Agenda

- Mercredi 19/11/14 : Test structurel
- Jeudi 20/11/14 : Test fonctionnel
  Test et pratiques agiles
  Autres Types de test

Au menu :

9h00 – 10h30 : Concepts du test fonctionnel

11h00 – 12h30 : Exercice d'introduction

13h30 – 19h00 : Mise en œuvre pratique

Test fonctionnel

Fabrice AMBERT, Fabrice BOUQUET, Fabien PEUREUX,
Jean-Marie GAUTHIER, Alexandre VERNOTTE
prenom.nom@femto-st.fr

ENVOL 2014
La Londe-les-Maures
18-21 novembre 2014

Rappel - techniques

- Deux techniques :
  - **Test structurel**
    Jeu de test sélectionné en s'appuyant sur une analyse du code source du logiciel (*test boite blanche / boite de verre*)
  - **Test fonctionnel**
    Jeu de test sélectionné en s'appuyant sur les spécifications (*test boite noire*)
- En résumé, les méthodes de test dynamique consistent à :
  - Exécuter le programme sur un ensemble fini de données d'entrées
  - Contrôler la correction des valeurs de sortie en fonction de ce qui est attendu

Test structurel (white box)

- Les données de test sont produites à partir d'une analyse du code source

Critères de test :
- tous les chemins,
- toutes les instructions,
- etc…

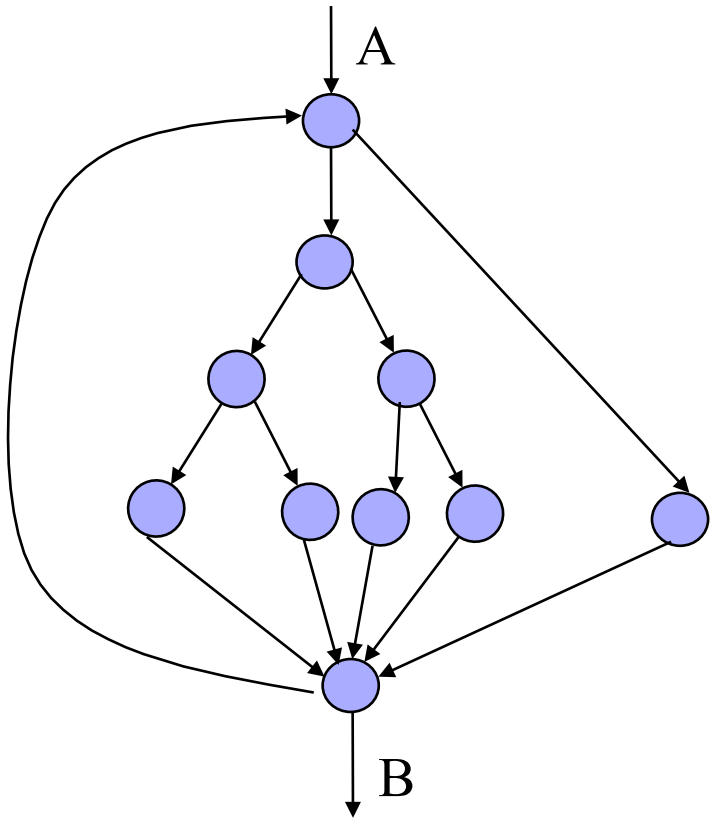
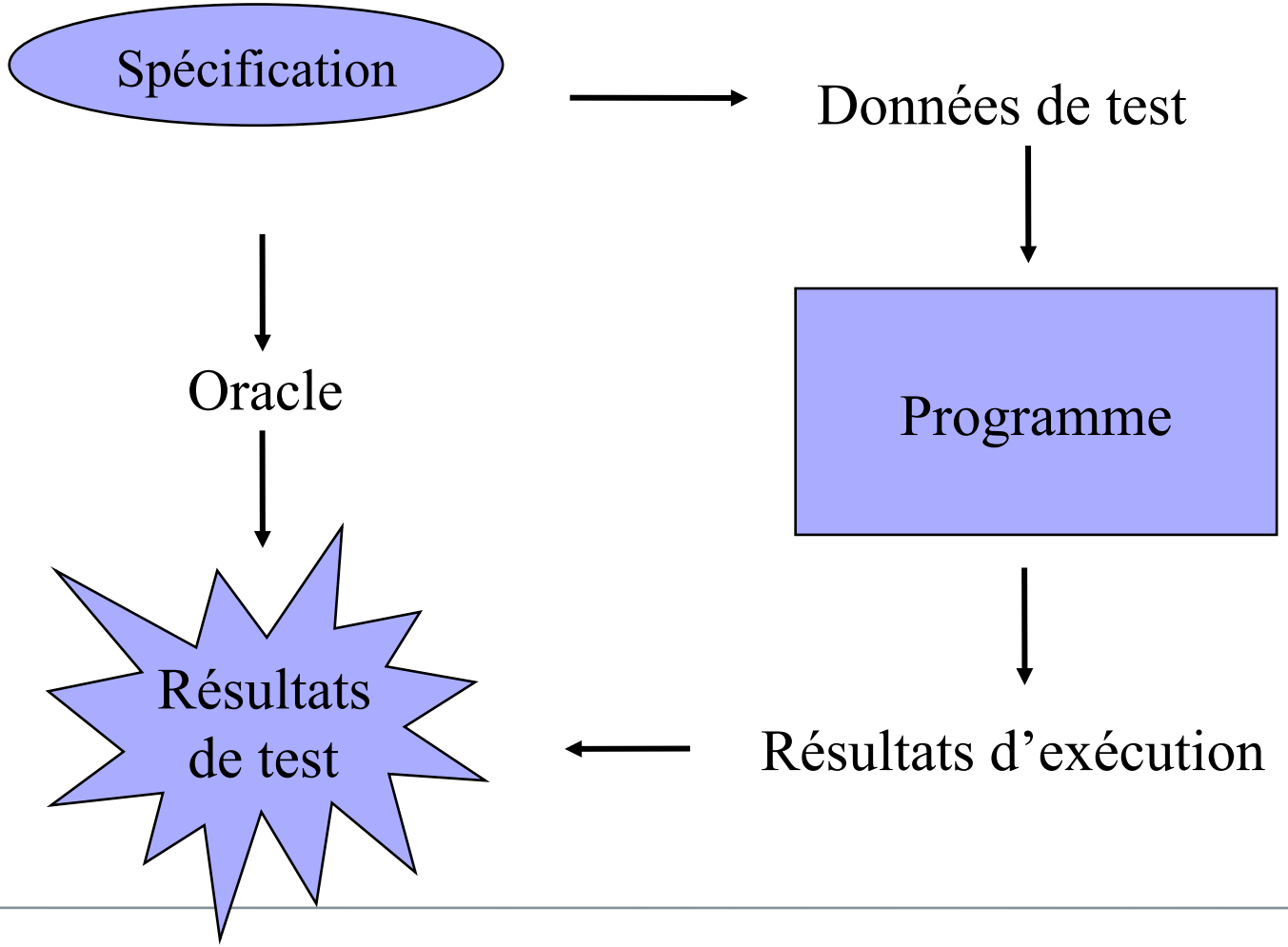


Fig. : Flot de contrôle d'un programme

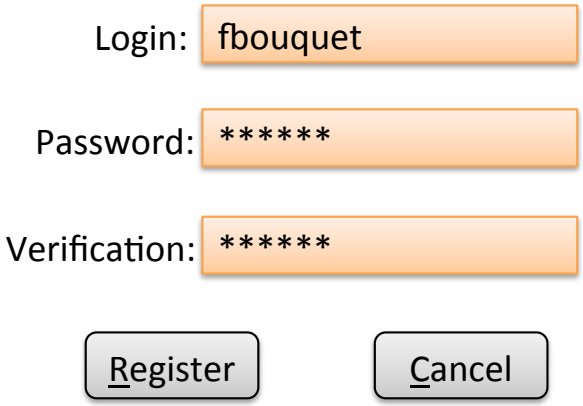
Test fonctionnel (black-box)

- Test de conformité par rapport à la spécification



Test fonctionnel – Exemple (1)

Spécification : "Formulaire d'enregistrement pour un site web sécurisé."

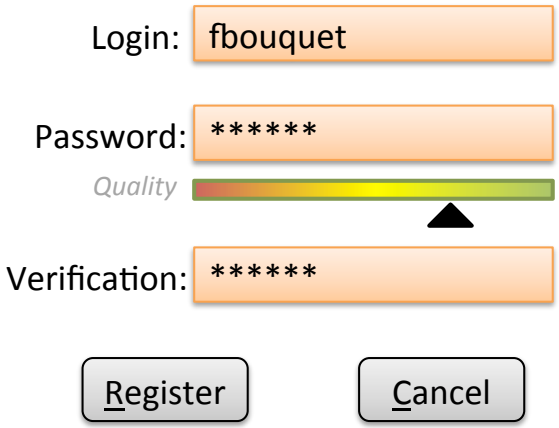


5 Cas : 10 tests

- Login (non) vide (2)
- Login (n') existe (pas) (2)
- Password (non) vide (2)
- Password et Verification (ne) sont (pas) les mêmes (2)
- Protocole http(s) (2)

Test fonctionnel – Exemple (2)

Spécification : "Formulaire d'enregistrement pour un site web sécurisé."

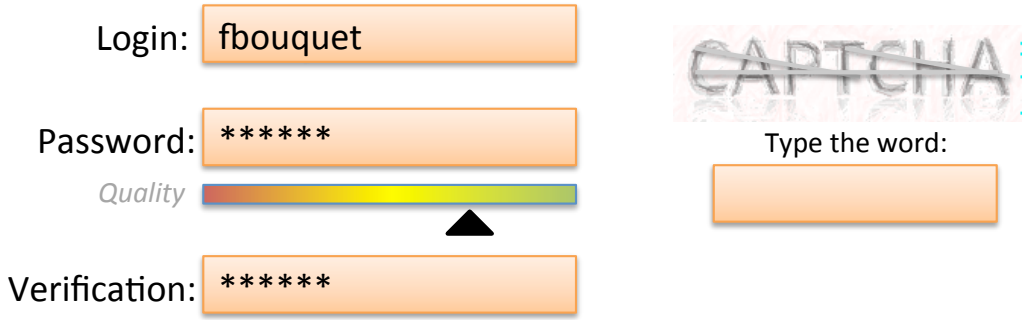


6 Cas : 13 tests

- Login (non) vide (2)
- Login (n') existe (pas) (2)
- Password (non) vide (2)
- Password et Verification (ne) sont (pas) les mêmes (2)
- Protocole http(s) (2)
- Vérifier qualité du password (1 par niveau) : poor, average, good

Test fonctionnel – Exemple (3)

Spécification : "Formulaire d'enregistrement pour un site web sécurisé."



7 Cas : 15 tests

- Login (non) vide (2)
- Login (n') existe (pas) (2)
- Password (non) vide (2)
- Password et Verification (ne) sont (pas) les mêmes (2)
- Protocole http(s) (2)
- Vérifier qualité du password (1 par niveau) : poor, average, good
- Vérifier si enregistrement (non) humain (2)

Le test fonctionnel

- Le test fonctionnel vise à examiner le comportement fonctionnel du logiciel et sa conformité avec la spécification du logiciel.

- Autres objectifs :
  - Guider le développement du logiciel (Acceptance Testing Driven Development)
  - Comprendre, garantir et démontrer l'adéquation du besoin et de la solution proposée
  - Documenter la solution avec des utilisations nominales et/ou particulières
  - Maîtriser la non régression suite aux maintenances correctives et/ou évolutives

- Méthode :
  - Identification des caractéristiques fonctionnelles (exigences)
  - Identification des points de contrôle et d'observation
  - Pour chaque exigence, définition des cas de test (données de test) sous la forme de scénarios d'un point de vue utilisateur

Exigences

- Une exigence est une contrainte exprimée par le client sur telle ou telle caractéristique du futur système

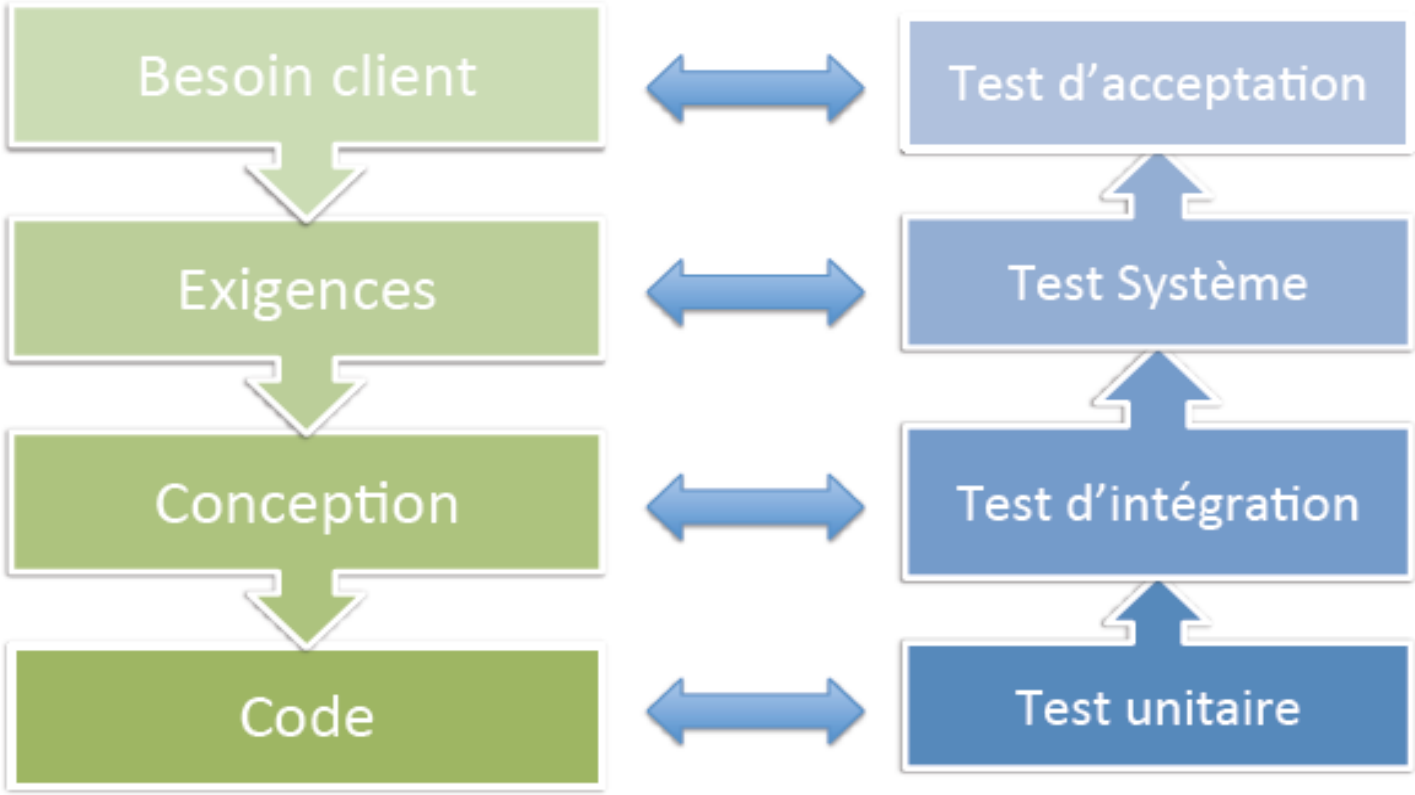
- Dans l'ingénierie système/logiciel, une exigence est généralement la description de ce que le système/logiciel livré doit être capable de faire (exigence fonctionnelle). On trouve aussi :
  - Exigences non fonctionnelles, portant sur la manière dont le système/logiciel doit exécuter ses fonctions
  - Exigences de sécurité,
  - Exigences de performance,
  - Exigences de qualité de service...

- A ces exigences « produit » s'ajoutent également les exigences de type « métier » (qui décrivent le « quoi ») et les exigences processus (qui décrivent le « comment », c'est à dire les contraintes).

Pourquoi des exigences

- La majorité des défauts (entre 40% et 60%) a comme origine le non respect des exigences.

⇒ Les exigences représentent un vecteur central d'amélioration.



Acteurs

- MOA :
  - Maître d'ouvrage : c'est l'initiateur du projet. Il est donc le représentant des utilisateurs finaux à qui l'ouvrage est destiné.

- MOE :
  - Maître d'œuvre : c'est l'entité retenue pour réaliser l'ouvrage. La MOA est client de la MOE à qui elle passe commande d'un produit nécessaire à son activité.

D'une manière générale, la MOE fournit ce produit ; soit il le réalise lui-même, soit il passe commande à un ou plusieurs fournisseurs qui élaborent le produit sous sa direction. Il peut également parfois y avoir des intermédiaires entre MOE et MOA.

Evolution des pratiques

	Capture des exigences	Conception préliminaire	Conception détaillée	Codage codage	Test unitaire	Test d'intégration	Test système
années 60-70	10%			80%		10%	
années 80	20%		60%			20%	
années 90	40%	30%			30%		
années 2000	20%	30%			50%		

Traçabilité des exigences

- **Traçabilité :**

  C'est la possibilité de lire facilement ce qu'il est advenu et ce qu'il est censé advenir de quelque chose.

- **Traçabilité des exigences :**

  C'est le fait de pouvoir à tout instant connaître facilement l'origine et les liens entre les exigences, ainsi qu'entre les exigences et le reste du projet ou le contexte (notamment les besoins utilisateur, réalisation et tests).

Pourquoi la traçabilité des exigences

- Elle aide à répondre aux questions du type :
  - D'où vient une exigence ? (Quel besoin cette exigence couvre-t-elle ?)
  - Pourquoi a-t-on conçu la solution de cette manière et quelles étaient les autres possibilités ?)
  - Cette exigence est-elle nécessaire ?
  - Où met-on en œuvre cette exigence ?
  - Comment interpréter cette exigence ?
  - Quelle décision de conception affecte la mise en œuvre de l'exigence ?
  - Cet élément de conception est-il nécessaire ?
  - La solution réalisée est-elle conforme aux exigences ?
  - Comment testera-t-on cette exigence ?
  - Toutes les exigences sont-elles prises en compte ? (peut se lire : est-ce que le projet est terminé ?)
  - Et si non quel est le pourcentage d'exigences prises en compte ? Et quelles sont les exigences non (encore) prises en compte ?
  - Quel est l'impact du changement d'une exigence ?

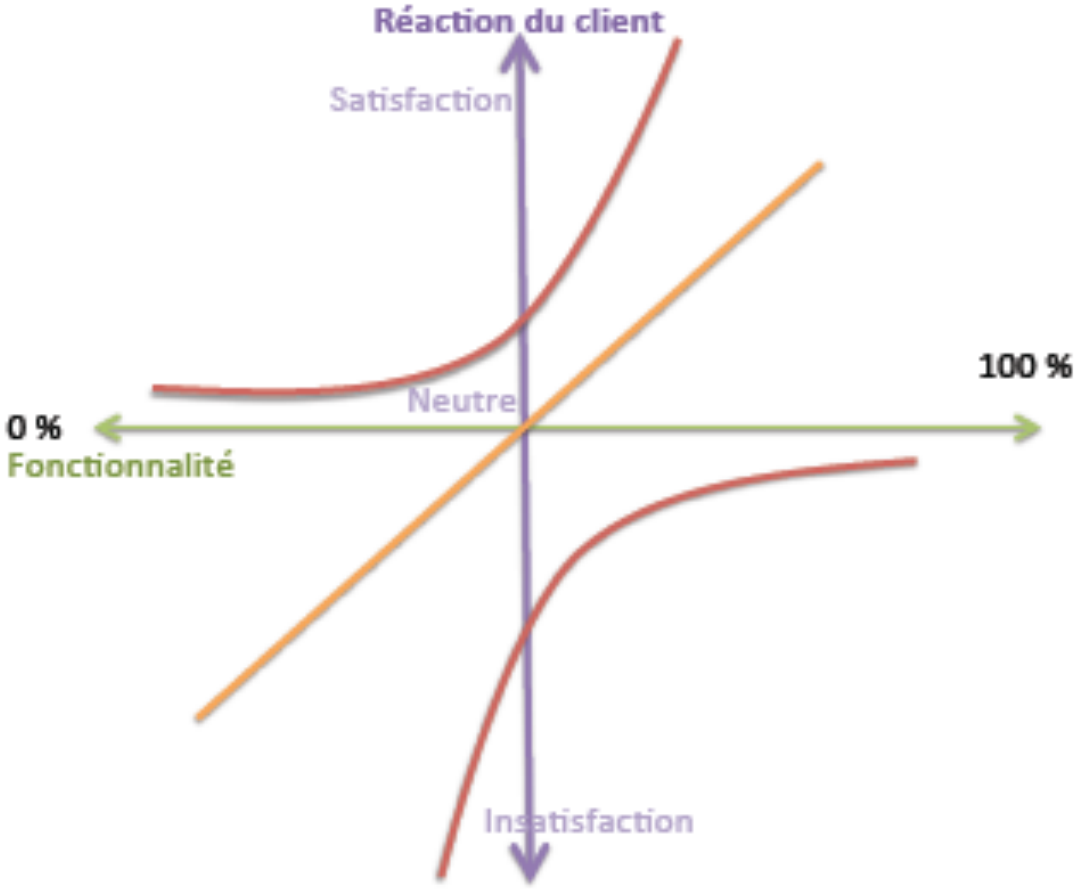
CMMI

Le **Capability Maturity Model Integration** contient 22 domaines de processus avec différent niveau de maturité.

—- Niveau de maturité 2 – Discipliné ————————————————
Gestion de configuration - CM - Configuration Management
Mesure et analyse MA - Measurement and Analysis
Surveillance et contôle de projet - PMC - Project Monitoring and Control
Planification de projet - PP - Project Planning
Assurance-qualité processus et produits - PPQA - Process and Product Quality Assurance
Gestion des exigences - REQM - Requirements Management
Gestion des accords avec les fournisseurs - SAM - Supplier Agreement Management
—- Niveau de maturité 3 – Ajusté ————————————————
Analyse et prise de décision - DAR - Decision Analysis and Resolution
Gestion de projet intégrée - IPM - Integrated Project Management +IPPD
Définition du processus organisationnel - OPD - Organizational Process Definition +IPPD
Focalisation sur le processus organisationnel - OPF - Organizational Process Focus
Formation organisationnelle - OT - Organizational Training
Intégration de produits - PI - Product Integration
Développement des exigences - RD - Requirements Development
Gestion des risques - RSKM - Risk Management
Solution technique - TS - Technical Solution
Validation - VAL - Validation
Vérification - VER - Verification
—- Niveau de maturité 4 – Géré quantitativement ————————————————
Gestion de projet quantitative - QPM - Quantitative Project Management
Performance du processus organisationnel - OPP - Organizational Process Performance
—- Niveau de maturité 5 – En optimisation ————————————————
Analyse causale et résolution - CAR - Causal Analysis and Resolution
Innovation et déploiement organisationnel - OID - Organizational Innovation and Deployment

Priorité des exigences

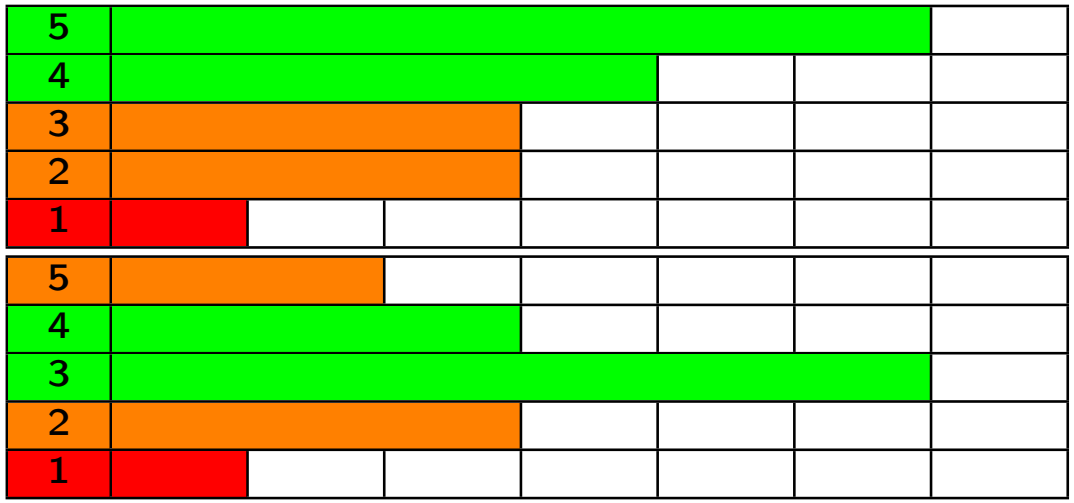
Le modèle de Kano : Satisfaction / Insatisfaction (ne sont pas symétriques)



Priorité des exigences

Formulaire de Kano - questionnaire en deux parties :

1. Perception client face à des caractéristiques
2. Perception client face à l'absence



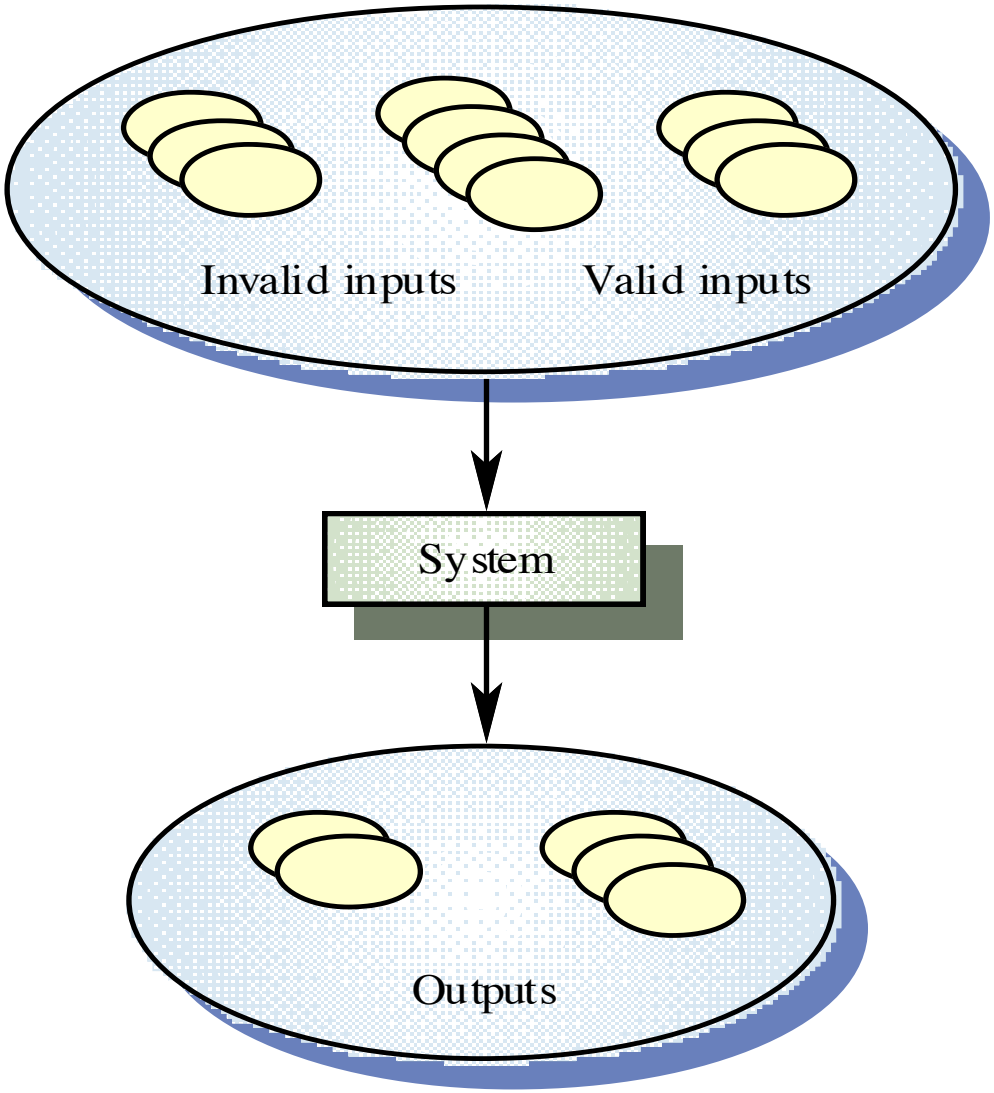
Formulaire Positif

Formulaire Négatif

Test fonctionnel

- Le test fonctionnel vise à examiner le comportement fonctionnel du logiciel et sa conformité avec la spécification du logiciel
  - ⇨ Identification des caractéristiques fonctionnelles (exigences)
  - ⇨ Identification des points de contrôle et d'observation
  - ⇨ Pour chaque exigence, définition des cas de test (données de test) sous la forme de scénarios d'un point de vue utilisateur
- Méthodes de sélection des données de test :
  - Analyse partitionnelle des domaines des données d'entrée
  - Test aux limites
  - Test combinatoire – Algorithmes Pairwise
  - Test aléatoire
  - Génération de tests à partir d'une spécification

Analyse partitionnelle



Une *classe d'équivalence* correspond à un ensemble de données de tests supposées tester le même comportement, c'est-à-dire activer le même défaut.

Analyse partitionnelle - Méthode

- Trois phases :
  - Pour chaque donnée d'entrée, calcul de classes d'équivalence sur les domaines de valeurs,
  - Choix d'un représentant de chaque classe d'équivalence,
  - Composition par produit cartésien sur l'ensemble des données d'entrée pour établir les données de test.

Règles de partitionnement

- Si la valeur appartient à un intervalle, construire :
  - une classe pour les valeurs inférieures,
  - une classe pour les valeurs supérieures,
  - n classes valides.

- Si la donnée est un ensemble de valeurs, construire :
  - une classe avec l'ensemble vide,
  - une classe avec trop de valeurs,
  - n classes valides.

- Si la donnée est une obligation ou une contrainte (forme, sens, syntaxe), construire :
  - une classe avec la contrainte respectée,
  - une classe avec la contrainte non-respectée

Test aux limites

- Principe : on s'intéresse aux bornes des intervalles partitionnant les domaines des variables d'entrées :
  - pour chaque intervalle, on garde les 2 valeurs correspondant aux 2 limites, et les n valeurs correspondant aux valeurs des limites ± le plus petit delta possible :

    $n \in 3 .. 15 \Rightarrow v1 = 3, v2 = 15, v3 = 2, v4 = 4, v5 = 14, v6 = 16$
  - si la variable appartient à un ensemble ordonné de valeurs, on choisit le premier, le second, l'avant dernier et le dernier

    $n \in \{-7, 2, 3, 157, 200\} \Rightarrow v1 = -7, v2 = 2, v3 = 157, v4 = 200$
  - si une *condition d'entrée* spécifie un *nombre de valeurs*, définir les cas de test à partir du nombre *minimum* et *maximum* de valeurs, et des tests pour des nombres de valeurs hors limites invalides.

    Un fichier d'entrée contient 1-255 records, produire un cas de test pour 0, 1, 255 et 256.

Test aux limites - Exemple

- Calcul des limites sur l'exemple du programme de classification des triangles

1, 1, 2	Non triangle
0, 0, 0	Un seul point
4, 0, 3	Une des longueurs est nulle
1, 2, 3.00001	Presque un triangle sans en être un
0.001, 0.001, 0.001	Très petit triangle
99999, 99999, 99999	Très grand triangle
3.00001, 3, 3	Presque équilatéral
2.99999, 3, 4	Presque isocèle
3, 4, 5.00001	Presque droit
3, 4, 5, 6	Quatre données
3	Une seule donnée
	Entrée vide
-3, -3, 5	Entrée négative

Test aux limites – Types de données

- Les données d'entrée ne sont pas seulement des valeurs numériques : booléen, image, son, …
- Ces catégories peuvent, en général, se prêter à une analyse partitionnelle et à l'examen des conditions aux limites :
  - True / False
  - Fichier plein / Fichier vide
  - Trame pleine / Trame vide
  - Nuances de couleur
  - Plus grand / plus petit
  - ….

Analyse partitionnelle et test aux limites – synthèse

- L' analyse partitionnelle est une méthode qui vise à diminuer le nombre de cas de test par calcul de classes d'équivalence

  ➔ importance du choix de classes d'équivalence : risque de ne pas révéler un défaut
- Le choix de conditions d'entrée aux limites est une heuristique solide de choix de données d'entrée au sein des classes d' équivalence

  ➔ cette heuristique n'est utilisable qu'en présence de relation d'ordre sur la donnée d'entrée considérée.
- Le test aux limites produit à la fois des cas de test nominaux (dans l'intervalle) et de robustesse (hors intervalle)

Test aux limites - Evaluation

- Méthode de test fonctionnel très productive :

  ➔ le comportement du programme aux valeurs limites est souvent pas ou insuffisamment examiné

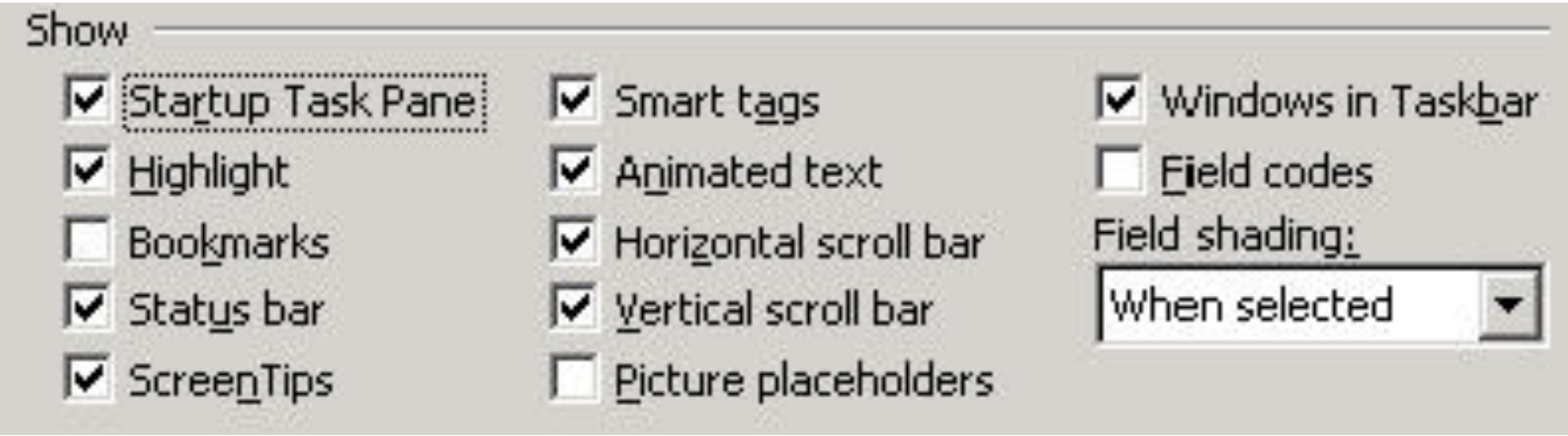
- Couvre l'ensemble des types de test

- Inconvénient : caractère parfois intuitif ou subjectif de la notion de limite

  ⇒ Difficulté pour caractériser la couverture de test.

Méthode pour le test combinatoire

- Les combinaisons de valeurs de domaines d'entrée donne lieu à explosion combinatoire
- Exemple : Options d'une boite de dialogue MS Word



➔ $2^{12} * 3$ (nombre d'item dans le menu déroulant) = 12 288

Test combinatoire : Pair-wise

- Tester un fragment des combinaisons de valeurs qui garantissent que chaque combinaison de 2 variables est testé
- Exemple : 4 variables avec 3 valeurs possibles chacune

OS	Réseau	Imprimante	Application
XP	IP	HP35	Word
Linux	Wifi	Canon900	Excel
Mac X	Bluetooth	Canon-EX	Pwpoint

Toutes les combinaisons : 81

Toutes les paires : 9

Pairwise – Exemple

- 9 cas de test : chaque combinaison de 2 valeurs est testée

	OS	Réseau	Imprimante	Application
Cas 1	XP	ATM	Canon-EX	Pwpoint
Cas 2	Mac X	IP	HP35	Pwpoint
Cas 3	Mac X	Wifi	Canon-EX	Word
Cas 4	XP	IP	Canon900	Word
Cas 5	XP	Wifi	HP35	Excel
Cas 6	Linux	ATM	HP35	Word
Cas 7	Linux	IP	Canon-EX	Excel
Cas 8	Mac X	ATM	Canon900	Excel
Cas 9	Linux	Wifi	Canon900	Pwpoint

L'idée sous-jacente : la majorité des fautes sont détectées par des combinaisons de 2 valeurs de variables

Test combinatoire

- L'approche Pair-wise se décline avec des triplets, des quadruplets, …. mais le nombre de tests augmente très vite

- Une dizaine d'outils permette de calculer les combinaisons en Pairwise (ou n-valeurs) :
  - http://www.pairwise.org/default.html
  - Prise en charge des exclusions entre valeurs des domaines et des combinaison déjà testée

- Problème du Pair-wise :
  - le choix de la combinaison de valeurs n'est peut-être pas celle qui détecte le bug …
  - Le résultat attendu de chaque test doit être fourni manuellement

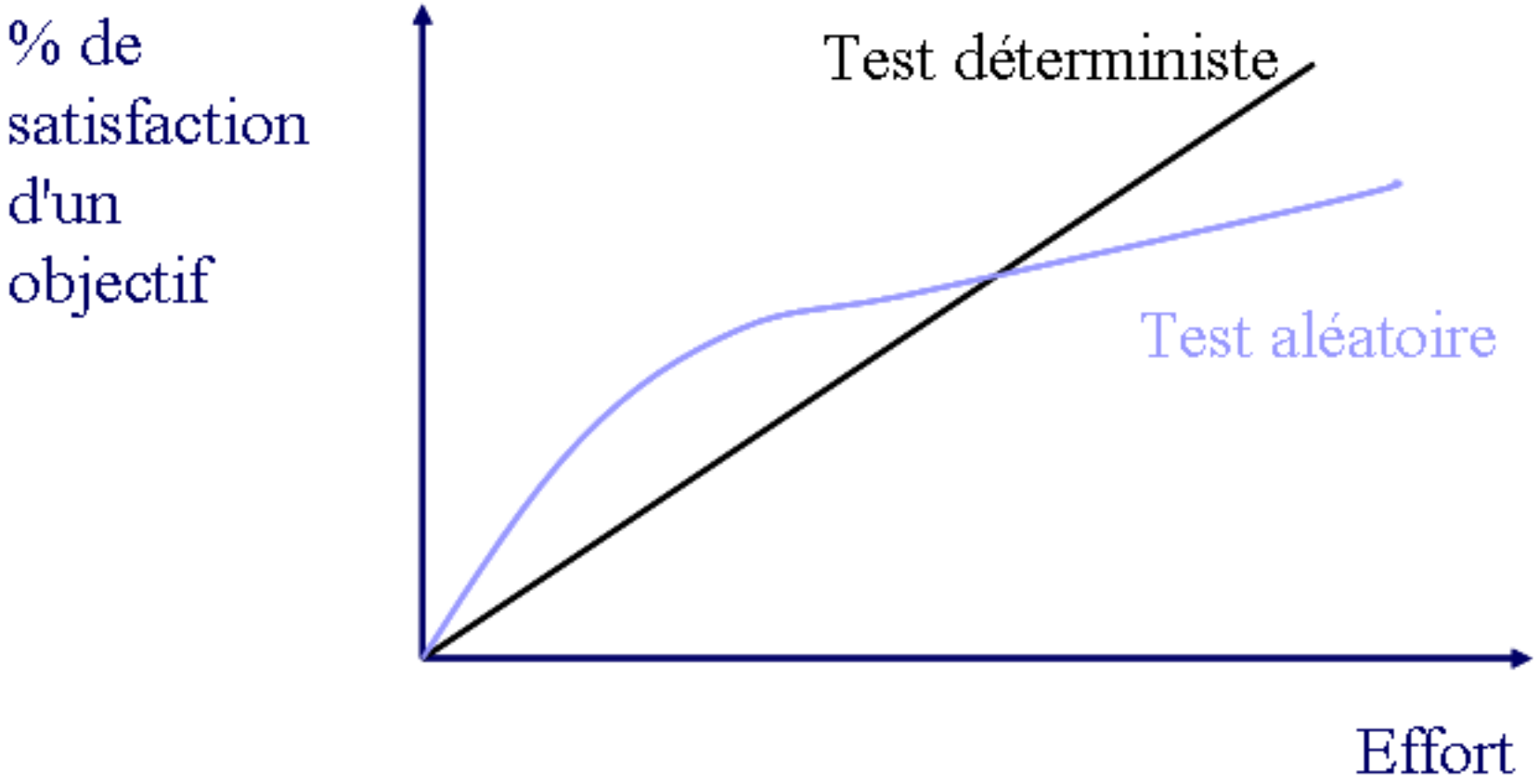
Test aléatoire ou statistique

- Principe : utilisation d'une fonction de calcul pour sélectionner les données de test :
  - fonction aléatoire : choix aléatoire dans le domaine de la donnée d'entrée,
  - utilisation d'une loi statistique sur le domaine.
- Exemples :
  - Echantillonnage de 5 en 5 pour une donnée d'entrée représentant une distance,
  - Utilisation d'une loi de Gauss pour une donnée représentant la taille des individus,
  - ...

Evaluation du test aléatoire

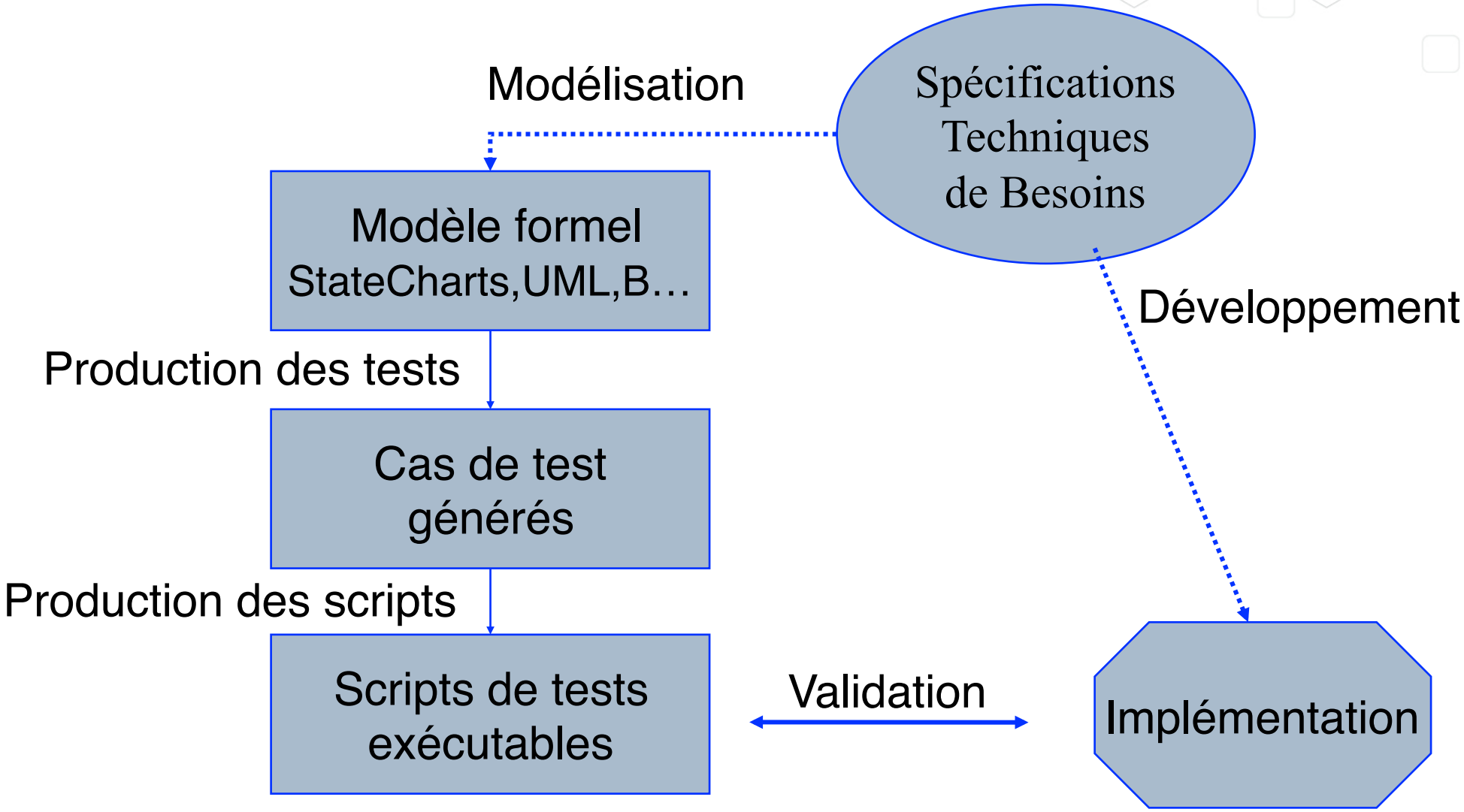
- Intérêts de l'approche statistique :
  - facilement automatisable pour la sélection des cas de test (plus difficile pour le résultat attendu)
  - objectivité des données de test.
- Inconvénients :
  - fonctionnement en aveugle,
  - difficultés de produire des comportements très spécifiques.

Productivité du test aléatoire ou statistique



Les études montrent que le test statistique permet d'atteindre rapidement 50 % de l'objectif de test mais qu'il a tendance à plafonner ensuite.

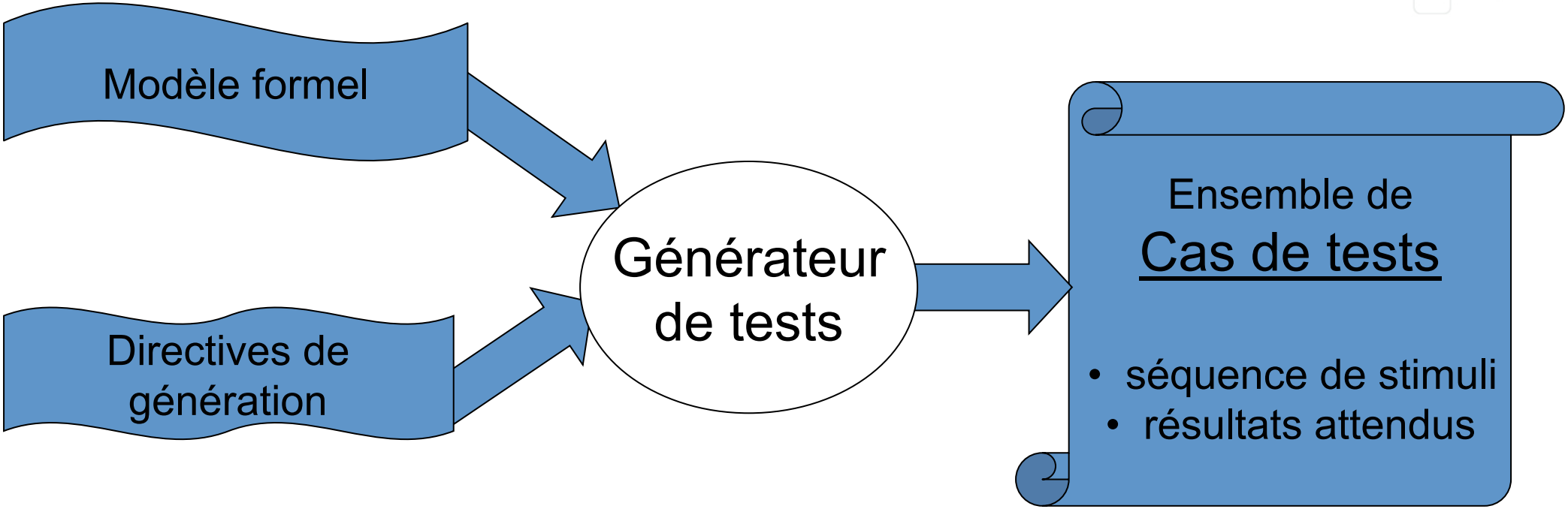
Test à partir de modèles



Langages de modélisation

- De nombreux paradigmes
  - Systèmes de transitions (Etats / Transitions / Evénements)
    - Flow Charts
    - Data Flow Diagrams
    - Diagramme d'état
  - Diagrammes objet & association (Entité-Relation, héritage, …)
    - Diagramme de classe (UML)
    - Entité-Relation
  - Représentation Pré-Post conditions
    - OCL (UML)
    - Machine Abstraite B
- Représentation :
    - Graphique (plus « intuitive »)
    - Textuelle (plus précise)
    - Assertion (embarquée ou proche du code)

Génération de tests à partir de modèle



- Directives de génération (définition de scénarii de test) :
  - Critères de couverture du modèle
  - Critères de sélection sur le modèle

Test et pratiques agiles

Fabrice AMBERT, Fabrice BOUQUET, Fabien PEUREUX,
Jean-Marie GAUTHIER, Alexandre VERNOTTE
prenom.nom@femto-st.fr

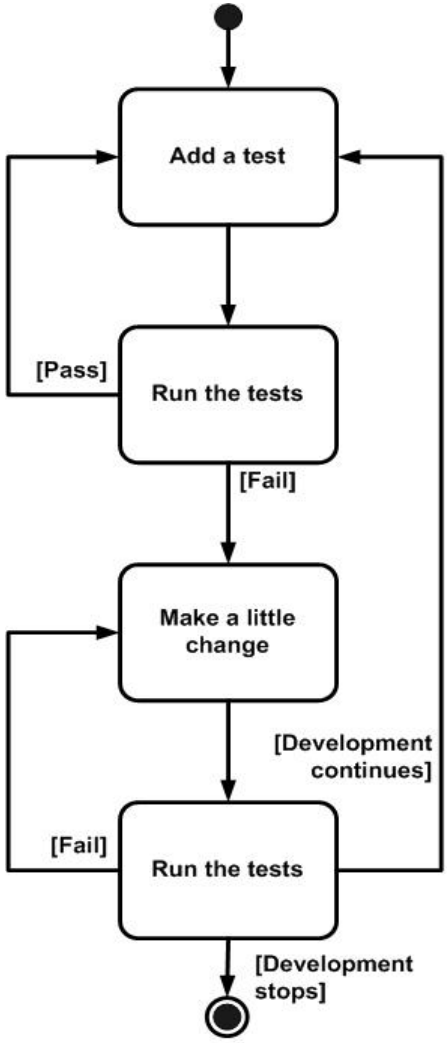
ENVOL 2014
La Londe-les-Maures
18-21 novembre 2014

Pratiques agiles (XP)

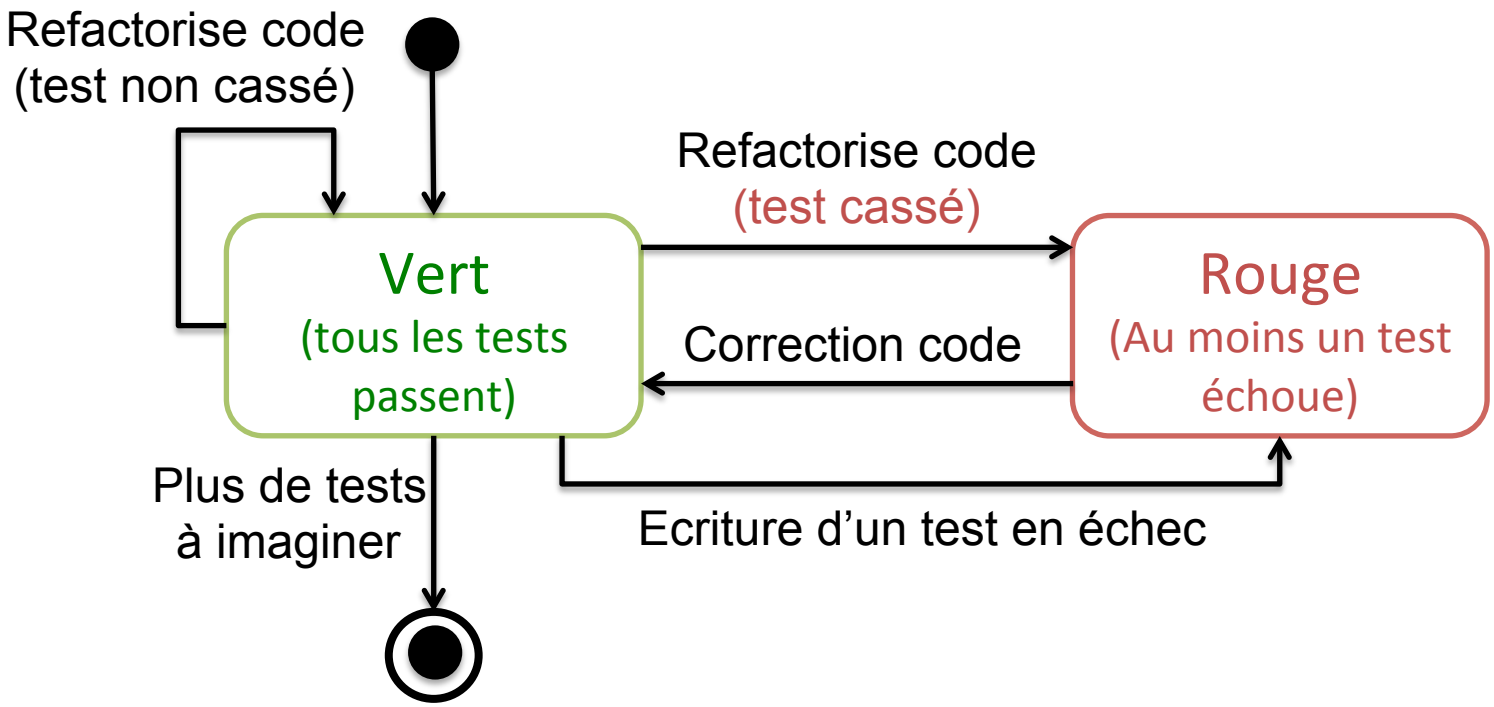
PROJET	CODE	EQUIPE
Livraisons fréquentes : l'équipe vise la mise en production rapide d'une version minimale du logiciel, puis elle fournit ensuite régulièrement de nouvelles livraisons en tenant compte des retours du client.	**Conception simple** : on ne développe rien qui ne soit utile tout de suite.	**Programmation en binômes** : les développeurs travaillent en binômes, ces binômes étant renouvelés fréquemment.
Planification itérative : un plan de développement est préparé au début du projet, puis il est revu et remanié tout au long du développement pour tenir compte de l'expérience acquise par le client et l'équipe de développement.	**Tests unitaires** : les développeurs mettent en place une batterie de tests structurels qui leur permettent de valider leur développement et de faire des modifications sans crainte (garantie de non régression).	**Responsabilité collective du code** : chaque développeur est susceptible de travailler sur n'importe quelle partie de l'application.
Client sur site : le client est intégré à l'équipe de développement pour répondre aux questions des développeurs et définir les tests fonctionnels.	**Restructuration (refactoring)** : le code est en permanence réorganisé pour rester aussi clair et simple que possible.	**Rythme régulier** : l'équipe adopte un rythme de travail qui lui permet de fournir un travail de qualité constant tout au long du projet.
Tests de recette : les testeurs mettent en place des tests fonctionnels qui vérifient que le logiciel répond aux exigences du client. Ces tests permettent une validation du cahier des charges par l'application livrée.	**Intégration continue** : l'intégration des nouveaux développements est faite de façon continue de manière à délivrer de la valeur de façon incrémentale.	**Métaphore** : les développeurs s'appuient sur une description commune du design.
	Règles de codage : les développeurs se plient à des règles de codage strictes définies par l'équipe elle-même.	

Test unitaire

- Acteur : développeur
- Objectifs :
  - Validation structurelle du code
  - Le produit fait les choses « bien »
- Pratique (TDD) :
  - Développer les tests en premier
  - Développer le code correspondant
  - Refactoriser le code / compléter la couverture
- GAINS :
  - Détecter au plus tôt les erreurs
  - Améliorer la testabilité du code
  - Simplifier l'architecture
  - Simplifier le code source
  - Assurer la non-régression

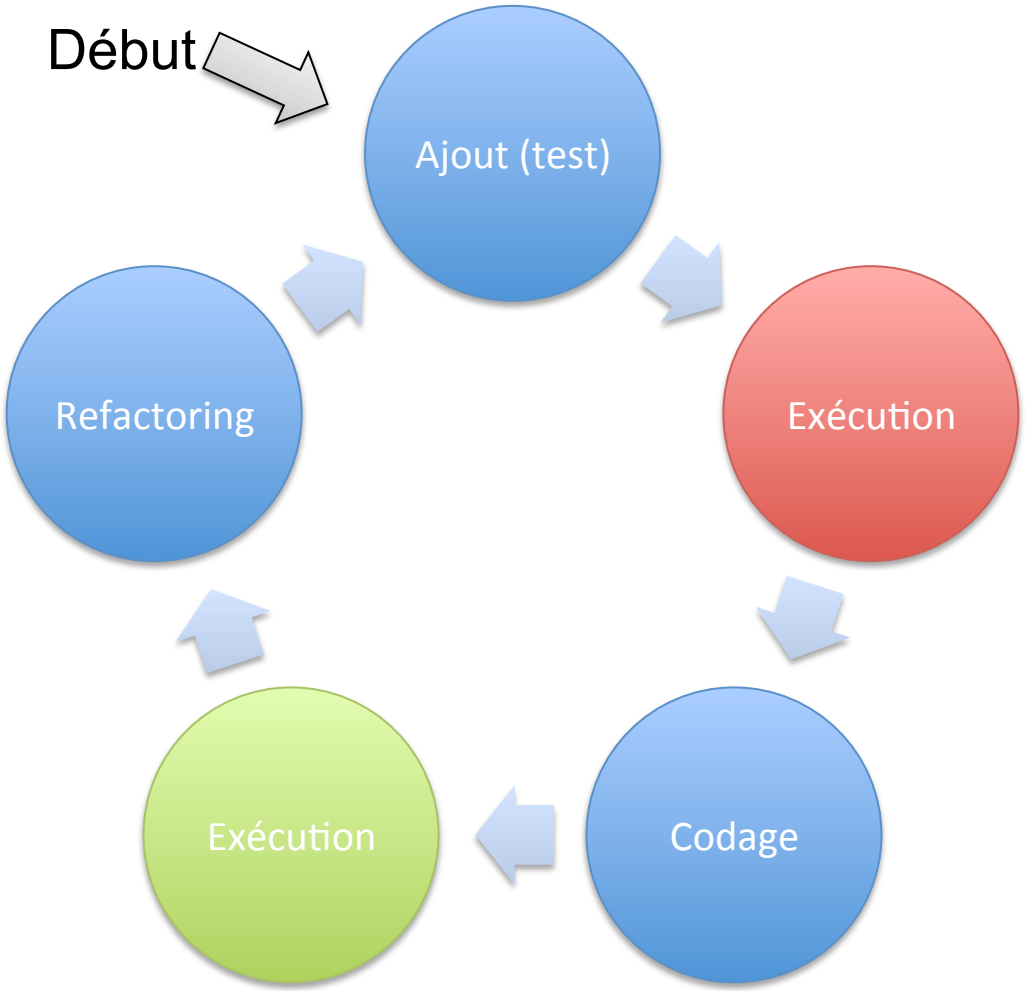


Test unitaire / Non-régression



Filet de sécurité qui avertit immédiatement d'une régression !

Test Driven Development (TDD)



Test d'acceptation

- Acteurs : client (définition) + développeur (automatisation)
- Objectifs :
  - validation fonctionnelle de l'application
  - Le produit fait les « bonnes » choses
- Pratique :
  - Définir textuel de scénarios d'usage par le « métier »
  - Développer le code correspondant par les développeurs
  - Valider par le développeur puis le client
- GAINS :
  - Spécification exécutable
  - Capturer les besoins réels
  - Garantir la conformité vis-à-vis des attentes du client
  - Documentation

Tests automatisés (1)

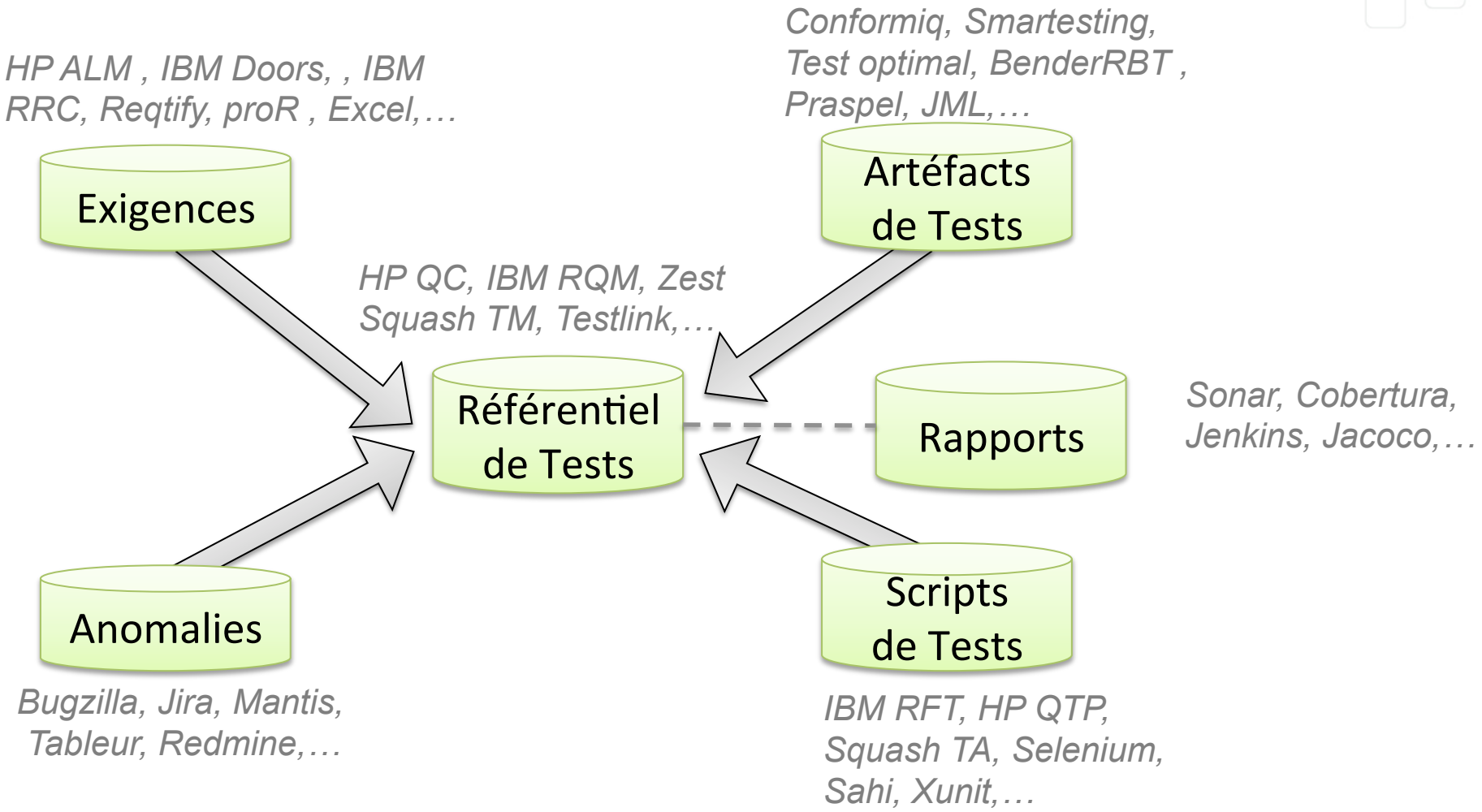
G. Meszaros, S. M. Smith, J. Andrea, The Test Automation Manifesto, In Extreme Programming and Agile Methods - XP/Agile Universe, LNCS 2753 (Sep 2003), pp. 73-81

- Les tests automatisés doivent être :
  - Clairs : faciles à comprendre
  - Concis : écrits aussi simplement que possible
  - Spécifiques : chaque test cible un objectif précis et particulier
  - Suffisants : ils doivent couvrir tous les besoins du système
  - Nécessaires : pas de redondance dans les tests

Tests automatisés (2)

- Robustes : un test doit toujours produire le même résultat
- Indépendants : ils ne dépendent pas de leur ordre d'exécution
- Auto-vérifiables : aucune intervention humaine ne doit être nécessaire
- Répétables : toujours sans intervention humaine
- Maintenables : ils doivent pouvoir être aisément modifiés
- Traçables : on doit pouvoir les (re)jouer au pas à pas

Test - Outillage



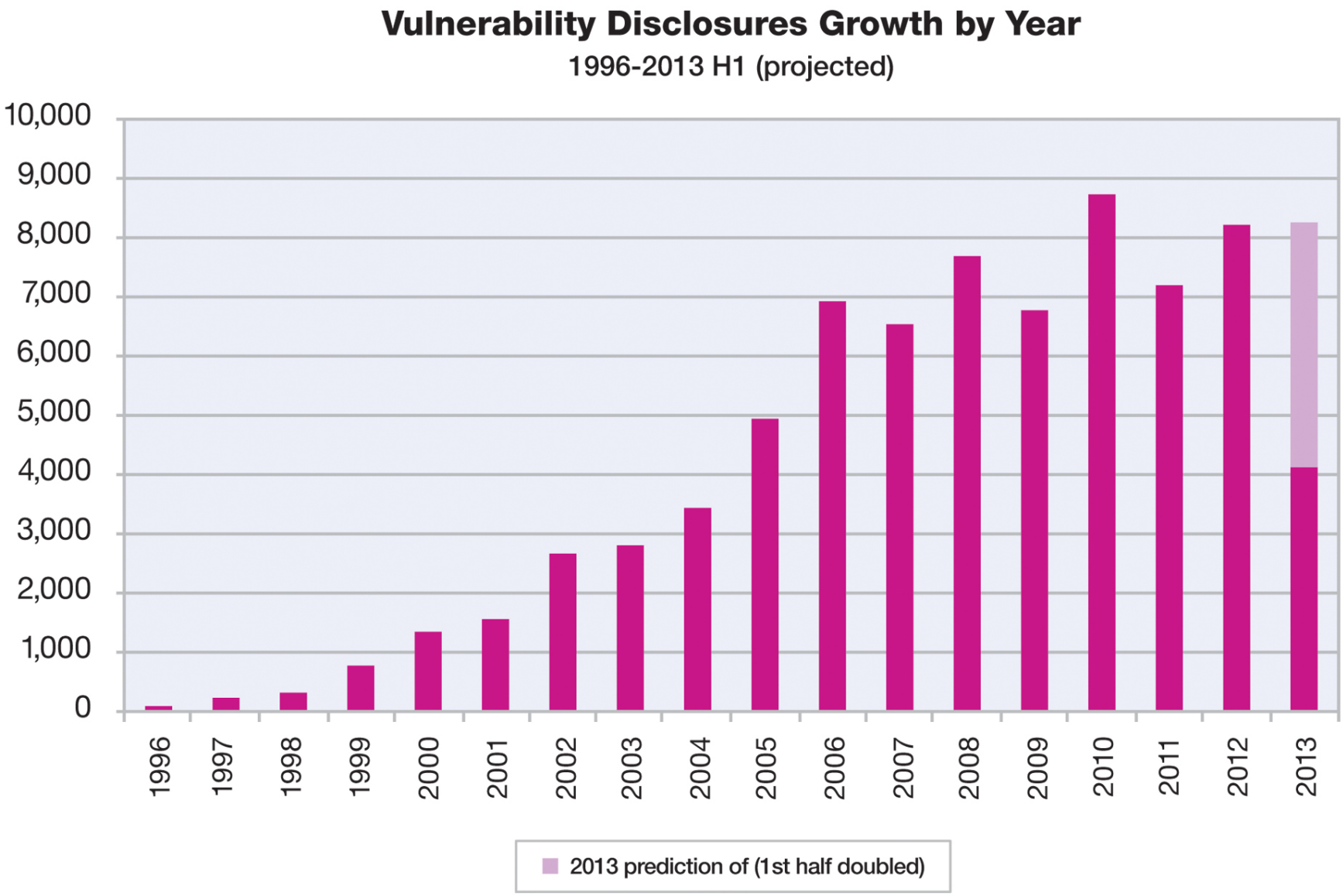
Test non fonctionnel

Fabrice AMBERT, Fabrice BOUQUET, Fabien PEUREUX,
Jean-Marie GAUTHIER, Alexandre VERNOTTE
prenom.nom@femto-st.fr

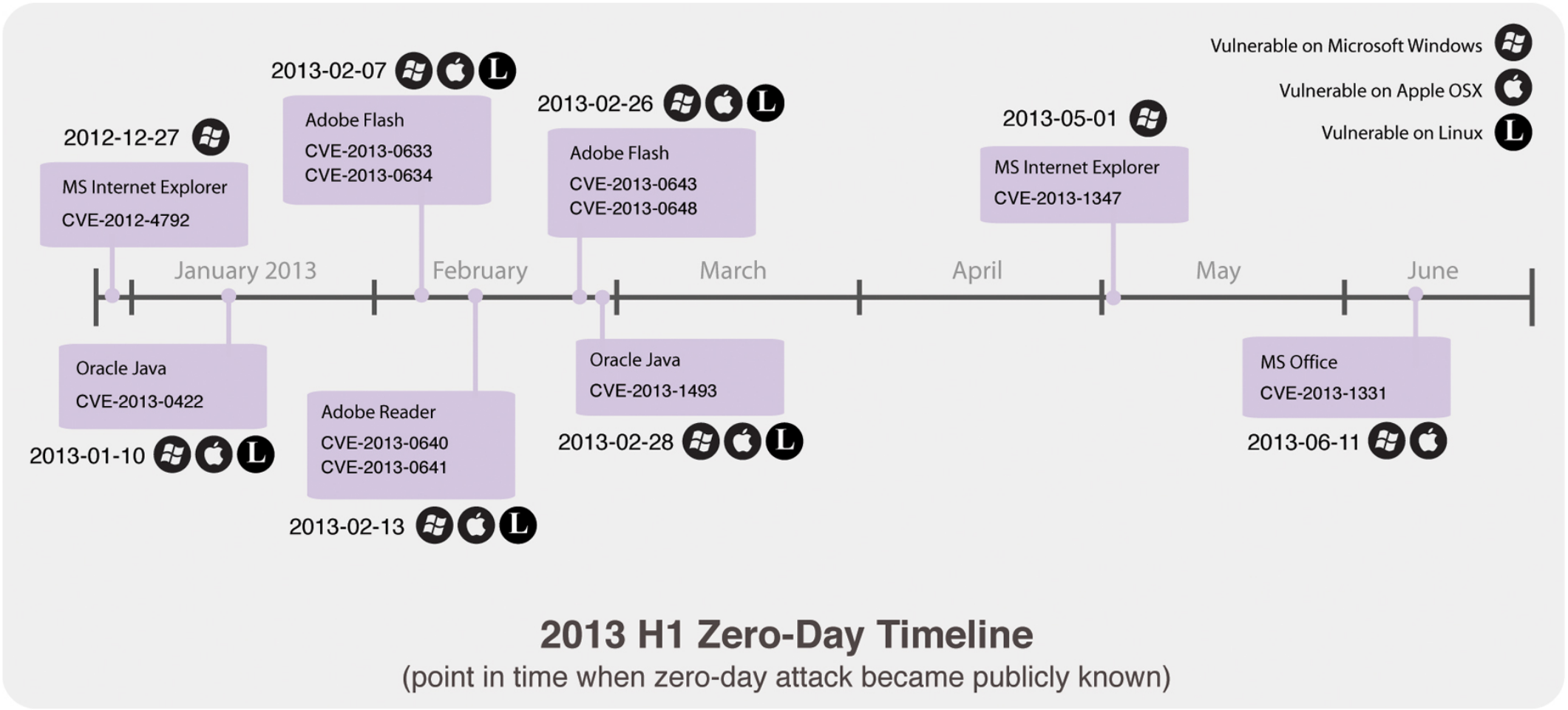
ENVOL 2014
La Londe-les-Maures
18-21 novembre 2014

Plan de la présentation

- Test de sécurité / vulnérabilité
- Test de performances / charges
- Test d'IHM



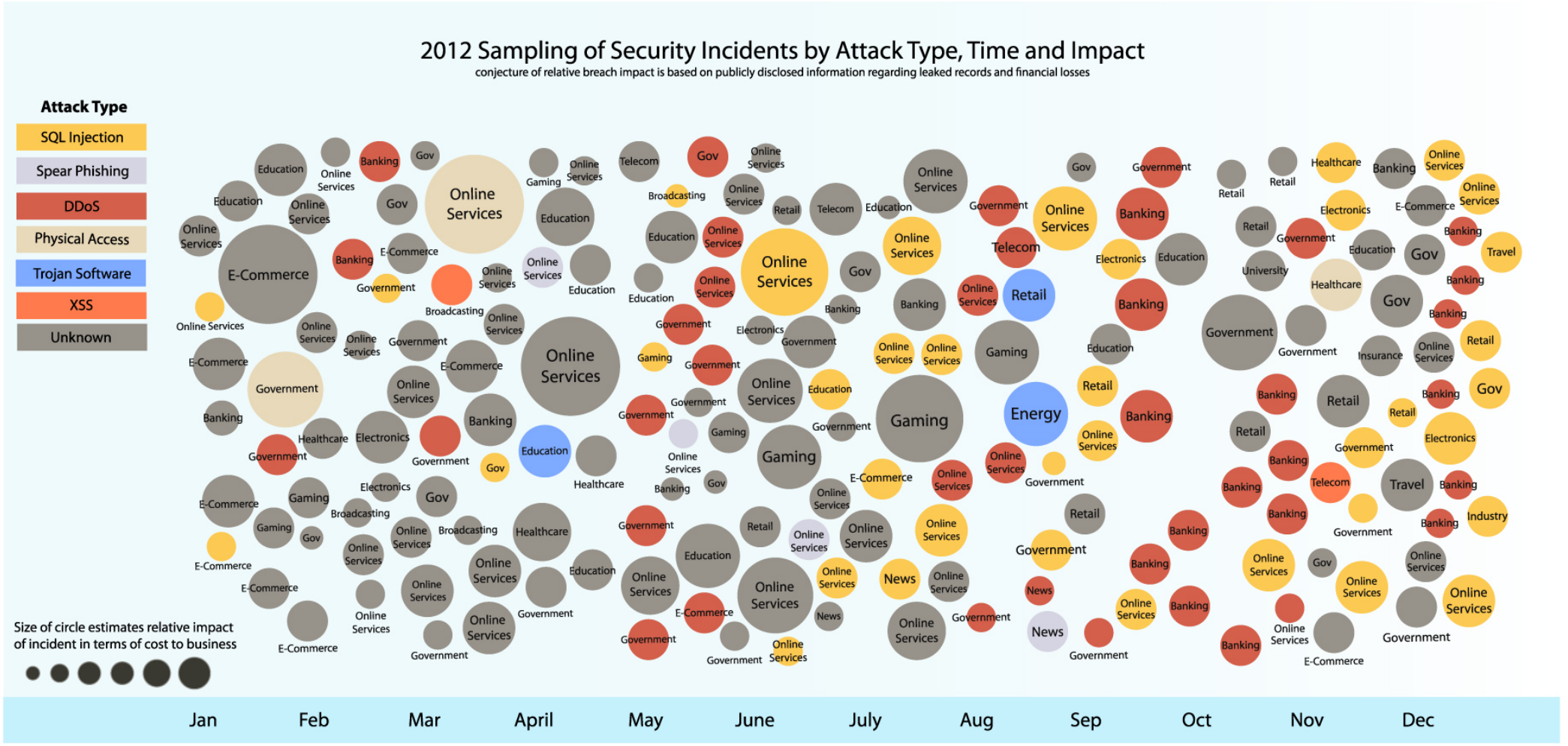
Source: IBM X-Force® Research and Development



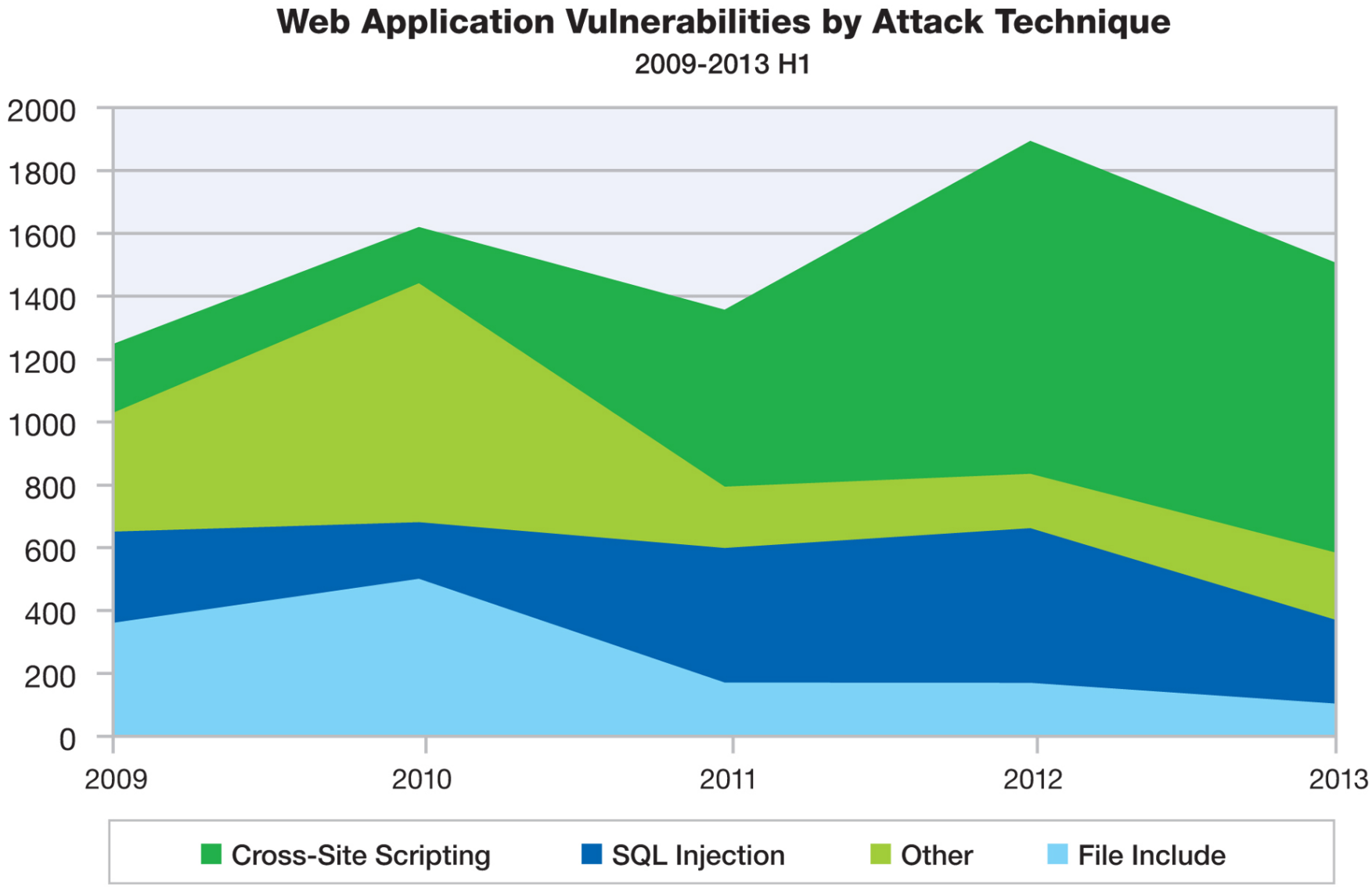
2013 H1 Zero-Day Timeline

(point in time when zero-day attack became publicly known)

Source: IBM X-Force® Research and Development



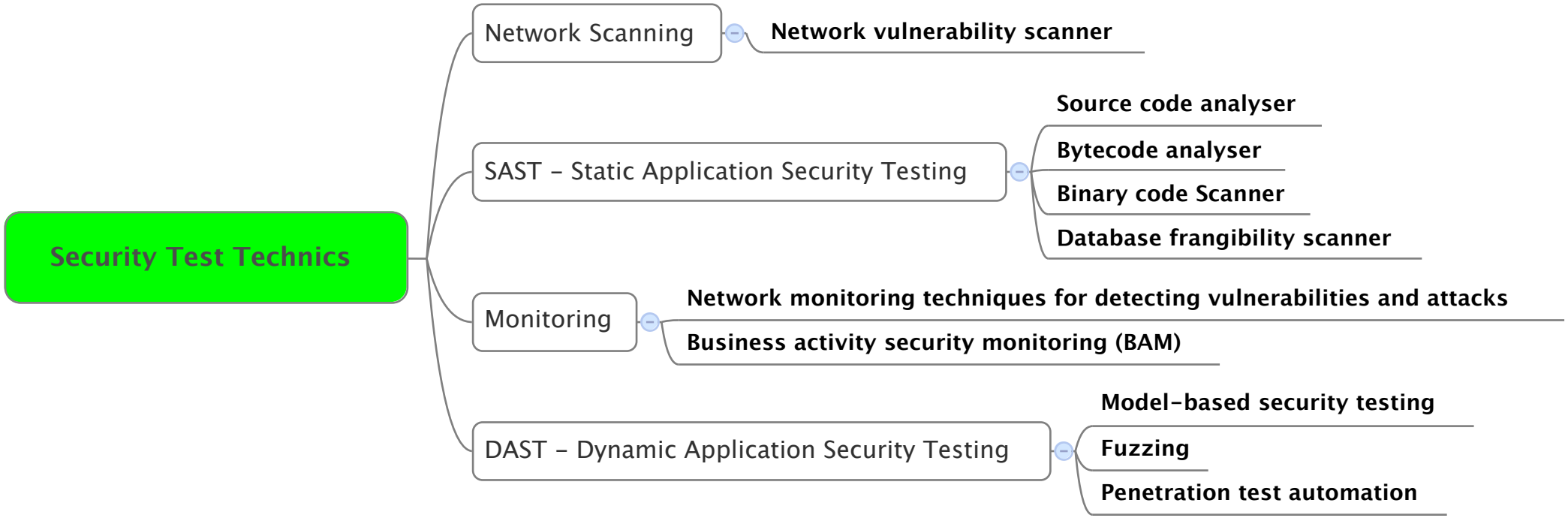
Source: IBM X-Force® Research and Development



Source: IBM X-Force® Research and Development

Test de sécurité

Tester la manière dont le système est protégé contre les accès internes ou externes non autorisés



Test de charge / performance

Tester en simulant un nombre d'utilisateurs(ions) virtuels prédéfinis et valider que le système répond correctement (temps/fonctionnement)

- Test de dégradations des transactions
- Test de stress
- Test de robustesse, d'endurance, de fiabilité
- Test de capacité, test de montée en charge
- Test aux limites
- Tests de Volumétrie des données

http://en.wikipedia.org/wiki/Load_testing

Test d'IHM

Tester en simulant des actions sur l'interface

- Différents univers :
  - OS
  - Langage
  - WEB
- Processus :
  - Enregistrement
  - Adaptation
  - Rejeu

http://en.wikipedia.org/wiki/List_of_GUI_testing_tools

BILAN

Fabrice AMBERT, Fabrice BOUQUET, Fabien PEUREUX,
Jean-Marie GAUTHIER, Alexandre VERNOTTE
prenom.nom@femto-st.fr

ENVOL 2014
La Londe-les-Maures
18-21 novembre 2014

Bilan (1)

- Le test est une activité aujourd'hui incontournable dans la qualité du logiciel (assurance qualité) – promue par l'agilité

- Définition des tests :
  - Plusieurs approches suivant les objectifs
  - Complémentarité des approches (structurelle / fonctionnelle)
  - Plusieurs techniques dédiées

- Validation par le test :
  - Gain de confiance dans le code produit (filet de sécurité face aux régressions)
  - Garantie de la « bonne » couverture du besoin (cahier des charges)
  - Capacité à mieux maîtriser / évaluer / démontrer la qualité du logiciel
  - Dans le contexte agile : expansion du rôle des tests
    - Spécifier le besoin (cas d'utilisation)
    - Guider le développement (TDD, ATDD)
    - Simplifier et minimiser le code développé

Bilan (2)

- Automatisation de l'exécution des tests :
  - Non nécessaire mais décuple le ROI des tests
  - Facilite souvent l'introduction du test auprès des équipes
  - Structuration et cadencement du processus de développement
  - Levier important pour améliorer la qualité du logiciel

- Freins et réticences :
  - Caractère destructif du test
  - Nouveaux langages à maîtriser
  - Difficulté de scripter les cas de test
  - Difficultés d'automatiser l'assignement du verdict
  - Plateformes IC perçues comme des « usines à gaz »

- Axes d'amélioration actuels :
  - Outillage croissant (langages visés, simplicité d'utilisation, …)
  - Technologie connexe pour la traçabilité (exigences, bugtracker, planificateur,…)
  - Solutions avancées d'automatisation (génération des cas de test / MBT)

Après, vous ne pourrez plus dire… (1)

Les 24 réponses les plus fréquentes à un sondage réalisé auprès de développeurs pour expliquer / justifier un problème rencontré avec leur développement :

24. "Il marche bien sur mon ordinateur.""

23. "Vous vous connectez comme quel utilisateur ?"

22. "C'est une fonctionnalité."

21. "C'est ce qu'il avait été demandé."

20. "C'est bizarre, étrange (une explosion solaire ?)."

19. "Il n'avait jamais fait cela avant."

18. "Ca marchait hier."

17. "Comment cela est-il possible ?"

16. "Ca doit être un problème matériel (hardware)."

15. "Vous avez fait n'importe quoi, ou vous ne savez pas vous en servir ?"

14. "Il y a un problème dans vos données."

13. "Je n'ai pas touché à ce module dernièrement !"

Après, vous ne pourrez plus dire…(2)

Les 24 réponses les plus fréquentes à un sondage réalisé auprès de développeurs pour expliquer / justifier un problème rencontré avec leur développement :

12. "Vous n'avez pas la bonne version."

11. "Ca doit venir d'une coïncidence (un morceau de dll resté en mémoire... redémarrer)"

10. "Je ne peux pas tout tester !"

9. "Ceci ne peut pas correspondre à mon/au code source."

8. "Ca marche, mais ça n'a pas été testé."

7. "Quelqu'un a changé mon code."

6. "Avez-vous vérifié qu'il n'y avait pas de virus dans le système ?"

5. "Bien que cela ne fonctionne pas, cela correspond-il à votre attente ?"

4. "Vous ne pouvez pas utiliser cette version sur votre système."

3. "Pourquoi voulez-vous faire cela de cette façon ?"

2. "Où (en) étiez-vous quand le programme a planté ?"

1. "Je pensais avoir corrigé ce problème."

Programme

- Mercredi 19/11/14 : test structurel
  - Exercice de couverture
  - Java : Junit, Mockito, Jacoco
  - C++ : CPPUnit, GoogleMock, Gcov
- Jeudi 20/11/14 : test fonctionnel
  - ALL : Squash TM, Selenium
  - Java : Concordion
  - C++ : Cucumber

Merci pour votre attention…