Tests d'intégration

Bibliographie

- Programmation par les tests, ESIREM, Céline ROUDET
- Comment écrire du code testable, Conférence Agile France 2010, Florence CHABANOIS
- Reflexion on Software Quality and Maintenance, Alexandre Bergel, Chili
- An Introduction to Test-Driven Development (TDD), Craig Murphy
- Tests et Validation du logiciel, http://home.nordnet.fr/~ericleleu
- Test à partir de modèles : pistes pour le test unitaire de composant, le test d'intégration et le test système, Yves Letraon
- Les tests en orienté objet, J. Paul Gibson http://www-inf.int-evry.fr/cours/CSC4002/Documents
- Mocks and Stubs, Martin Fowler
- Introduction au test du logiciel, Premiers pas avec JUnit, Mirabelle Nebut
- Écrire du code testable Par Aurélien Bompard

Tests d'intégration

✓Différents modules d'une application peuvent fonctionner unitairement, leur intégration, entre eux ou avec des services tiers, peut engendrer des dysfonctionnements.

✓Il est souvent impossible de réaliser les tests unitaires dans l'environnement cible avec la totalité des modules à disposition.

➡Les tests d'intégration ont pour objectif de créer une version complète et cohérente du logiciel (avec l'intégralité des modules testés unitairement) et de garantir sa bonne exécution dans l'environnement cible.

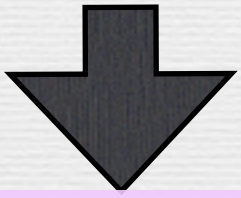
Tests d'intégration

Objectif

Vérifier les interactions entre composants unitaires

Difficultés principales de l'intégration

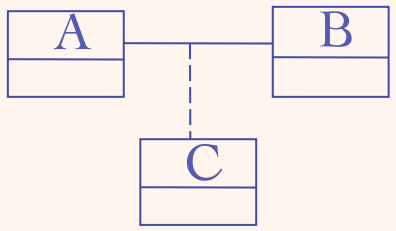
- Interfaces floues
- Implantation non conforme à la spécification
- Réutilisation de composants



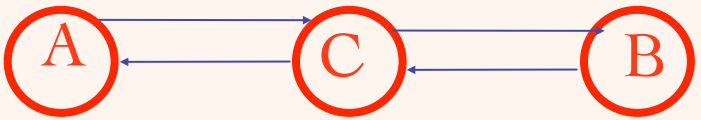
1) modéliser la structure de dépendances entre chaque composant et son environnement (graphe de dépendance des tests)

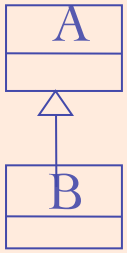
2) Choisir un ordre pour intégrer (assembler)

Graphe de dépendance : construction

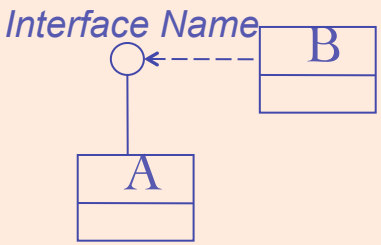


association class

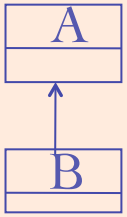




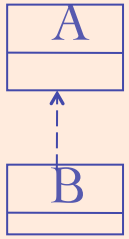
inheritance



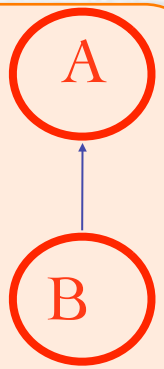
Interfaces

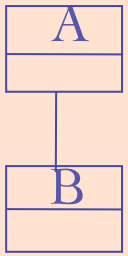


navigability

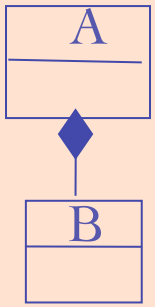


dependency

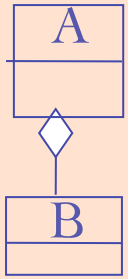




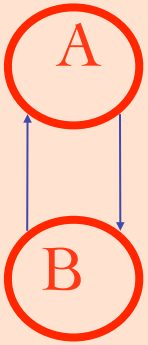
association



composition



aggregation



Compilateur GNU pour Eiffel :

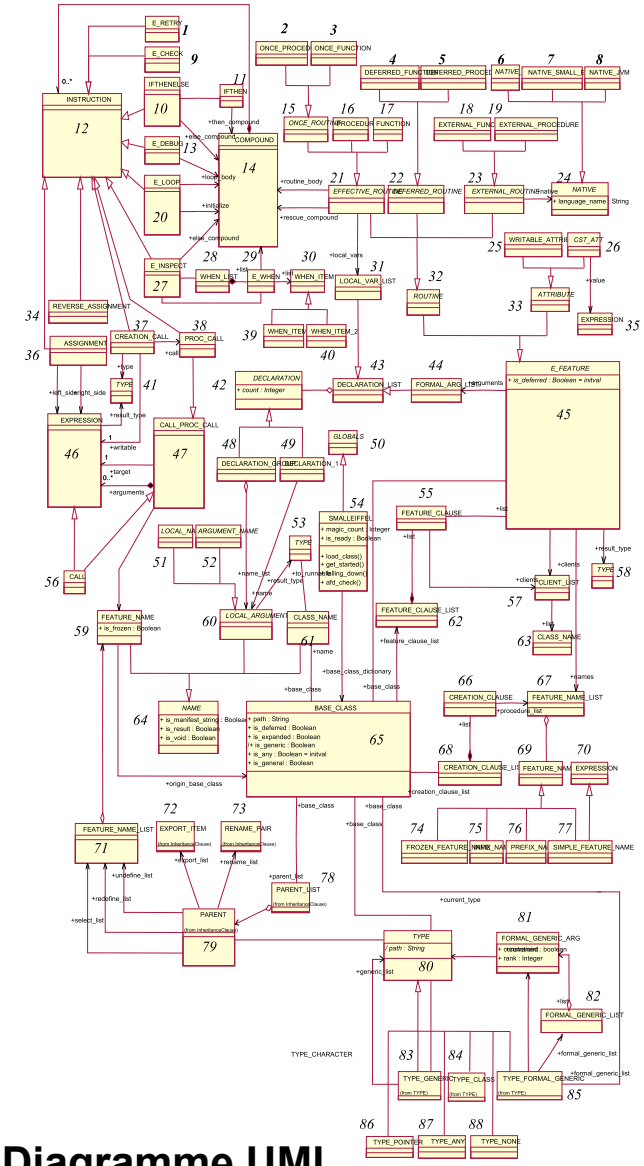
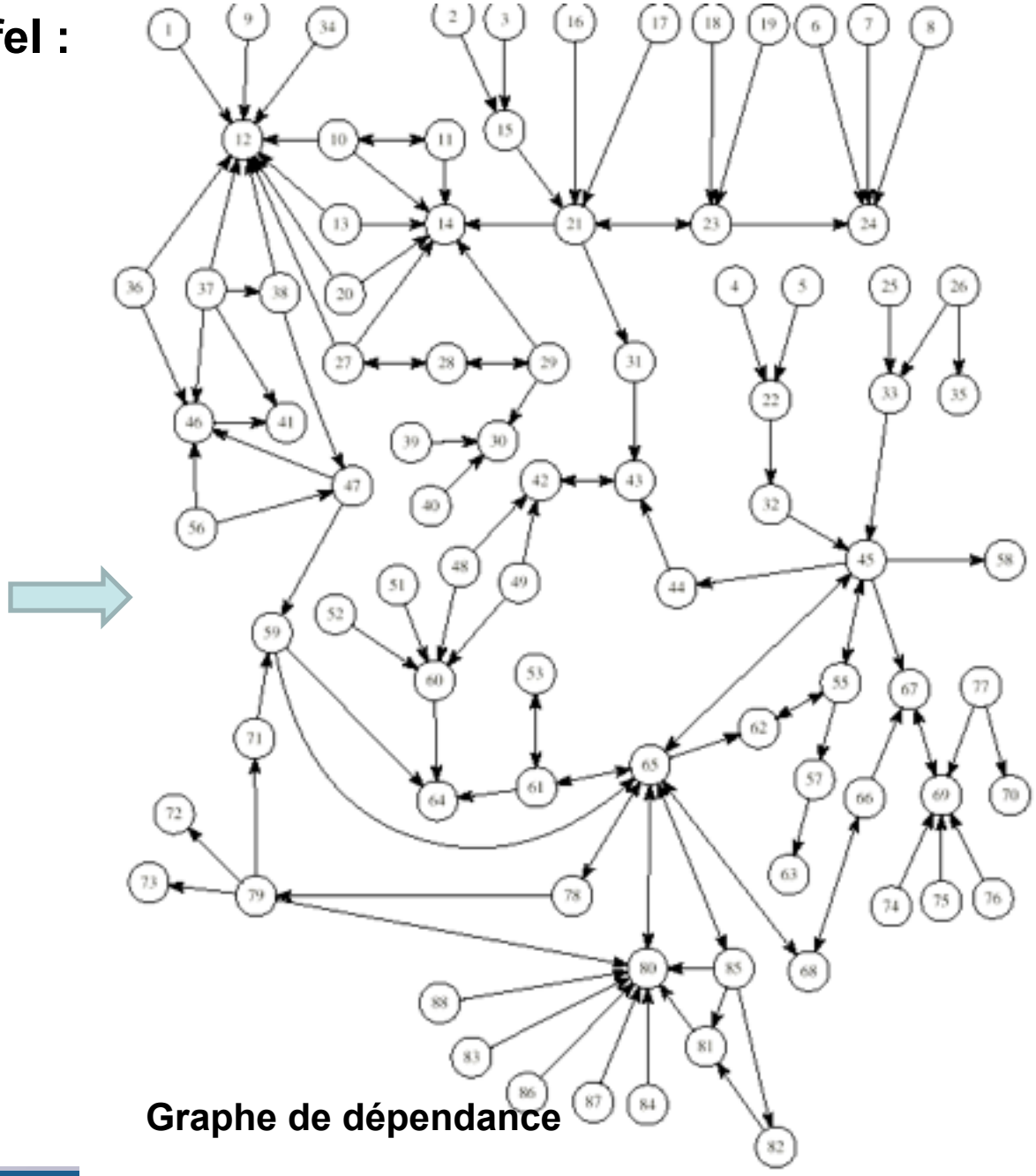


Diagramme UML



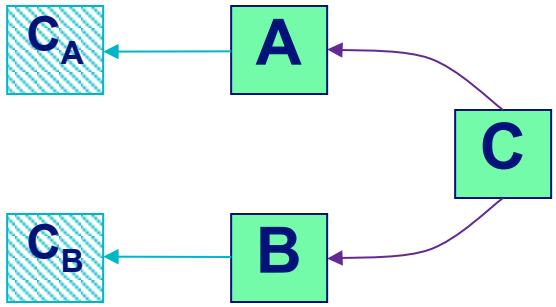
Graphe de dépendance

Test d'intégration : les interdépendances

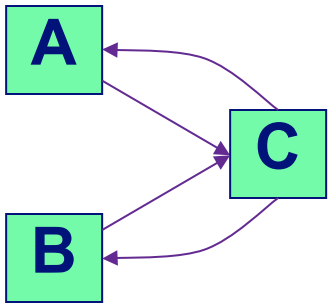
- Une solution simple consiste à contraindre le concepteur
  - pas de boucle dans l'architecture
    - c'est souvent possible
    - **mais** les optimisations locales ne sont pas toujours optimales globalement
    - **mais** concevoir des composants interdépendants est souvent naturel

Bouchon de test

- Bouchon : une unité qui simule le comportement d'une unité



Bouchon spécifique

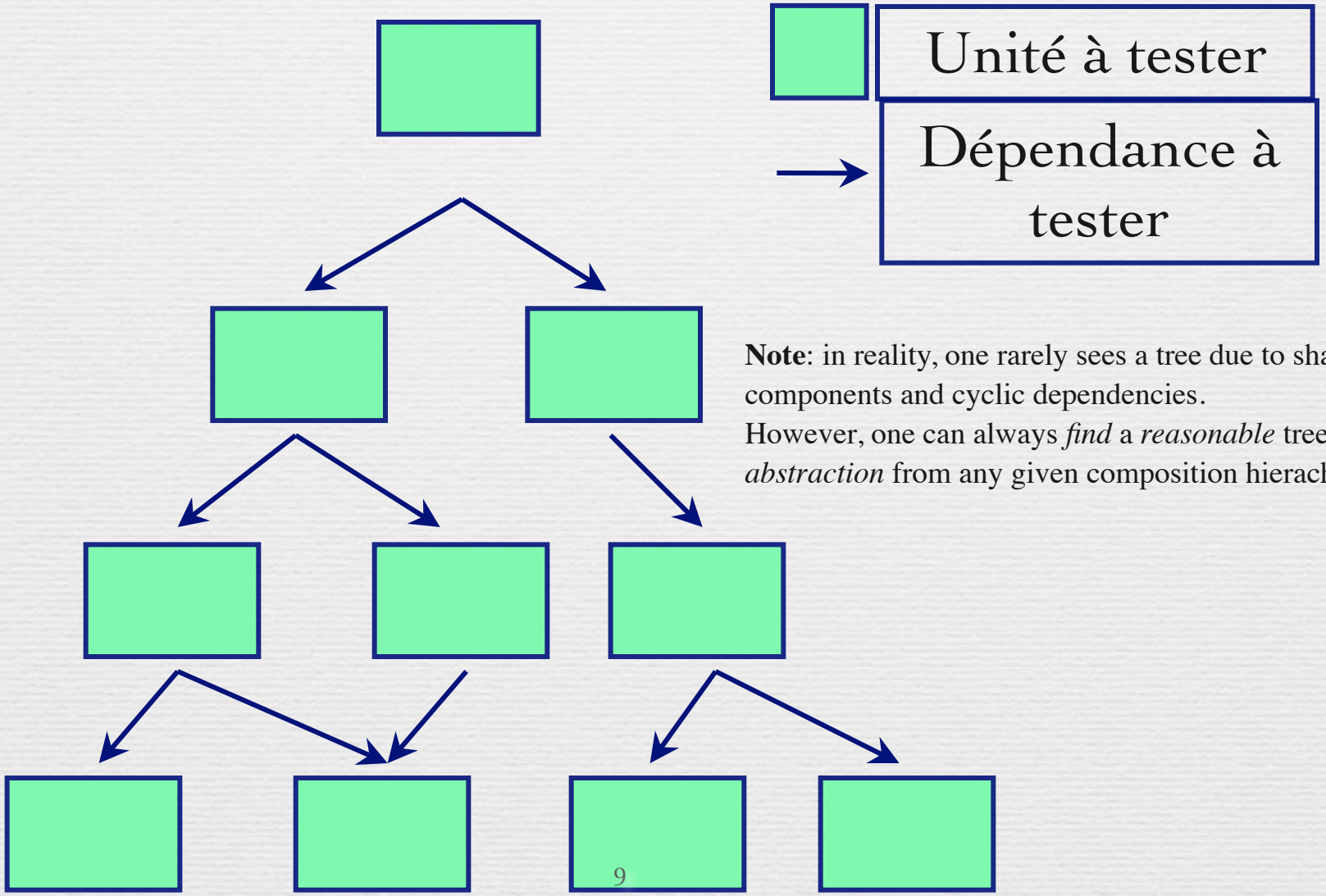


CFC

Diffusion Libre

Tests d'intégration

➡ Architecture des dépendances



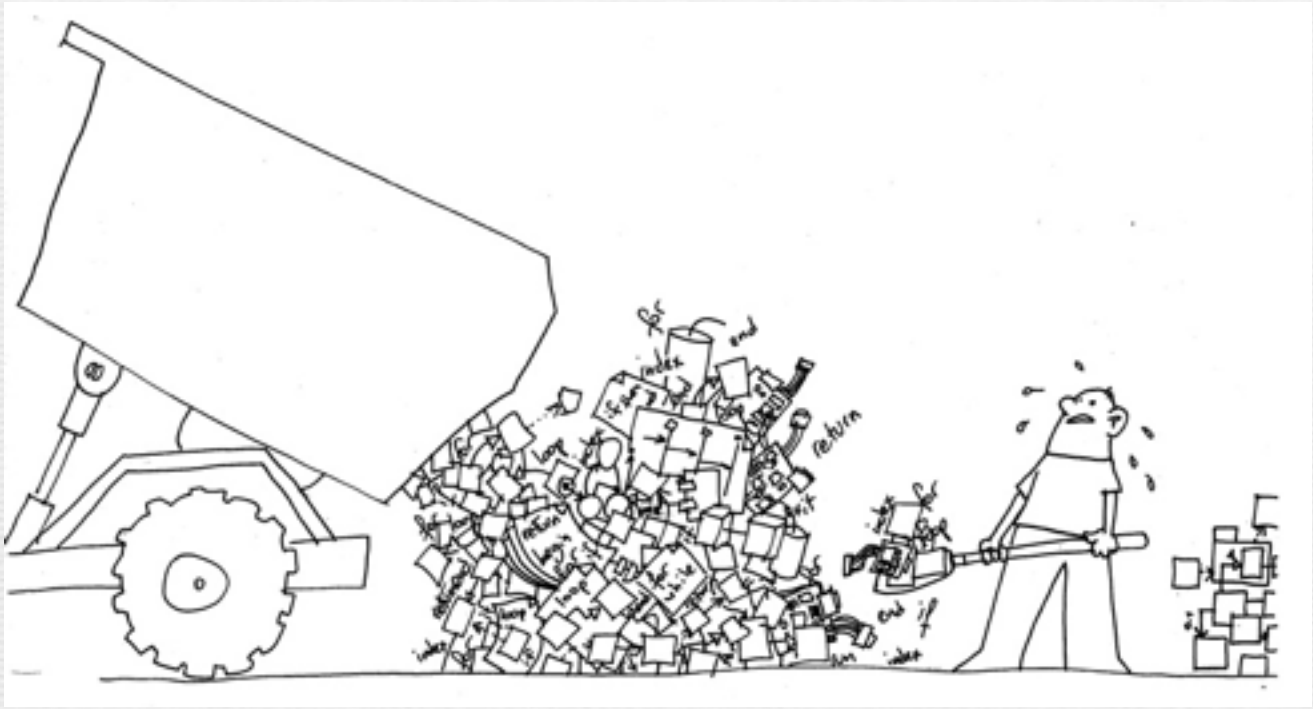
Note: in reality, one rarely sees a tree due to shared components and cyclic dependencies.
However, one can always *find* a *reasonable* tree *abstraction* from any given composition hierachy.

Stratégies

- Big-bang : tout est testé ensemble (peu recommandé)
- Top-down (peu courant)
- Bottom-up (la plus classique)

Intégration - Big-Bang

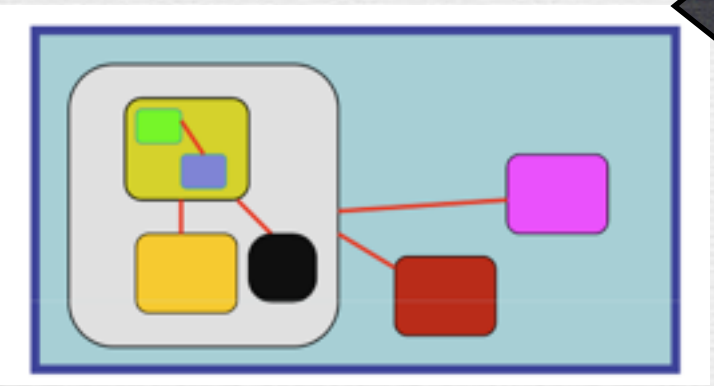
- ***Big Bang*** – Validation du système –



http://emmanuelchenu.blogspot.com/

Intégration - Big-Bang

Intégration de tous les composants à tester en une seule étape. (intégration massive)



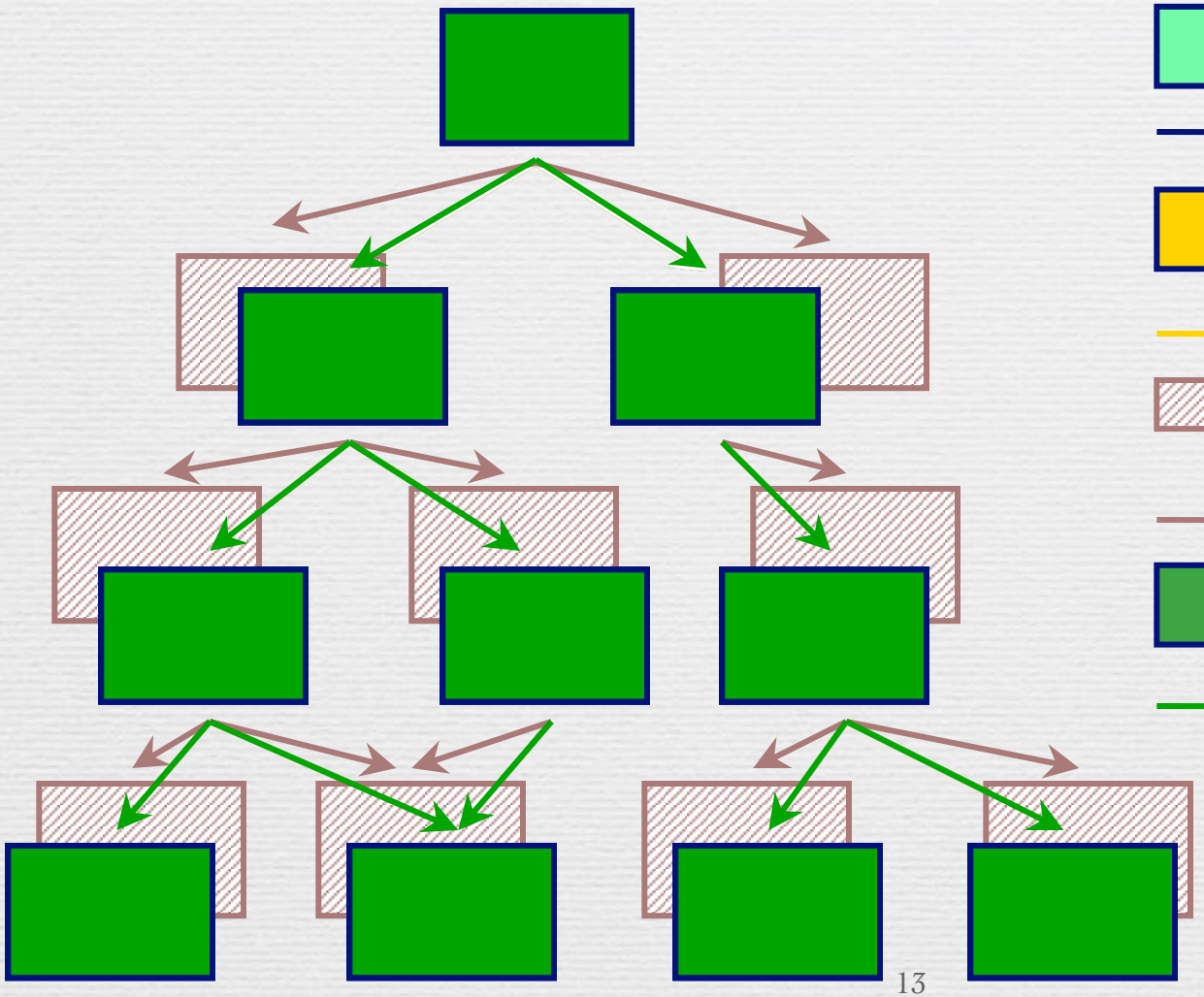
Tests à l'interface du système

Usage Model testing

Principaux Problèmes:

- Les tests produisent des erreurs : Quelle en est la cause?
- La complexité induit des tests manquants
- Les tests ne commencent que lorsque tous les composants ont été «codés».

Approche descendante



Unité à tester

Dépendance à tester

Unité sous test

Dépendance sous test

Bouchon de test (stub)

Dépendance simulée

Unité testée

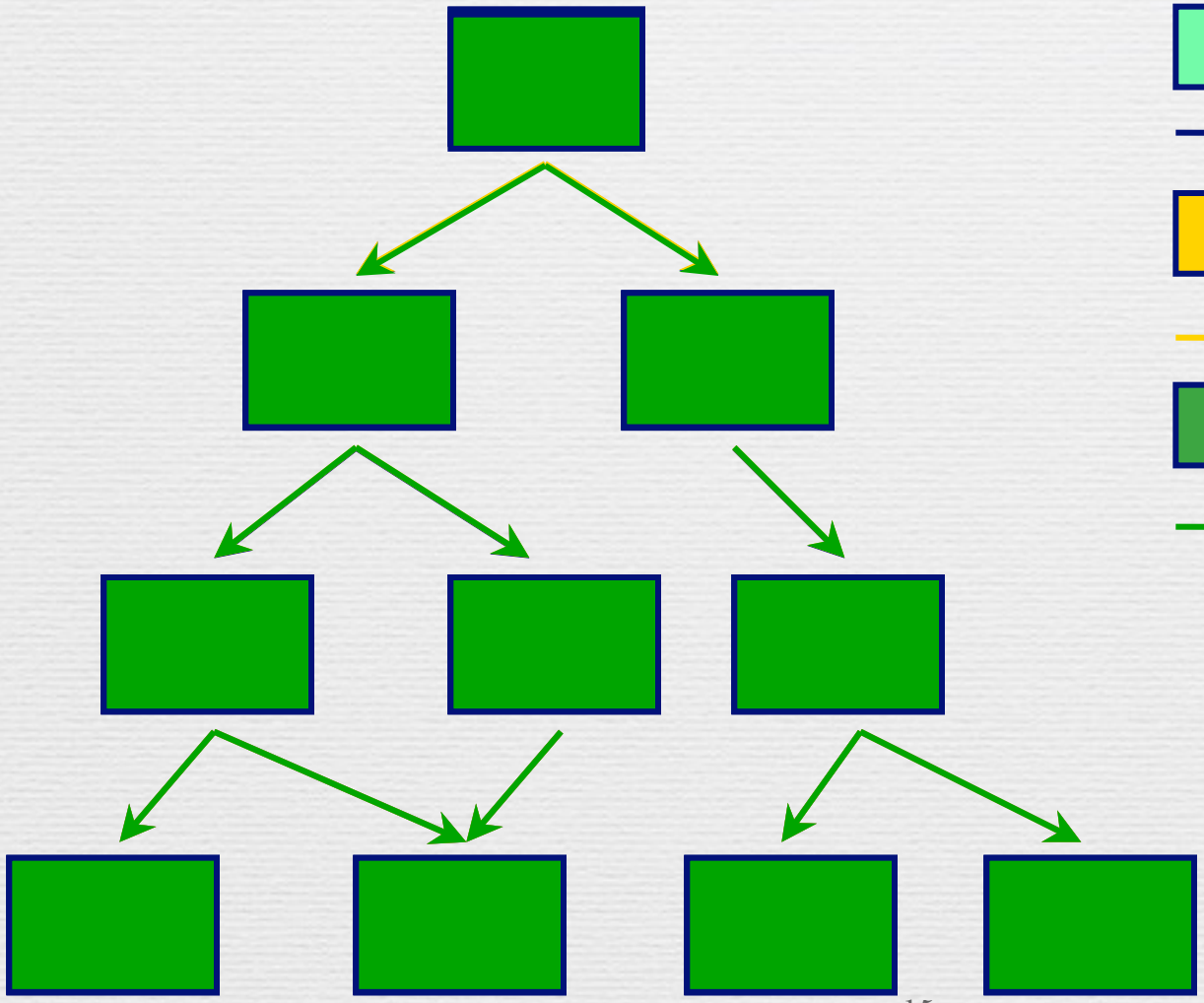
Dépendance testée

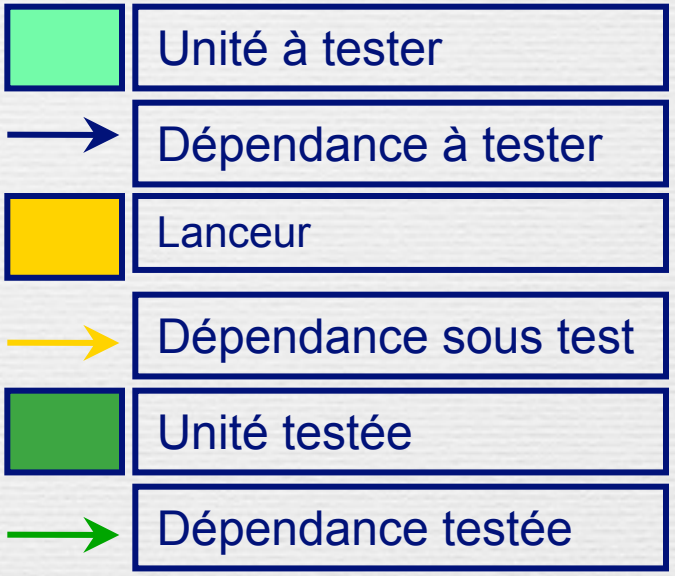
Approche descendante

- Création de **bouchons**
- Test tardif des couches basses
- Détection précoce des défauts d'architecture

- Effort important de simulation des composants absents et multiplie le risque d'erreurs lors du remplacement des bouchons.
- La simulation par « couches » n'est pas obligatoire

Approche Ascendante





Approche ascendante

- **Avantages**
  - Faible effort de simulation
  - Construction progressive de l'application s'appuie sur les modules réels. Pas de version provisoire du logiciel
  - Les composants de bas niveau sont les plus testés,
  - Définition des jeux d'essais plus aisée
  - Démarche est naturelle.
- **Inconvénients**
  - Détection tardive des erreurs majeures
  - Planification dépendante de la disponibilité des composants

Approche Mixte

- Combinaison des approches descendante et ascendante.
- **Avantages** :
  - -Suivre le planning de développement de sorte que les premiers composants terminés soient intégrés en premier ,
  - -Prise en compte du risque lié à un composant de sorte que les composants les plus critiques puissent être intégrés en premier.

- La principale difficulté d'une intégration mixte réside dans sa complexité car il faut alors gérer intelligemment sa stratégie de test afin de concilier les deux modes d'intégration : ascendante et descendante.