under a particular workload)
- S : System Maintainability
  - testing the system's ability to update, modify the application
  - **Regression testing**
- E : Portability
  - **Portability testing:** test on various O.S.

Types of software testing



Kind of testing approaches as proposed by J. Tretmans

Reliability testing

- Testing a software's ability to function, given environmental conditions, for a particular amount of time
  - Maturity / frequency of failure of the software
  - Fault tolerance / ability of software to withstand from failure
  - Recoverability / bring back a failed system to full operation

- Test cases : create inputs
  - At the limit of their definition range
  - Outside the limit of their definition range
- Turn off the computer when the software is running, then restart.

Tiny project

- Reliability testing, create test cases (input files)
  - With negative values
  - Outside the limits of the definition range, negative or positive values
  - With no any value
  - With billion of lines

These test cases can be executed as unit, integration or validation testing.

Performance testing

Nominal load testing

- Check the software response time in nominal load conditions (regarding the specification)

- Check the software resources usage (CPU, memory, I/O, …) in nominal load conditions

Stress testing, same as previous tests but:

- Check the performance of the software under maximum work load (e.g. high number of users)

Tiny project

- Performance testing
  - Create use cases with N millions of lines.
    - N between 1 to 10
    - Plot response time against N
  - Perform this for unit/integration/validation testing

Regression testing

- Some studies suggest that, on average, programmers **fixing a bug** will **introduce a new bug**

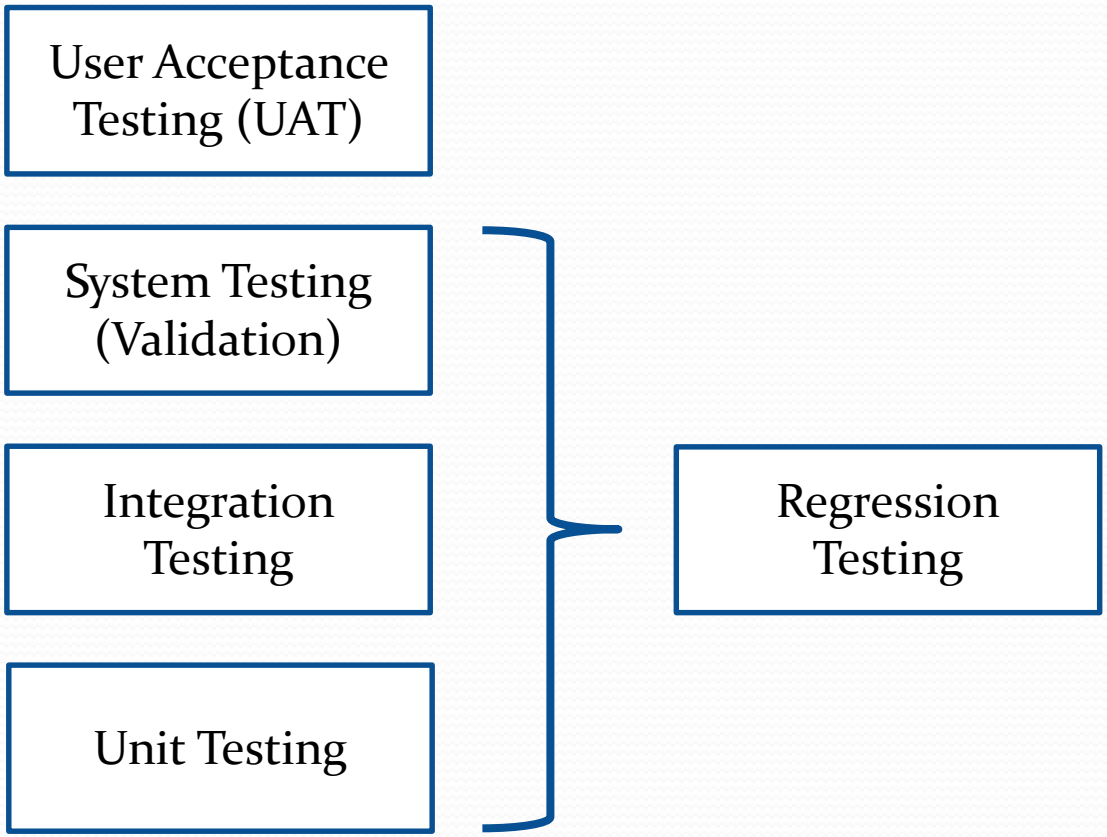
  50 %

  of the time

- **Regression testing** is used to check if any new bugs have been introduced through previous bug fixes.

- Since there are plenty of previous testing, it is better to use a continuous integration framework

Regression testing



Summary

Introduction

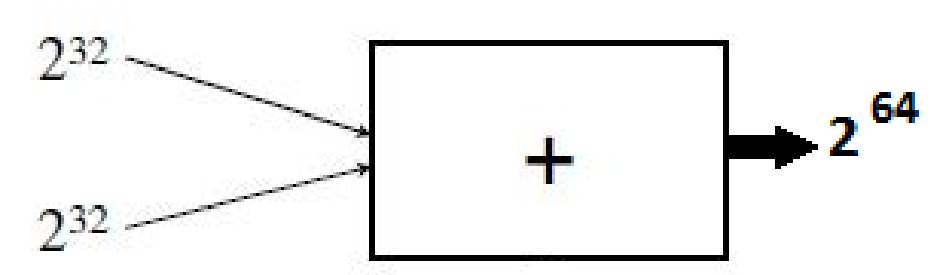
- Why test ?
- Software quality
- Systems development life cycle (reminder)
- Agile software development

Software testing

- Introduction
- Types of Software Testing
- **Test Strategy, Test Plan**
- Anomaly
- Coverage matrix (Matrice de couverture)
- Continuous integration and test frameworks

Test strategy

- The number of possible tests for even simple software components is practically infinite
  - Combinatorial explosion



- We have to define a **test strategy** : select tests that are feasible for the available time and resources

Test strategy

- Test strategy defines:
  - Resources (human, material)
  - Test process

- Test strategy is compelled by:
  - Software development life cycle
  - Development languages
  - Kind of app
  - …
- The materialization of test strategy is the « **test plan** »

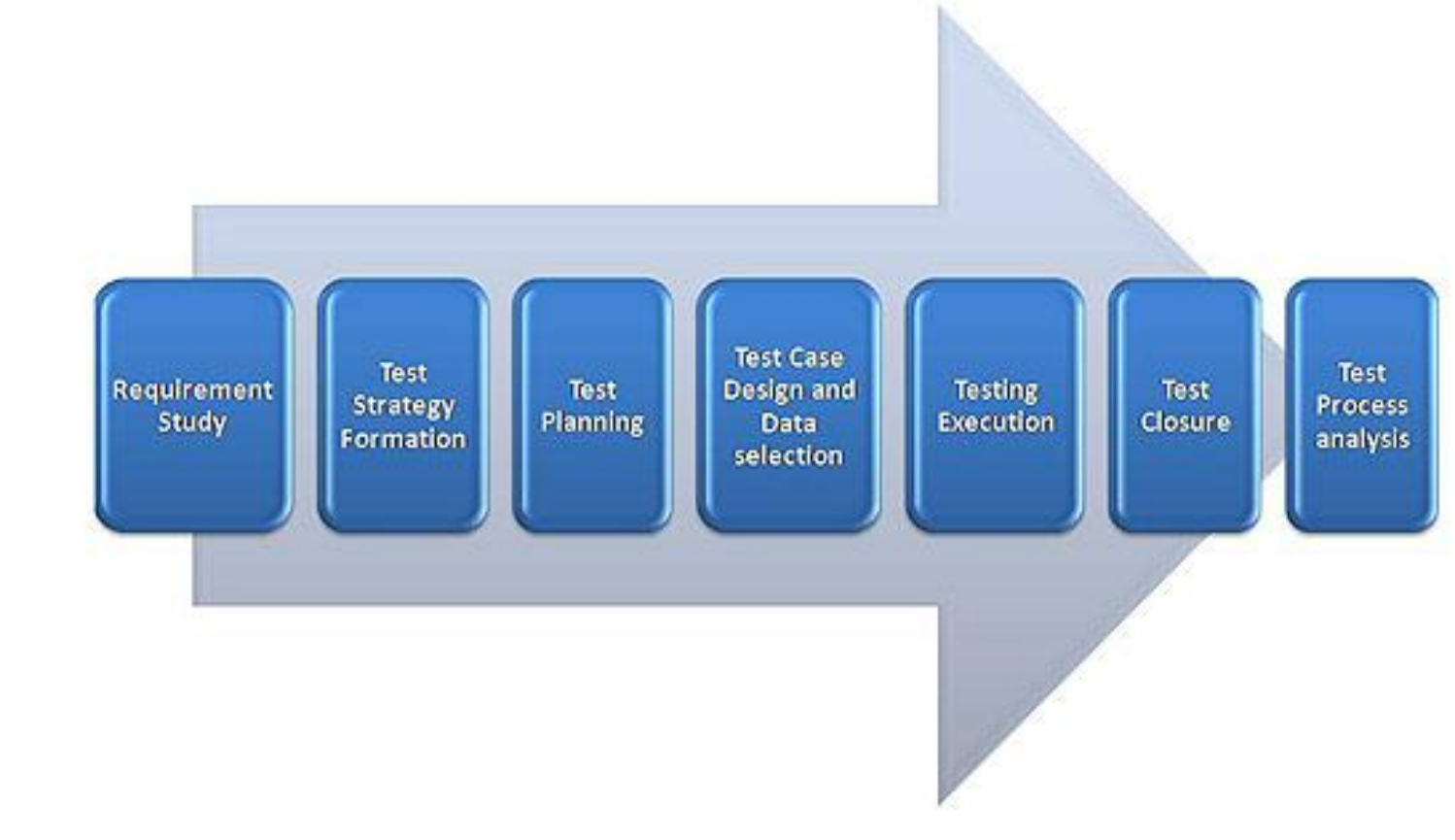
Test strategy
Test process

Test process

The process consisting of all life cycle ***activities****,* both ***static*** and ***dynamic****, concerned with* ***planning, preparation*** and ***evaluation*** *of software products to determine that they* ***satisfied specified requirements****, to demonstrate that they are* ***fit for purpose*** *and to* ***detect defects****.*

International Software Testing Qualifications Board (ISTQB)

Test strategy
Test process



Test strategy
Test process

- Test process is a process based on **resources**:
  - Human resources: test operator, test manager, Quality Assurance manager, …
  - Tools: VCS, storage & backup systems for use cases inputs/outputs, test automation Framework, …

Stratégie de test
Définitions

Test : ensemble d'un ou plusi

Cas de test : un ensemble de ées
de tests), de **préconditions** d
attendus et de **post conditi**
développées pour un **objectif** s
particulière.

Exemple : exécuter un chemin particulier d'un programme ou vérifier le respect d'une exigence spécifique.

walkthrough.

Test case A set of input values, execution preconditions, expected results and execution postconditions, developed for a particular objective or test condition, such as to exercise a particular program path or to verify compliance with a specific requirement.

Stratégie de test
Plan de test

C'est un document décrivant l'étendue, l'approche, les ***ressources*** *et le* ***planning*** *des activités de test prévues. Il identifie entre autres les* ***éléments*** *et caractéristiques à tester,* ***qui*** *fera chaque tâche, le degré d'indépendance des testeurs,* ***l'environnement*** *de test, les* ***techniques*** *de* ***conception*** *des tests et les techniques de* ***mesure*** *des tests à utiliser, et tout risque.*

[IEEE 829].

Plan de test

Le plan de test doit répondre aux questions :

Pourquoi tester ?

- Les plans de test logiciel doivent répertorier les objectifs à atteindre

Quoi tester ?

- Périmètre d'intervention de l'activité de test, lister les éléments du logiciel qui seront testés
- Définir les éléments qui seront exclus de la stratégie

Source : ALL4TEST.

Plan de test

Comment tester ?

- Approche globale du test, c.-à-d.. spécifier les niveaux de test, les types de test et les méthodes, en fonction des objectifs.
- Définir les critères utilisés afin d'établir si chaque élément du logiciel a réussi ou échoué.
- Définir les ressources matérielles :
  - configuration matérielle
  - configuration logicielle
  - outils de production et de support des tests
  - prérequis fonctionnels et techniques

Source : ALL4TEST.

Plan de test

Qui teste ?

- responsabilités de chaque équipe/personne
  - Test Manager, opérateur, …

Quand ?

- calendrier des tests

Définition des livrables

- Plan de Test
- Cas de test (données d'entrée, résultats attendus, …)
- Scripts de Test
- Un rapports de test contenant des fiches de test, journal de test, fiche d'anomalie

Source : ALL4TEST.

Plan de test (exemple)

Extrait du **Plan d'Assurance Qualité** pour le développement des applications du Segment Sol de la mission spatiale CoRoT :

QUOI

*Chaque **application est l'objet de tests**. Les tests des applications du segment sol doivent répondre, **de la manière la plus exhaustive possible, aux exigences** émises dans le document DA 11 (Spécification ….).*

POURQUOI

QUOI **QUI**

*Les tests **unitaires** sont **à la charge des développeurs**. Pour chaque produit dont ils sont responsables, les développeurs doivent écrire au moins un test de cas **nominal** par « fonction » (telle que définie dans le document DA 11) et plusieurs tests de **cas aux limites** (paramètres en entrée proche des valeurs limites) et plusieurs tests de **cas dégradé** (paramètre d'entrée hors des limites de fiabilité de l'algorithme).*

COMMENT

Plan de test (exemple)

QUOI

*Les **tests d'intégration, de validation et de non régression** sont élaborés en accord avec le **chef de projet logiciel et le responsable qualité des laboratoires**.*

QUI

*Les tests sont **numérotés** séquentiellement pour chaque produit. Les fichiers de tests sont placés dans le répertoire « test » de chaque produit. Chaque test **est placé dans un répertoire** T<numéro du test>.*

*Le répertoire « in » contient **les données d'entrée** du test fournis par le responsable de produit. Le répertoire « out » contient **les résultats attendus** de l'exécution du process (i.e. les résultats de référence en vue de la recette).*

LIVRABLES

***Chaque exécution** d'un test doit faire l'objet d'une **fiche de test**. Le numéro d'identification du test est composé comme suit :*

<CODE_PRODUIT>.T<Numéro_chronologique>

Plan de test (exemple)

La fiche de test *indique le « type de test » dont les modalités sont les suivantes :*

- *U pour les tests unitaires …;*
- *I pour les tests d'intégration …;*
- *N pour les tests de non régression …;*
- *V pour les tests de validations ;*
- *P pour les tests de performance.*

La fiche de test est placée directement dans le répertoire T<numéro du test>.

Les fiches de test doivent clairement indiquer comment les données d'entrées doivent être accédées et où seront placées les données de sortie afin de pouvoir les comparer aux fichiers présents dans le répertoire out.

FICHE DE TEST			
N° Test : CODE_PRODUIT.TX	Opérateur :	**Date de test :**	
Titre :		**Type de test : U/I/NR/V**	
Détail des tests	Résultats attendus	Résultats obtenus	**Conforme**
Objectif : **Mode opératoire :**			**Oui** **Non**
Step 1			
Step 2			
Step 3			
=== FIN DU TEST ===			

Données d'entrées :

Environnement :

Observations :

Journal de test

- Il s'agit d'un document qui regroupe pour une campagne de tests donnée, l'ensemble des résultats, généralement sous la forme :
  - Référence du test
  - Date d'exécution
  - Résultat « OK ou Fail »
  - Commentaire en cas d'échec.

Summary

Introduction

- Why test ?
- Software quality
- Systems development life cycle (reminder)
- Agile software development

Software testing

- Introduction
- Types of Software Testing
- Test Strategy, Test Plan
- **Anomaly**
- Coverage matrix (Matrice de couverture)
- Continuous integration and test frameworks

Fiche d’anomalie

- Lorsqu’un test échoue (résultat attendu <> résultat obtenu) pour tout ou partie du test on rédige **une fiche d’anomalie (FA).**
- La FA reprend et complète la partie « résultat obtenu » de la fiche de test mais peut ne porter que sur un point particulier de l’exécution du test.
- De nombreuses applications internet permettent de saisir et de suivre le cycle de vie des FA (Jira, Redmine, Bugzilla, …)

Fiche d’anomalie

Une fiche d’anomalie comporte généralement les champs suivants :

- Référence du produit
- Version du produit
- Référence anomalie
- Rédacteur
- Détectée par
- Date de détection
- Type de fiche (Anomalie ou Demande d’évolution)
- Etat (en fonction du workflow des anomalies, ouvert, en cours de résolution, refusé, résolu, testé, validé, livré, fermé, …)
- Reproductible ou non (Heisenbug)
- Référence du test (le cas échéant)
- Description
- Environnement d’exécution (fichiers joints)
- Historique des échanges

FA sous le logiciel JIRA

JIRA Dashboards Projects Issues Agile Create issue Quick Search

GEDEON / GD-144

Crash TSIM lorsque qu'une TC et une TC time update sont émis en même temps

Edit | Comment | Assign | More | Stop Progress | Resolve Issue | Workflow | Admin

Details

Type:	Bug	Status:	In Progress
Priority:	Major	Resolution:	Unresolved
Component/s:	IO-G2S		
Labels:	None		

Description

Lorsque GTCScript produit une genericCmd au même moment que l'OBT produit un message temps (TC time update aussi nomé TCodeData)il arrive environ un fois sur 5 un plantage de TSIM.

cf https://lesia.obspm.fr/repos/GERICOS/workstation/Divers/EmbeddedAppTests/GD-114/

Activity

All | **Comments** | Work Log | History | Activity | Commits | Subversion

Damien Balima added a comment - 14/Jun/12 11:28 AM

looks like a concurrency problem. It may be fix with a mutex...

People

Assignee:	Damien Balima
	Assign to me
Reporter:	Lee Roy Malac-Allain
Votes:	0 Vote for this issue
Watchers:	2 Start watching this issue

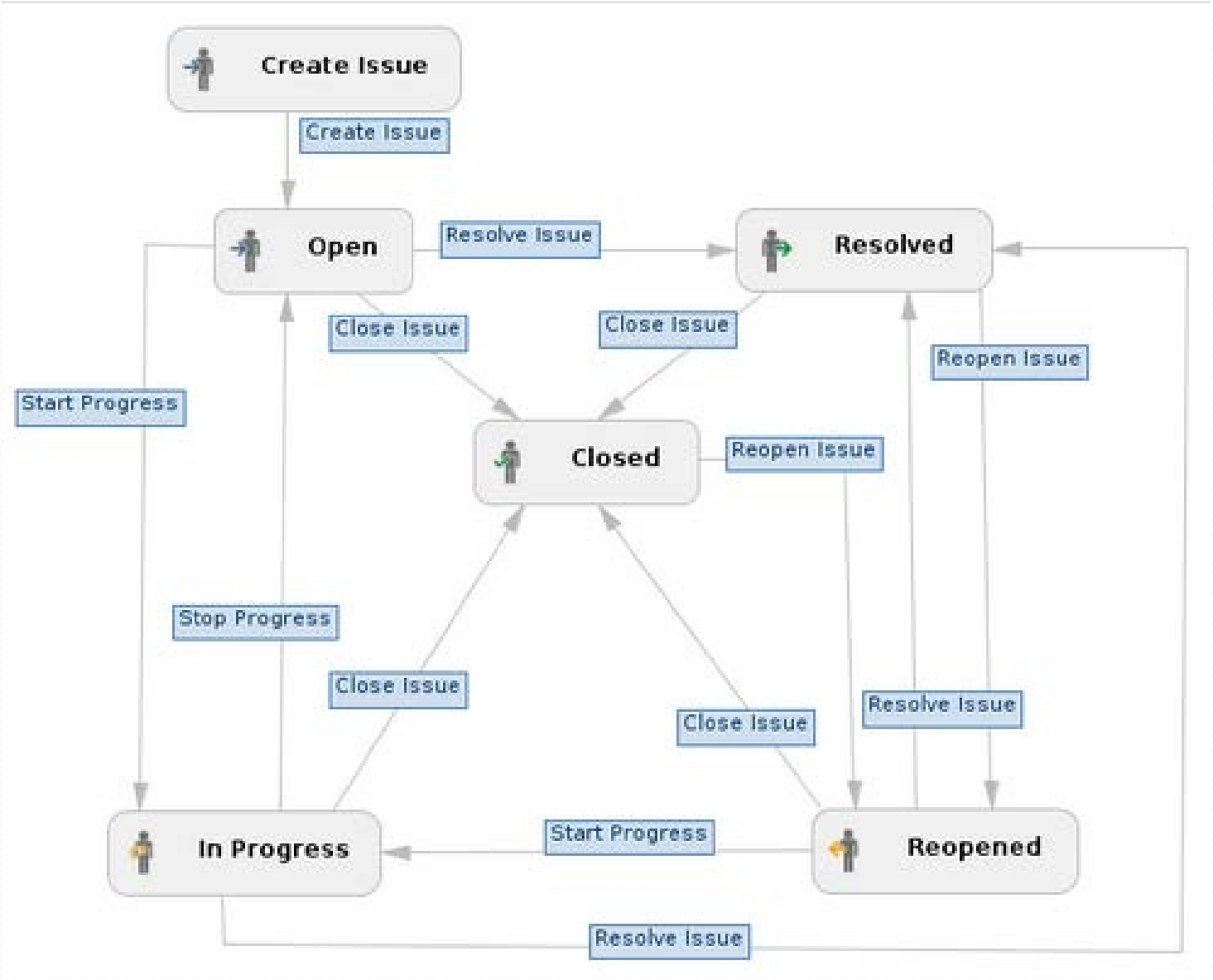
Dates

Created:	13/Jun/12 6:05 PM
Updated:	30/Jul/12 5:38 PM

Agile

View on Board

Cycle de vie d'une anomalie sous JIRA



Déclinaison des anomalies

La découverte d'une anomalie à un niveau du cycle de développement va donner lieu à son analyse et à une éventuelle déclinaison de nouvelles FA aux niveaux inférieurs voire supérieurs.

Micro projet : anomalie au niveau validation (Top-Down)

Découverte d'un résultat obtenu différent du résultat attendu pour le **test de validation** de bout en bout « calcul d'une médiane »

- Ouverture d'une **fiche d'anomalie niveau validation**
- Analyse :
  - le module d'écriture des résultats ne reproduit pas fidèlement la sortie du module de calcul de la médiane => **nouvelle FA au niveau intégration**
  - Le module de calcul de la médiane donne également un résultat erroné => **nouvelle FA au niveau unitaire**

Micro projet : anomalie au niveau unitaire (Bottom-Up)

Découverte d'un résultat obtenu différent du résultat attendu pour le **test unitaire** « calcul d'une médiane » au niveau du module de calcul de la médiane

- Ouverture d'une **fiche d'anomalie niveau unitaire**
- Analyse :
  - Le module de calcul de la médiane donne un résultat erroné.
    - Parcours du cycle de développement (avec éventuellement ouverture de FA au niveau intégration voire validation)

Summary

Introduction

- Why test ?
- Software quality
- Systems development life cycle (reminder)
- Agile software development

Software testing

- Introduction
- Types of Software Testing
- Test Strategy, Test Plan
- Anomaly
- **Coverage matrix (Matrice de couverture)**
- Continuous integration and test frameworks

Matrice de couverture

- C'est un tableau reliant chaque test à une ou plusieurs exigences.
  - Chaque case dans le tableau signifie : si ce test échoue, cette spécification/exigence n'est pas respectée.
  - Chaque exigence doit être couverte **par au moins un test**.
  - Typiquement on aura une matrice de couverture par niveau de développement (tests unitaires, d'intégration, de validation).

	Exigence 1	Exigence 2	Exigence 3
Test 1		X	
Test 2	X		X
Test 3			X

Summary

Introduction

- Why test ?
- Software quality
- Systems development life cycle (reminder)
- Agile software development

Software testing

- Introduction
- Types of Software Testing
- Test Strategy, Test Plan
- Anomaly
- Coverage matrix (Matrice de couverture)
- **Continuous integration and test frameworks**

Intégration continue

*L'**intégration continue** est un ensemble de pratiques utilisées en génie logiciel consistant à vérifier **à chaque modification de code source** que le résultat des modifications **ne produit pas de régression** dans l'application développée.*

Wikipedia

Intégration continue

- Nécessite obligatoirement une **automatisation** de la compilation et des tests.

- Prérequis :
  - Utilisation d'un **logiciel de gestion de version**
  - Obligation des développeurs de faire des commit réguliers
  - Utilisation d'un **framework de test** pour obtenir une standardisation des résultats interprétable par un logiciel
  - Utilisation d'une **application d'intégration continue** pour compiler, exécuter les tests, automatiser le déploiement, présenter les résultats.

Intégration continue

Avantages

- test immédiat des modifications
- notification rapide en cas de code incompatible ou manquant
- les problèmes d'intégration sont détectés et réparés de façon continue, évitant les problèmes de dernière minute
- une version est toujours disponible pour un test, une démonstration ou une distribution

Framework de test

- Framework d'automatisation des tests
  - Python : Pytest, unittest (Pyunit), Nose, …
  - C : Cunit, Check, …
  - C++ : CPPUnit, doctest, …
  - Java : Junit, TestNG, …

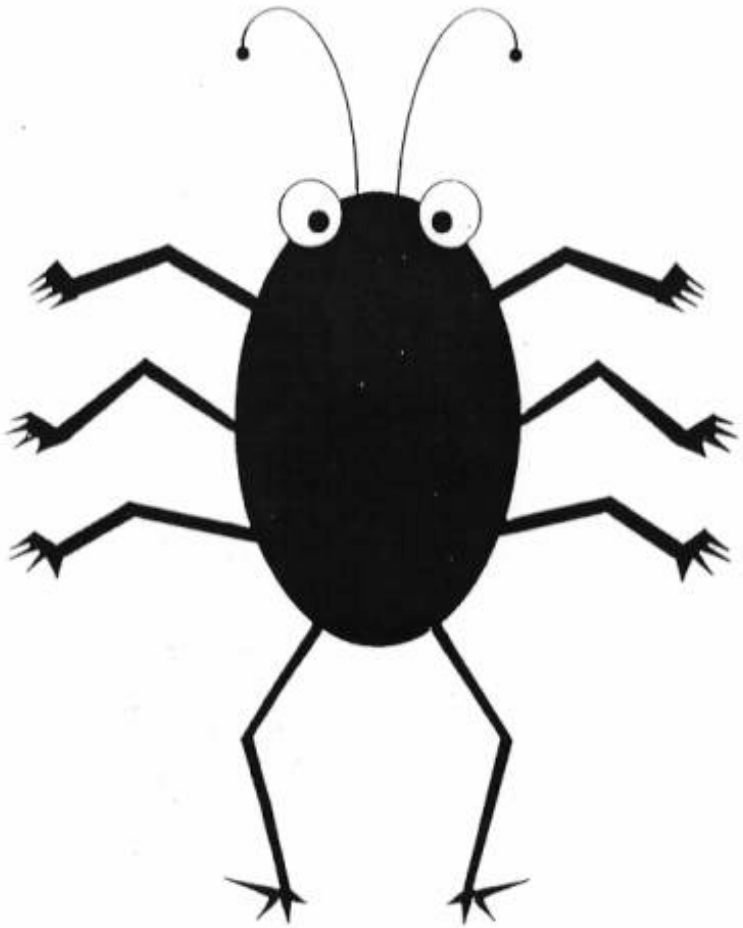
https://en.wikipedia.org/wiki/List_of_unit_testing_frameworks

- Applications de suivi des anomalies et demandes de modification
  - JIRA
  - Redmine
  - Bugzilla
  - Mantis
  - Module « Issues » de GitHub
  - …

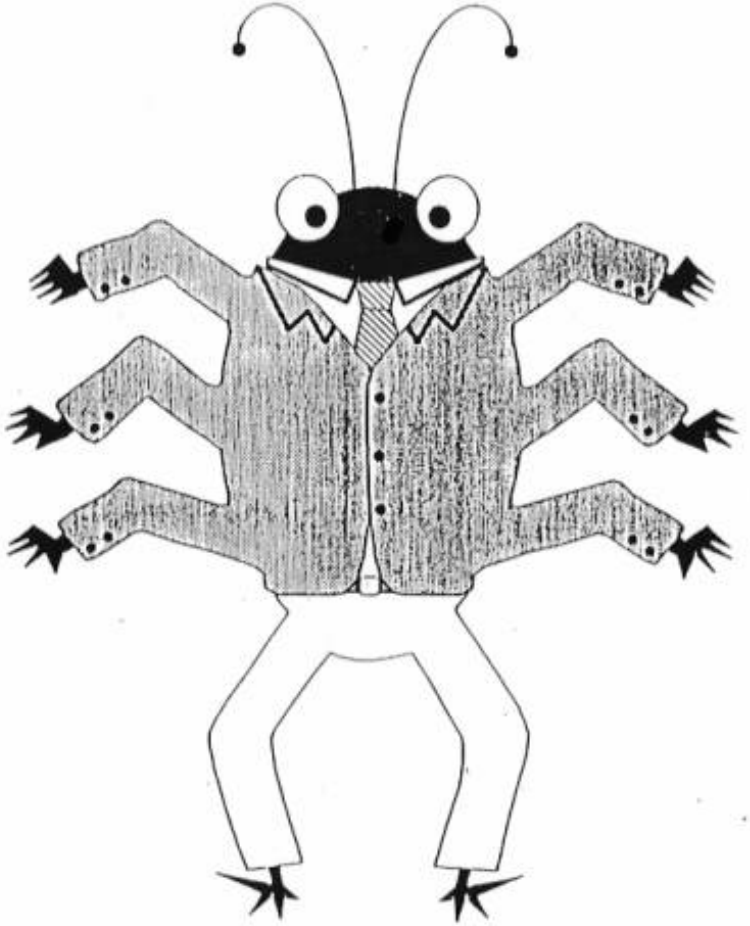
Test Driven Development (TDD)

La logique des tests poussée à l'extrême :

Le ***Test-Driven Development*** (**TDD**) ou en français **développement piloté par les tests** est une technique de développement de logiciel qui préconise **d'écrire les tests unitaires avant d'écrire le code source** d'un logiciel.



BUG



FEATURE


3 BIGGEST
SOFTWARE LIES
#1.
"The program is fully tested & bug-free"
#2.
"We're working on documentation"
#3.
"Of course, we can modify it!"
DOH!
TestMatick