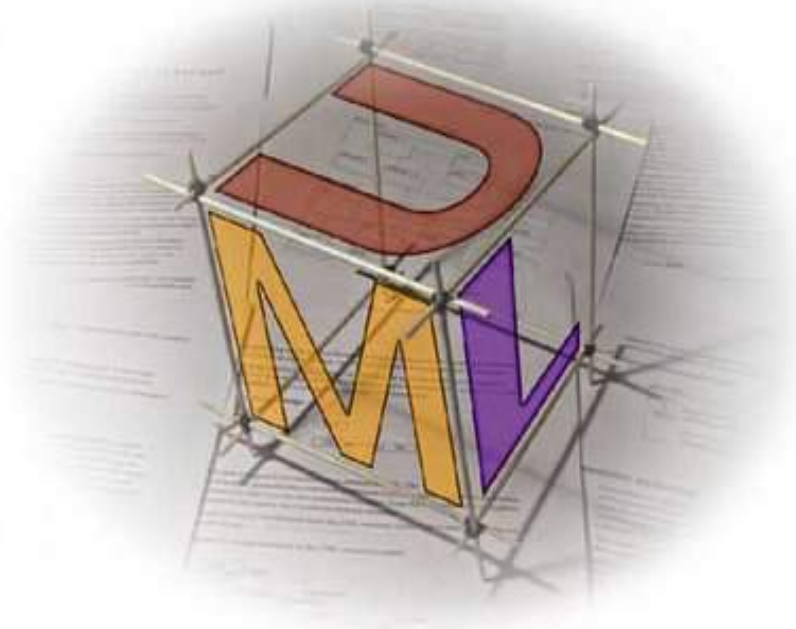


Analyse et conception objet du logiciel : Méthode de conception objet et notation UML

G.L.

Rémy Courdier



Email : Remy.Courdier@univ-reunion.fr

Plan du cours

G.L.

■ Introduction au Génie Logiciel

■ L'approche Orientée Objet et Notation UML



■ Les diagrammes de modélisation

■ Relations entre les différents diagrammes

■ De l'analyse à la conception

■ Relation entre les notations OMT et UML

■ Les design patterns

Chapitre 1 : Introduction au Génie Logiciel

The logo consists of the letters 'G.L.' in a stylized, bold, black font. The 'G' and 'L' are connected, and there is a small dot after the 'L'. The logo is positioned in the top right corner of the slide, partially overlapping the chapter title.

- **Le Génie Logiciel : Genèse et Objectifs**
- **Les Cycles de vie de développement industriel de logiciels**
- **Les bases de la qualité du logiciel**
- **Des méthodes fonctionnelles aux méthodes “Objet”**

1.1 Le Génie logiciel : Genèse et Objectifs

The logo consists of the letters 'G.L.' in a stylized, bold, black font. The 'G' and 'L' are connected, with a small dot after the 'L'. The logo is positioned in the top right corner of the slide.

Sans estimation fiable, un projet de développement informatique échoue dans plus de 80 % des cas : livraisons hors délai ou hors budget. Aux États-Unis, plus de 50 % des projets dépassent de 189 % les budgets prévus. seuls 16 % des projets sont finalisés dans le temps et le budget impartis, 9 % pour les grandes entreprises. (Standish Group 2019)

- Difficulté de maîtrise des coûts
- Difficulté de réalisation de plannings
- Difficulté de maîtrise des délais de réalisation
- Difficulté d'amélioration de la productivité et de la qualité
- Difficulté de gestion de projets logiciels de grande ampleur (Programming in the Large)

Nombreux échecs ou résultats fournis par les logiciels insatisfaisants pour les clients finaux.

Tout ceci dans un contexte de compétition internationale sévère

Quelques idées sur les coûts...

The logo consists of the letters 'G.L.' in a stylized, bold, black font. The 'G' and 'L' are connected, and there is a small dot after the 'L'. The logo is positioned in the top right corner of the slide.

■ Répartition : (Ref. Boehm)

✓ Analyse/Conception

- ⇒ 30-35 % : Système d'exploitation, Aérospatiale
- ⇒ 40-45 % : Contrôle et Régul. indus., Calcul scientifique, Gestion

✓ Codage

- ⇒ 15-20 % : Système d'expl., Contrôle et Régul. indus., Aérospatiale
- ⇒ 25-30 % : Calcul scientifique, Gestion

✓ Test/Intégration

- ⇒ 25-35 % : Contrôle et Régul. indus., Calcul scientifique, Gestion
- ⇒ 45-50 % : Système d'exploitation, Aérospatiale

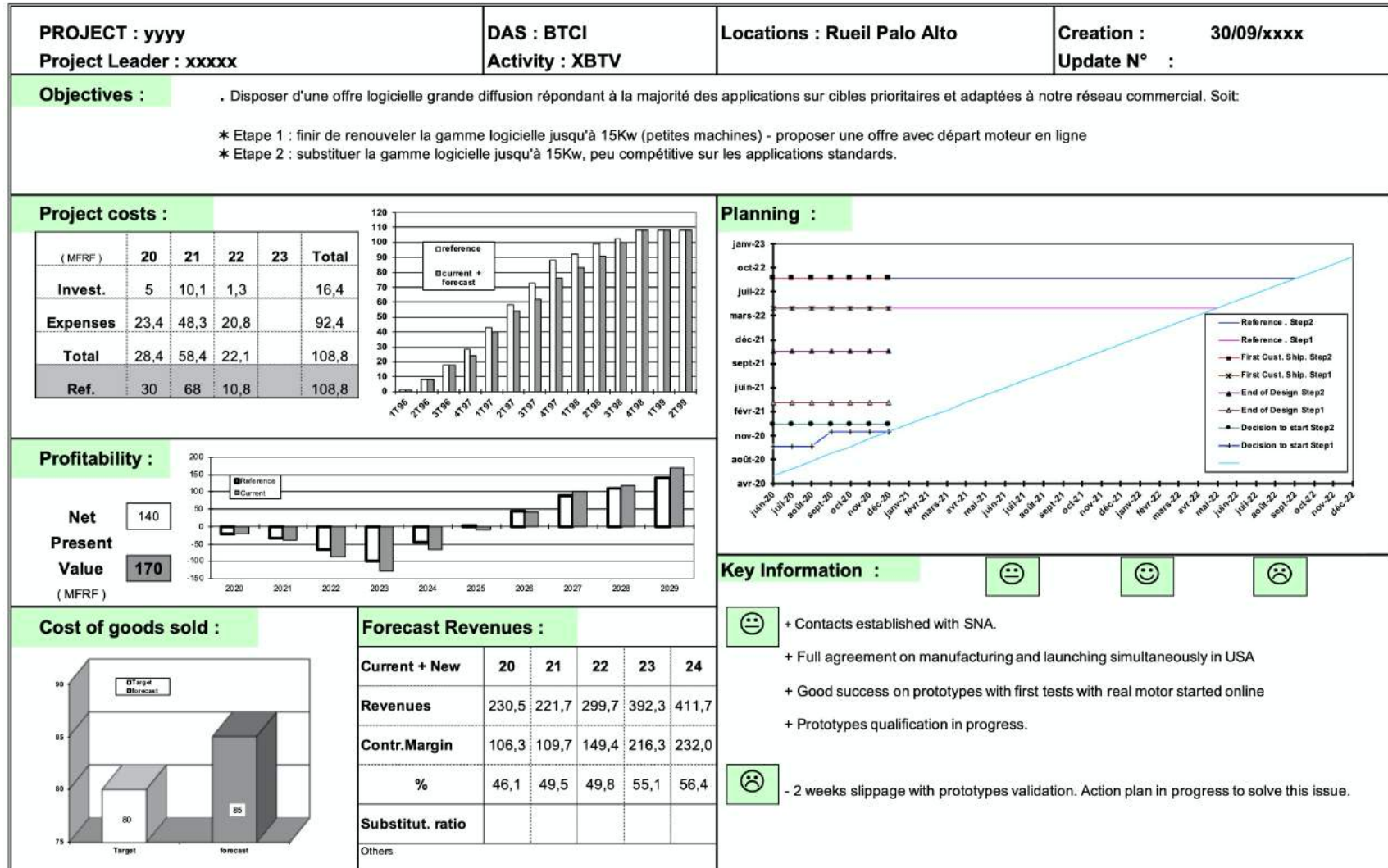
✓ Maintenance

- ⇒ coûts très importants...

■ Peu de capitaux d'investissement nécessaires

■ Frais de personnel élevés

Faire une gestion de projet rigoureuse



Ne suffit pas à éviter de nombreux échecs...

G.L.



Trop de tâtonnement...

G.L.

*Logiciels
insatisfaisants*

&

*Difficulté de
maintenabilité*



Quelques thèmes tirés par le Génie Logiciel

The logo consists of the letters 'G.L.' in a stylized, bold, black font. The 'G' and 'L' are connected, and there is a small dot after the 'L'. The logo is positioned in the top right corner of the slide.

*Il n'y a pas de remède miracle, mais
quelques voies à creuser...*

- Qualification du personnel par la formation
- Procédures de gestion de la qualité logiciel
- Outils dédiés au GL (CASE, Logiscopes)
- Langages et environnements de programmation
- Prototypage
- Méthodes formelles et semi-formelles
- Réutilisabilité

■ L'approche "objet"

1.2. Les Cycles de vie de développement industriel de logiciels

■ qu'est-ce qu'un cycle de vie logiciel?

- ✓ Enchaînement des activités de développement logiciel
- ✓ Définition des Pré et Post conditions pour chaque phase
- ✓ Procédures de gestion et d'encadrement
- ✓ Procédures de mesures
- ✓ Cycle de vie logiciel : synonyme de méthodologie logiciel

■ Etapes d'un cycle de vie

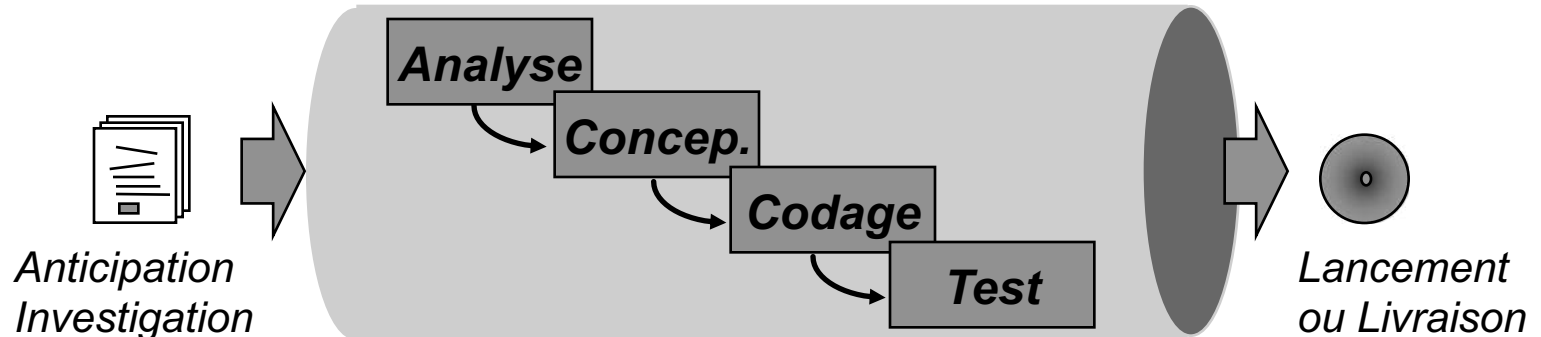
- ✓ Analyse : opportunité fonctionnelle et faisabilité technique
- ✓ Conception : choix tactiques de réalisation et d'architecture
- ✓ Codage : réalisation informatique du détail des opérations
- ✓ Test : tests unitaires et d'intégration
- + Exploitation / Maintenance

■ Les deux grandes catégorie de cycles de vie :

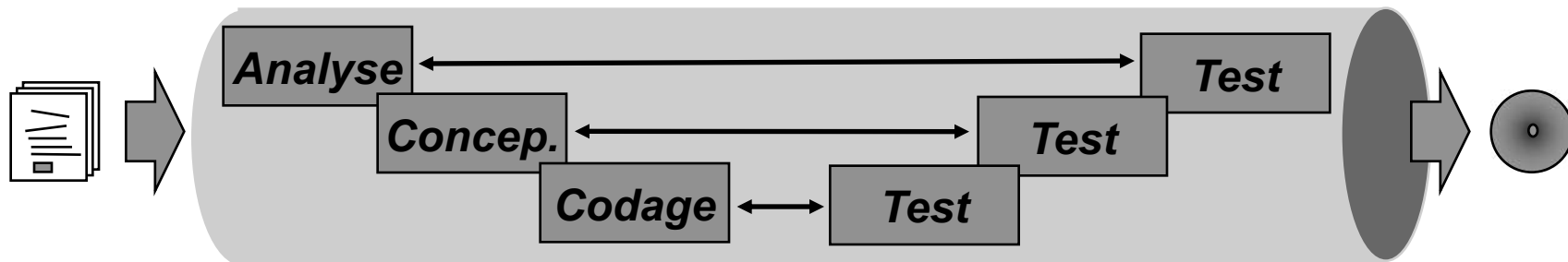
- ✓ Les cycles linéaires : succession d'étapes ordonnées
- ✓ Les cycles itératifs : réalisation incrémentale par évolutions

Les cycles de vie linéaires

■ Cycle en cascade : ouvre des points de visibilité



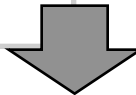
■ Cycle en V : variante courante du cycle en cascade



Problème de l'effet tunnel où l'on ne voit tourner quelque chose qu'à la fin. => Détection d'erreurs tardive

limites du modèle linéaire

Les projets présentent bien souvent une part d'inconnu et donc de risques.



- Méconnaissance des besoins par le client
- Incompréhension des besoins par le fournisseur
- Instabilité des besoins
- Choix technologiques
- Mouvements de personnels
- ...



Le processus de développement d'un logiciel n'est pas naturellement linéaire...

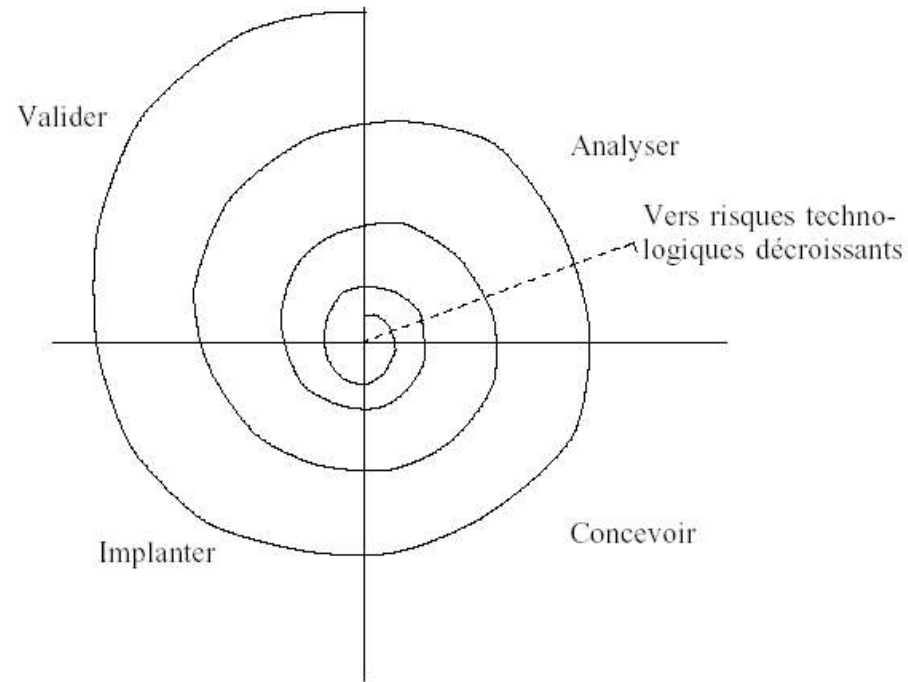
Les cycles de vie itératifs

- **Evaluation d'éléments concrets au cours du développement : élimination de l'effet tunnel**
 - ✓ basé sur l'évolution de prototypes exécutables, mesurables
 - ✓ diminution de l'importance des documents de spéc. détaillée
 - ✓ livraisons intermédiaires => résultats concrets réguliers de l'équipe de développement
 - ✓ meilleurs anticipation et prise en compte des problèmes
 - ✓ meilleure gestion de la prise en compte de modifications de spécification qui peuvent être intégrées dans une itération future
 - ✓ intégration progressive de composants
 - ✓ ...
- **En général, dans les cycles de développement itératifs, chaque itération reproduit le cycle en cascade à une plus petite échelle.**

Le cycle en spirale (Boehm)

- A chaque spire, il y a itération complète sur les phases :
 - ▢ Analyse
 - ▢ Conception
 - ▢ Codage
 - ▢ Test
- A chaque itération, le logiciel doit être dans un état quasi commercialisable
- Grand intérêt en prototypage incrémental
- Très utilisé sur les projets reposant sur l'objet.

« *Design a little, code a little* »



La première spire doit comprendre les éléments les plus abstraits et Le cœur fonctionnel minimum du système

Que faire pour s'améliorer

G.L.

■ Premier « brainstorming »

- ✓ Embaucher des supers chef de projets ?
- ✓ N'embaucher que des experts pour développer ?
- ✓ Faire des heures sup et travailler le week-end ?
- ✓ Faire des opérations coups de poing ?
- ✓ Faire des plans d'actions ?

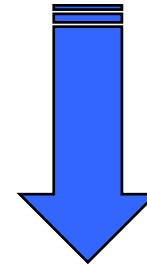
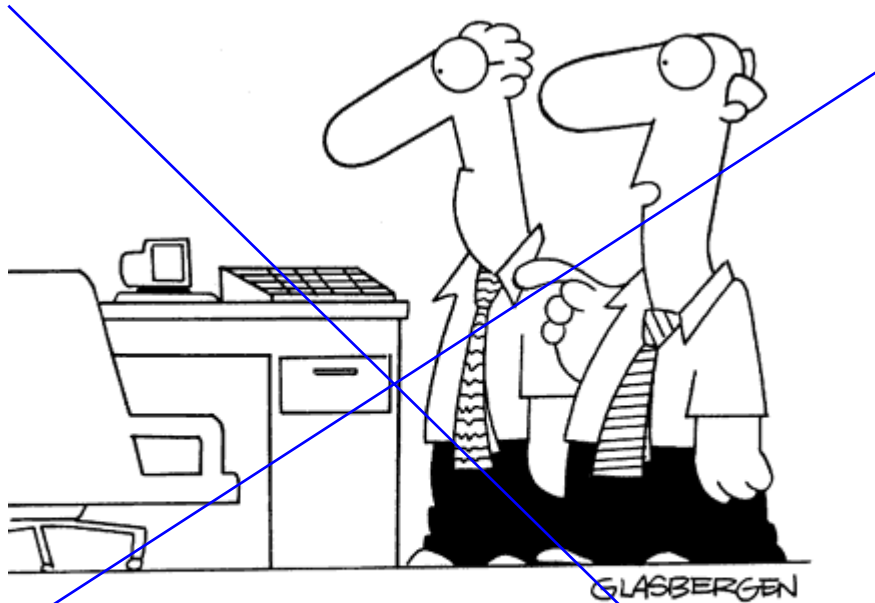
Ceci est déjà fait depuis bien longtemps, alors

■ Trouver des pistes ...

Que faire pour s'améliorer (2)

G.L.

■ ... Trouver des pistes



**Renforcer
la gestion
méthodologique
du cycle de vie
logiciel**

Facteurs de qualité logiciel

■ Facteurs externes (visibles par le client)

- ✓ Exactitude : le logiciel fournit les bons résultats
- ✓ Robustesse : le logiciel réagit correctement à des données fausses
- ✓ Stabilité : possibilité d'intégrer des modif. de spécification légères
- ✓ Fiabilité : exactitude + robustesse
- ✓ Efficacité : performances d'exécution, encombrement mémoire,...

■ Facteurs internes

- ✓ Maintenabilité (support du temps..., testabilité, traçabilité)
- ✓ Portabilité
- ✓ Cohésion : forte cohésion dans les modules
- ✓ Faible couplage entre les modules

Obtention de certification qualité

G.L.



HOPEX CLOUD
documentation, 2018

« The datacenters of MEGA Private Cloud also protect against attacks such as DDoS (Distributed Denial of Service) and have been awarded certifications such as ISO 27001 and ISO 9001. »

HOPEX CLOUD

Integrated Software Solutions to Help Drive Business and IT Transformation in the Cloud

HOPEX Cloud delivers flexibility, reliability and security through private and public cloud. It uses a single integrated platform for all of MEGA's applications, while offering the cloud solution to meet every company's unique needs.

we've been certified
ISO/IEC 27001:2013

An advertisement for CAST HIGHLIGHT. The top section has a dark background with the CAST HIGHLIGHT logo in yellow and white. To the right are the links 'USE CASES' and 'FEATURES & ANALYTICS'. Below this is a grid of dark squares. Overlaid on the grid is the text 'LOST IN YOUR APPLICATION PORTFOLIO? FORGET ABOUT OPINIONS... GET THE FACTS. IN MINUTES.' in yellow and white. At the bottom, a quote from M. Harris, CEO of David Consulting Group, is displayed in white text.

CAST HIGHLIGHT

USE CASES FEATURES & ANALYTICS

LOST IN YOUR APPLICATION PORTFOLIO?
FORGET ABOUT OPINIONS...
GET THE FACTS. IN MINUTES.

"A must-have for CIOs. Shed light on your portfolio's health, potential savings, and software risk, in a week."
M. Harris, CEO, **David Consulting Group**

Normes de l'ISO

G.L.

■ ISO 9126

- ✓ Ensemble de normes qui définit le modèle de qualité pour un produit logiciel
- ✓ Conception – Fabrication - Utilisation

■ II – ISO 14 598

- ✓ Ensemble de normes publié par l'AFNOR sous le titre « Ingénierie du logiciel –Évaluation de produit logiciel »
- ✓ Définit les démarches méthodologiques pour l'évaluation de la qualité logiciel

■ III – ISO 25 000

- ✓ Square : Software QUALity Requirements and Evaluation
- ✓ Poser le cadre et les références pour définir et évaluer les exigences qualité
- ✓ Retenu par le SEI pour améliorer les performances du CMMI
- ✓ A terme : devrait remplacer les normes ISO 9126 et ISO 14598

NORME ISO 9126

G.L.

- **ISO 9126 partie 1 (1992)**
 - ✓ Caractéristiques de qualité
 - ✓ Directives d'utilisation
 - ✓ Statut de norme
- **ISO 9126 partie 2 (2003)**
 - ✓ Métrologie interne
 - ✓ Rapport technique
- **ISO 9126 partie 3 (2003)**
 - ✓ Métrologie externe
 - ✓ Rapport technique
- **ISO 9126 partie 4 (2003)**
 - ✓ Métrologie d'usage
 - ✓ Rapport technique

Définit le modèle de qualité pour un produit logiciel



ISO 14598

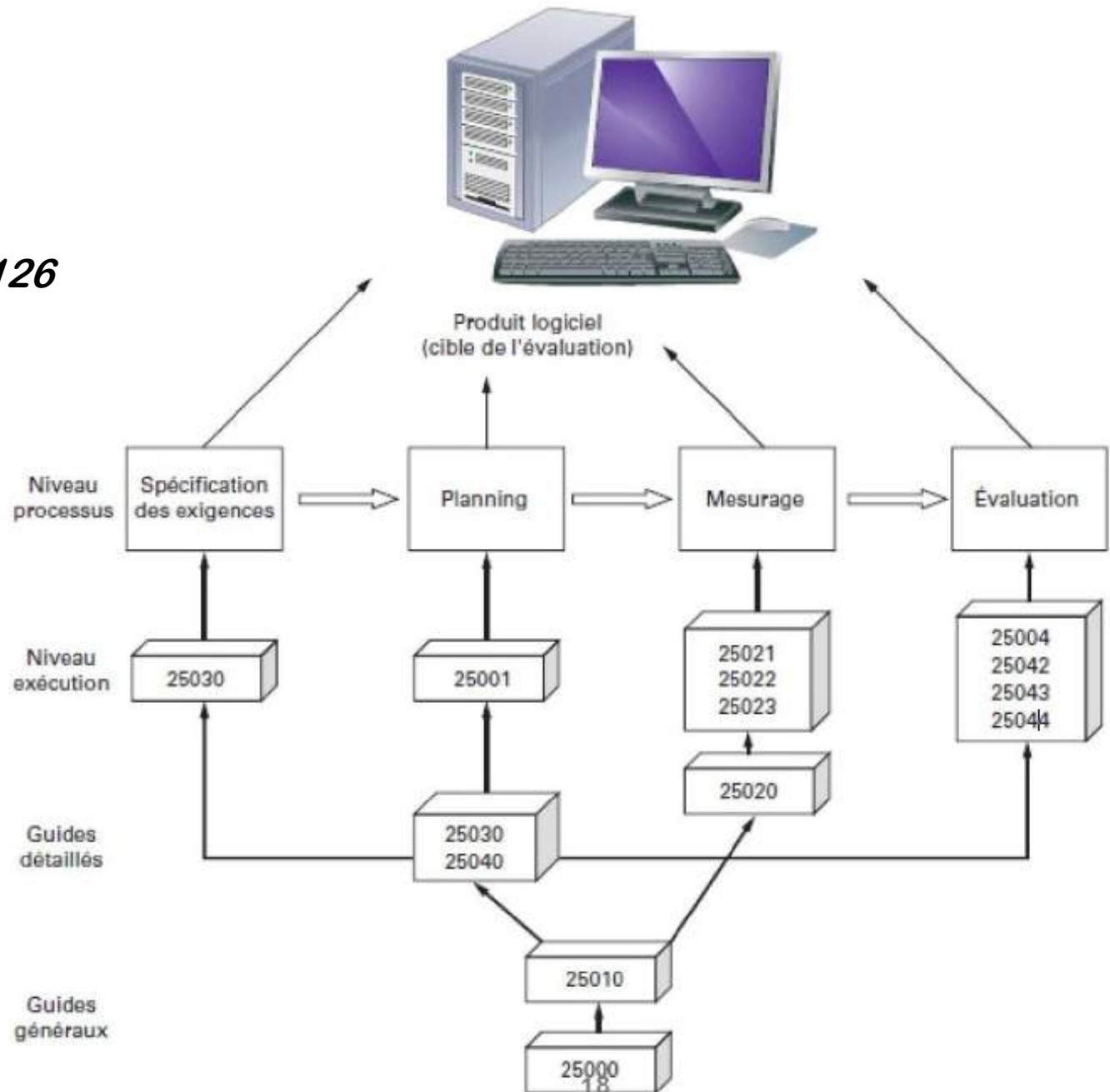
G.L.

Définit des démarches méthodologiques pour l'évaluation de la qualité logiciel



ISO 25000

*Remplace les
normes ISO 9126
et ISO 14598*



1.3. Bases pour la Qualité logiciel



■ CMM (Capability Maturity Model) du SEI (Carnegie Mellon)

✓ Niveau 1 : Initial

- ⇒ Peu de formalisation, Abandon de toute méthode en cas de crise
- ⇒ Le processus de développement est « ad hoc », et parfois même chaotique. Peu de procédures sont définies et le succès repose sur des efforts individuels.

✓ Niveau 2 : Répétable - Méthodes élémentaires de gestion

- ⇒ Processus stabilisés, résultats statistiquement répétables
- ⇒ Une procédure de gestion minimale est définie pour *suivre les coûts, les délais et les fonctions*. Les procédures nécessaires sont en places pour répéter les succès antérieurs à des projets similaires.

✓ Niveau 3 : Défini - Définition du processus de développement

- ⇒ Les processus de gestion et technique sont documentés, standardisés à un processus standard de l'organisation. Tous les projets utilisent une version approuvée et adaptée des processus standards pour développer et maintenir le logiciel.
- ⇒ Gestion de configuration rigoureuse, respects des normes et standards, inspections et tests formels, existence d'un service de GL ou Qualité logiciel.

✓ Niveau 4 : Maîtrisé : Gestion du processus de développement

- ⇒ Des mesures détaillées du développement et de qualité sont collectées. Les processus et le produit sont quantitativement compris et contrôlés.

✓ Niveau 5 : Optimisé : Contrôle et optimisation

- ⇒ Les processus sont *continûment améliorés* par les analyses des mesures.

Modèle d'évolution des capacités logiciel de CMM

G.L.

L'état du CMMI est
maintenue par :

Software Engineering
Measurement and
Analysis (SEMA)

à l'adresse

<http://www.sei.cmu.edu/>



Zoom sur CMM
ppt – pdf - format IE

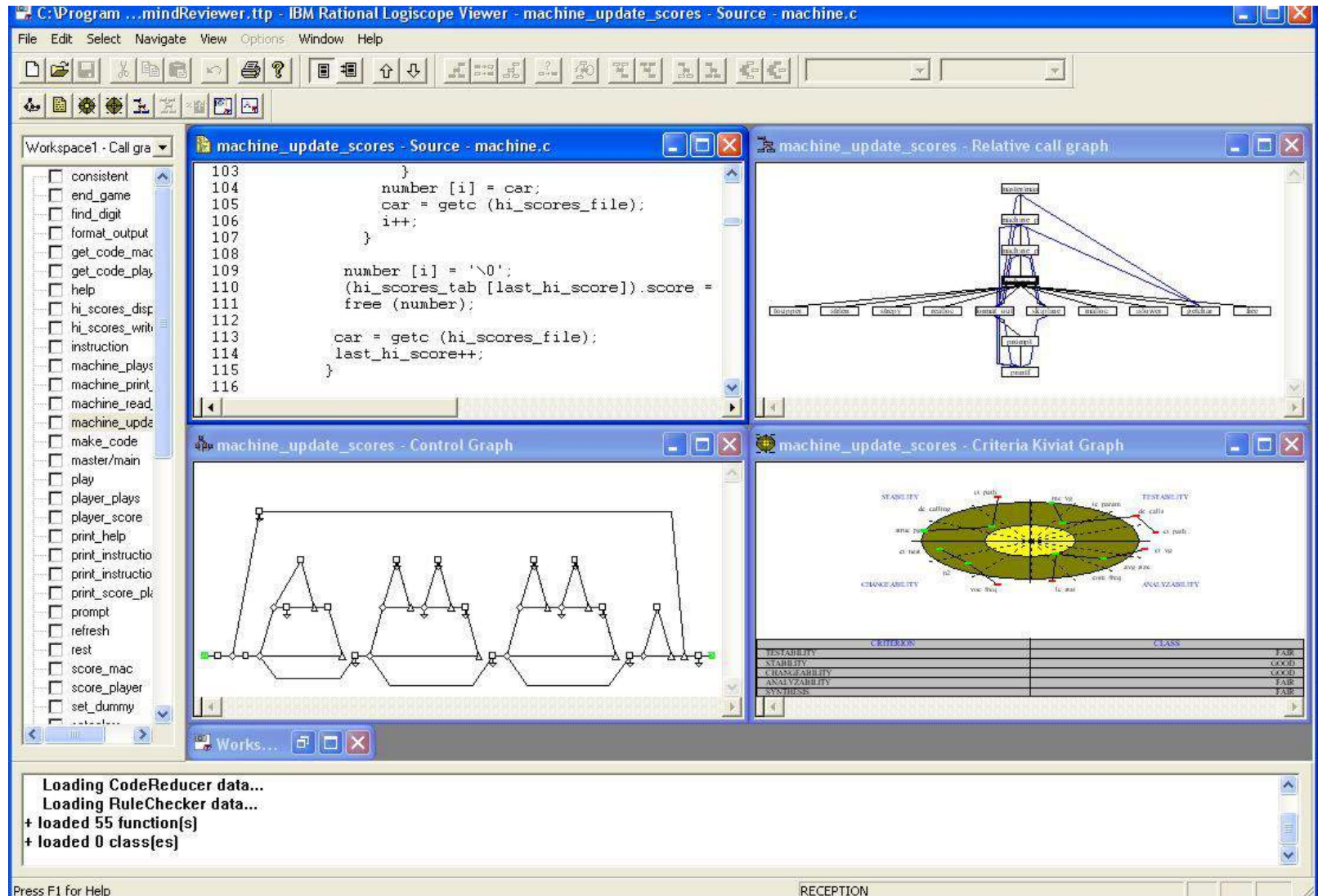
Le profilage de code

- Étape permettant de vérifier et d'évaluer le code source d'un logiciel en calculant les métriques
- Métrique : Nombre qui mesure la grandeur/la complexité du code d'un logiciel

■ Exemples de métriques

- ✓ Nombre de lignes de codes
- ✓ Evaluation du nombre d'opérateurs et d'opérandes
- ✓ Mesure le nombre cyclomatique : nombre de chemins indépendants dans le graphe
- ✓ ...

G.L.



Un mot sur la gestion de versions logiciel

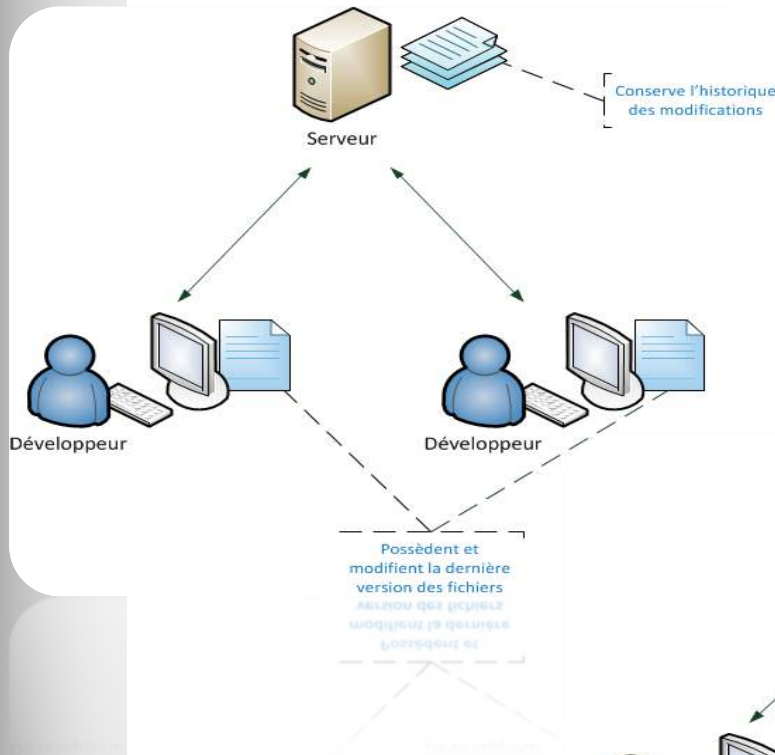
G.L.

- Maîtriser les états des fichiers d'une application logicielle à travers ses nombreuses évolutions dans son cycle de vie.
 - ≡ **historisation, archivage (crash poste développeur)**
- Tracer toutes les modifications sur les fichiers d'une application
 - ≡ **comparaison de versions, retour arrière**
- Faciliter le travail en équipe et la parallélisation des tâches
 - ≡ **Résolution de conflits de synchronisation, simplification de la coordination**
- Permettre le développement simultanée de variantes de versions d'une application.
 - ≡ **Flexibilité, souplesse, expérimentation**

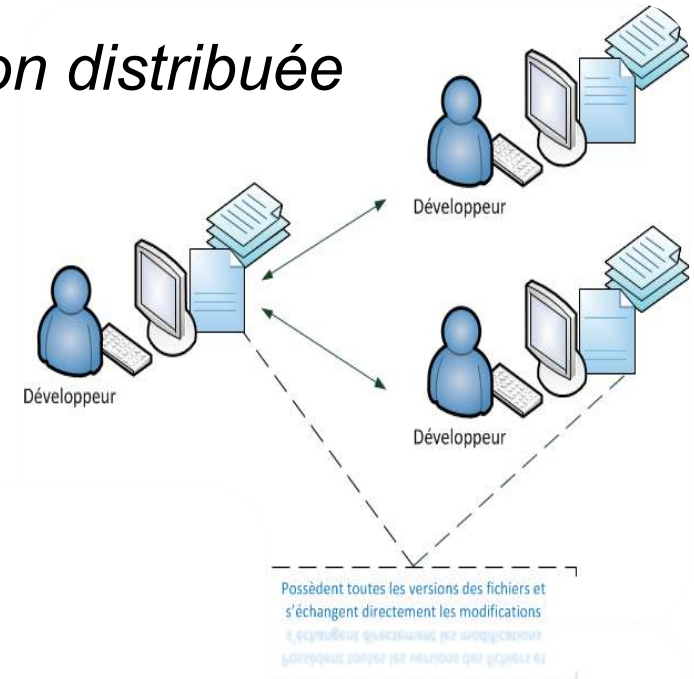
Type de solutions

G.L.

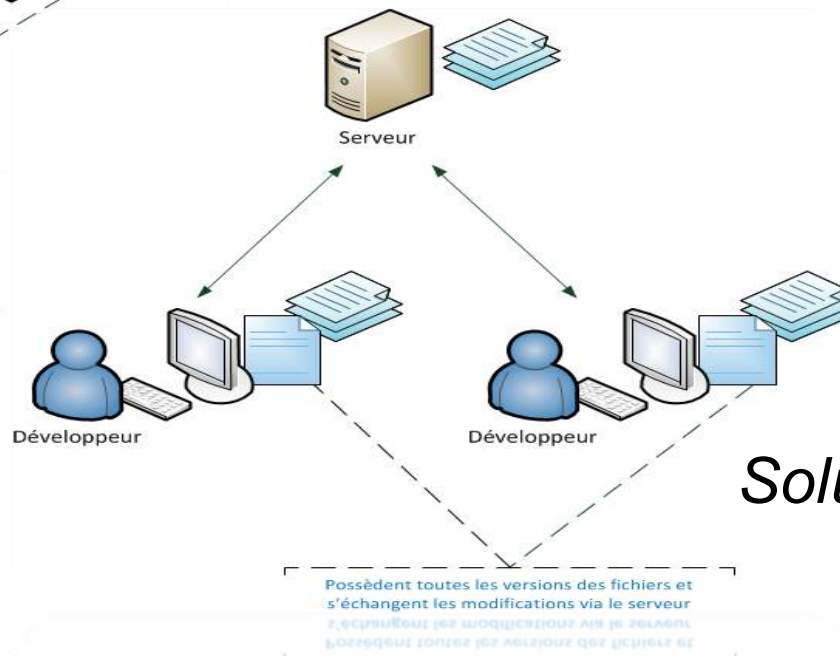
Solution centralisée



Solution distribuée



Solution distribuée avancée



Exemple d'outils de gestion de versions logiciel

G.L.

Solutions centralisées

Current Version System, en abrégé **CVS**

Le plus ancien (1990), encore très répandu

Projets : Open BSD

Apache Subversion, en abrégé **SVN**



Conçu pour remplacer CVS, meilleure implémentation
Le plus utilisé, simple d'utilisation

Projets: Apache, Redmine, Struts

Mercurial



Plus récent et plus puissant

Projets : Mozilla, Python, Openoffice.org

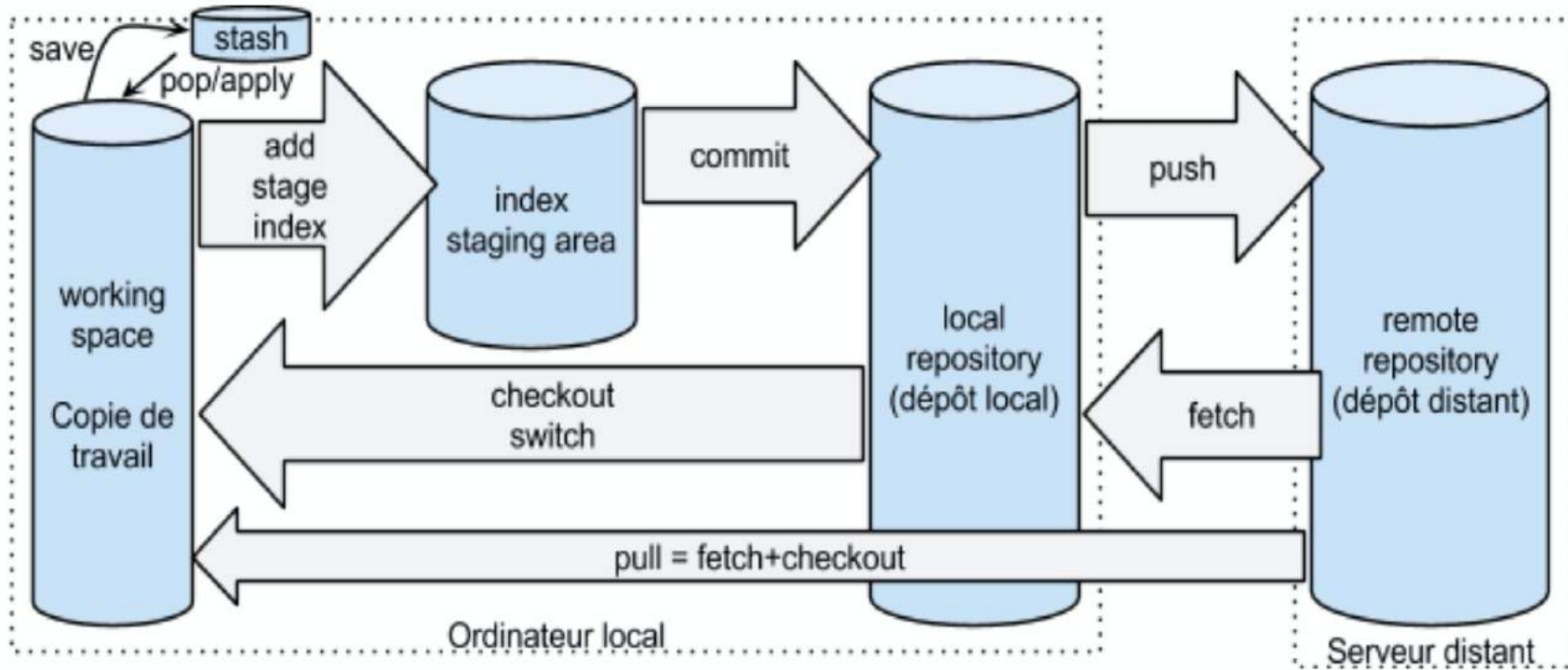


Puissant et récent (2005 par Linus Torvalds)
Spécialement optimisé pour le noyau Linux

Projets : Kernel de Linux, Debian, VLC, Android

Solutions distribuées

Processus et vocabulaire de Git



Source : Les logiciels de gestion de version- exemple de GIT et application avec SmartGIT, Rémi SHARROCK - www.remisharroock.fr

La modularité où les boîtes noires réutilisables...

Découpage du logiciel en modules “indépendants” présentant des caractéristiques d’abstraction, d’encapsulation, et de faible couplage

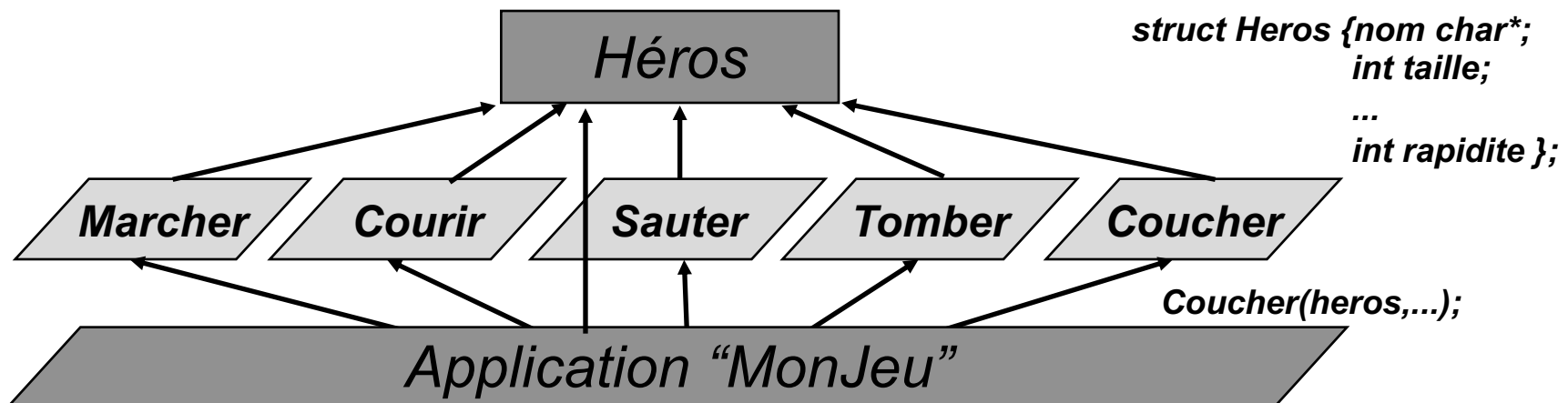
- ✓ **abstraction** : chaque module doit correspondre à une abstraction pré-existante ; il doit regrouper des traitements communs applicables à des entités ou concepts variés. L’abstraction permet de détacher au maximum le code du module du contexte.
- ✓ **encapsulation** : masquage de la mise en œuvre effective du module, du “comment c’est fait” . Seules les éléments accessibles de l’extérieurs sont visibles et spécifiés précisément.
- ✓ **faible couplage** : limitation du niveau d’interaction entre modules (dépendances de génération,...). Il est indispensable que les liens entres modules soient bien définis (couches logicielles) et le moins nombreux possible pour qu’il y ait effectivement modularité.

1.4. Des méthodes fonctionnelles aux méth. objet

L'approche fonctionnelle

G.L.

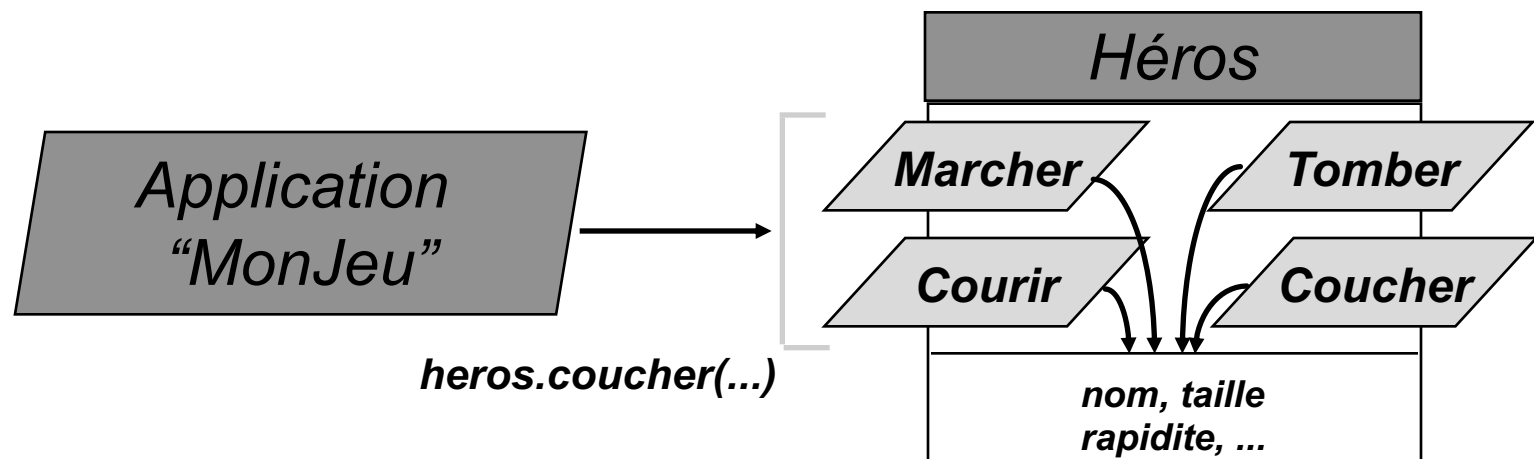
- **Raisonnement en terme de fonctions du système**
 - ✓ l'accent est mis sur les fonctions et non sur les données
- **Séparation des données et du code de traitement**
 - ✓ transposition dans les méthodes des contraintes du matériel
- **Diffusion des responsabilités**
 - ✓ intégrité des données non garanties
 - ✓ ajout possible de nouvelles opérations à tout moment
- **Décomposition fonctionnelle descendante**



L'approche Objet

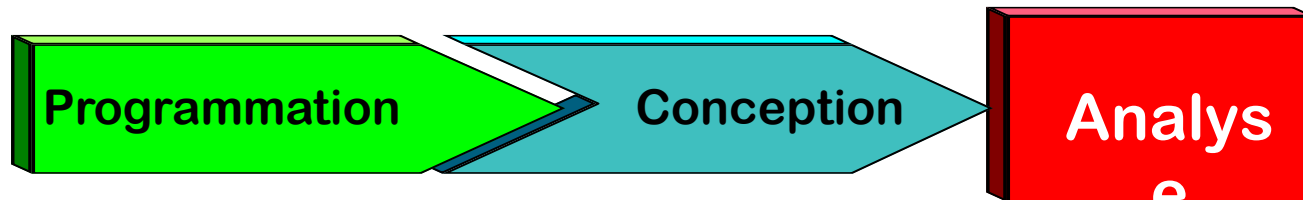
G.L.

- Regroupement données-traitements
- Diminution de l'écart entre le monde réel et sa représentation informatique (approche naturelle)
 - Les informaticiens sont pervers : le monde est avant tout objet
- Localisation des responsabilités : encapsulation
- Décomposition par identification des relations entre objets :
 - association, composition , généralisation/spécialisation



Evolution des méthodes

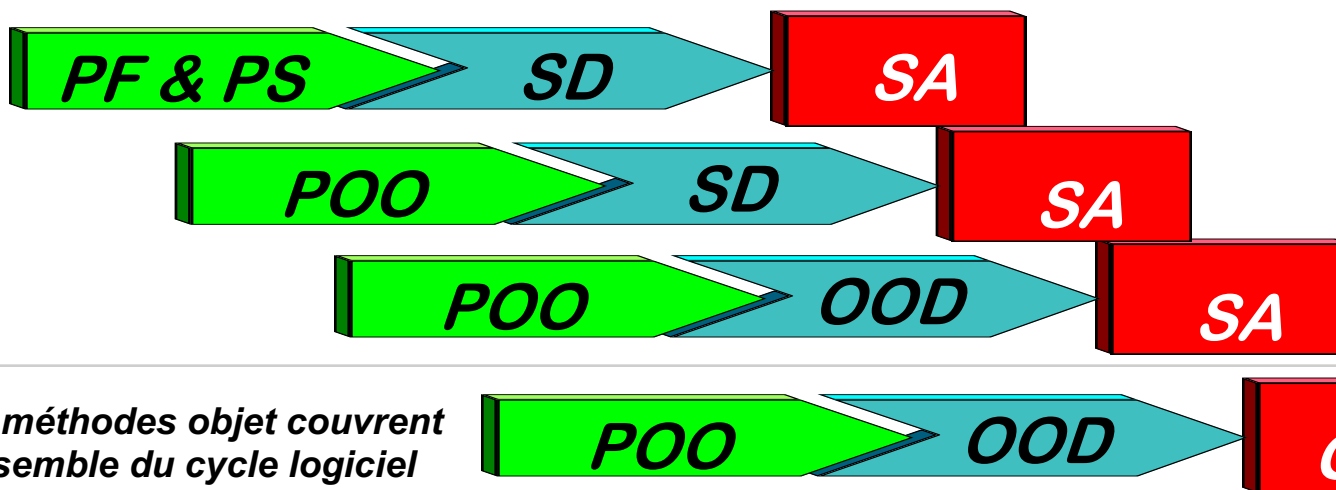
G.L.



L'évolution des méthodes s'est faite de la programmation vers l'analyse

PF & PS : Prog. fonctionnelle et structurée
SD : Conception structurée
SA : Analyse structurée

POO : Programmation orientée objet
OOD : conception objet
OOA : Analyse objet



Les méthodes objet couvrent l'ensemble du cycle logiciel

Panorama des méthodes

G.L.

■ Méthodes structurées

- ✓ programmation structurée (Dijkstra)
- ✓ Décomposition fonctionnelle descendante (SA/SD)
- ✓ SADT/SART

■ Modélisation de Systèmes d'Information

- ✓ Diagrammes entités-relations
- ✓ Merise

■ Méthodes Objet

- ✓ OOD : Booch (91,93)
- ✓ OOA : Coad-Yourdon (90)
- ✓ HOOD : pour Ada (88)
- ✓ OOM : Bouzeghoub (93) merise
- ✓ OOSE : Jacobson
- ✓ ...Shlafer-Mellor, Meyer, Embley
- ✓ OMT : Rumbaugh (91,93)

***UML***

La spécification UML

Grady Booch
Jim Rumbauch
Ivar Jacobson
(trois amigos)

Groupe de travail
dirigé par
Mary Loomis et
Jim Odell

Jim Rumbauch, UML 2.0
Guide de référence UML
Ouvrage de référence pour
les notations UML

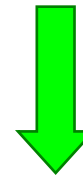


UML V0.8 diffusée en 10/95

UML

Unified Modeling Langage
for Object-Oriented Development

UML V1.0 remise à l'OMG le 17/01/97



UML V2.0 remise à l'OMG en 2004
www.omg.org & www.uml.org

Fin du Chapitre 1

Introduction au génie logiciel