Génie Logiciel : La crise du logiciel et évolution de la spécification

Dr. Selma Belgacem Ben Mansour

ISSAT de Sousse

2021-2022

Plan du cours

Plan du cours

Définitions

- Un **logiciel** (*Software*) est "l'ensemble des programmes, procédés et règles, et éventuellement de la documentation, relatif au fonctionnement d'un ensemble de traitement de données". (Selon l'arrêté ministériel français du l'arrêté du 22 décembre 1981 relatif à l'enrichissement du vocabulaire de l'informatique [Journal officiel du 17 janvier 1982]).
- Le **génie logiciel** (*Software Engineering*) est "l'ensemble des activités de conception et de mise en oeuvre des produits et des procédures tendant à rationaliser la production du logiciel et son suivi". (Selon l'arrêté ministériel français du 30 décembre 1983 relatif à l'enrichissement du vocabulaire de l'informatique [Journal officiel du 19 février 1984])

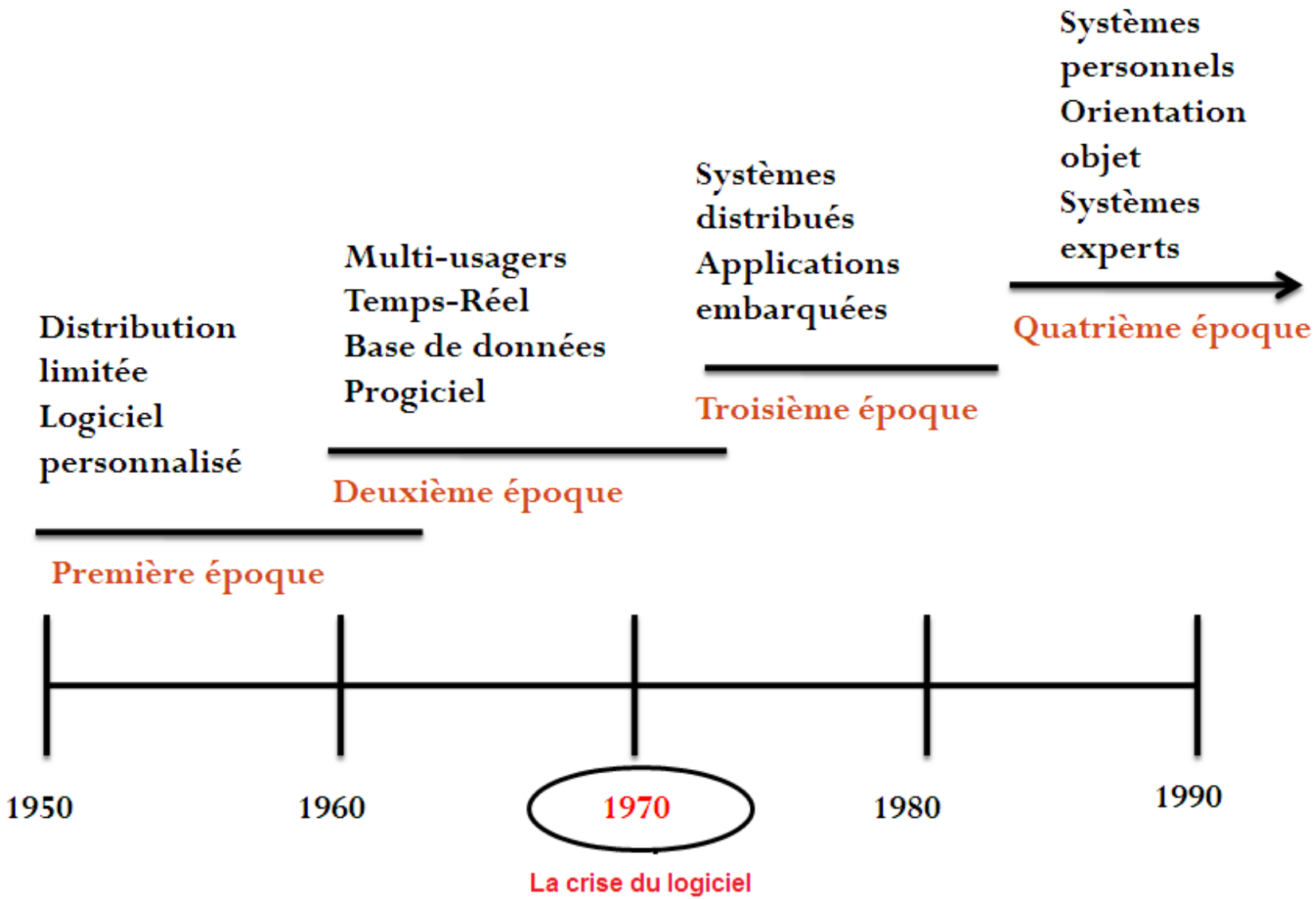
Définition

- Le **facteur principal** qui guide les processus du génie logiciel est l'ensemble des **besoins** exprimés par le **client** qui représente le futur utilisateur du logiciel. Le logiciel produit doit **répondre efficacement** à ces besoins.

- Le **génie logiciel** est proposé comme solution de **la crise du logiciel.**

Plan du cours

La crise du logiciel



Amami Maha, "*Cours génie logiciel*", ISG Tunis, 2009-2010.

La crise du logiciel

La crise du logiciel

La crise du logiciel est caractérisée par une forte dégradation de la qualité des logiciels vis à vis au développement rapide du matériel. Elle a commencé depuis les années 70.

statistiques

Selon le *Standish Group*, en 1995, sur 8380 logiciels développés dans 365 entreprises, 16% des logiciels sont réussis avec délai, budget et spécification respectés, 53% des logiciels sont terminés mais avec délai, budget et spécification non respectés, et 31% des logiciels sont annulés.

La crise du logiciel

Exemples

- En 1962, la sonde Mariner 1 (Sonde : véhicule spatiale, Mariner : programme NASA pour des études astronomiques) vers Vénus s'est perdue dans l'espace à cause d'une erreur de programme FORTRAN (Langage de programmation pour le calcul scientifique.).
- En 1993, le Le système informatique de réservation RESARAIL issu du projet Socrate de la SNCF (*Société nationale des chemins de fer français*) a connu un échec lors de son lancement et a entraîné d'importantes difficultés pour les clients de la SNCF.
- En 1996, le vol 501 d'une fusée du lanceur européen Ariane 5, s'est explosé 36,7 secondes après le décollage à cause d'un bug informatique causant une perte d'un demi milliard de dollars.
- ...

La crise du logiciel

Causes

- Manque de la communication avec l'utilisateur lors du développement du logiciel ce qui mène à une inadéquation des fonctionnalité du logiciel avec les besoins de l'utilisateur.
- Manque de clarté au niveau des besoins de l'utilisateur.
- Manque de la modélisation structurée et de l'application des méthodes du génie logiciel.
- Manque d'organisation et de coopération dans les équipes de travail.
- Faiblesse des tests
- Coût élevé de la maintenance
- Conditions de travail non confortable

La crise du logiciel

Solutions

- Application des méthodes du génie logiciel
- Application du développement itérative du logiciel : génération séquentielle de prototypes du logiciel et application d'un test pour chaque prototype.
- Évaluation élémentaire des composantes du logiciel avec la participation de l'utilisateur.

Plan du cours

Critères de qualité d'un logiciel

• La norme **ISO** 9126 (remplacée depuis 2011 par la norme 25010 et ré-examinée en 2017) a définit des **indicateurs de qualité des logiciels** :

- **Utilité** : adéquation entre le logiciel et les besoins des utilisateurs.
- **Efficacité** : le logiciel permet à ses utilisateurs d'atteindre le résultat prévu.
- **Efficience** : le logiciel atteint le résultat avec un effort moindre ou requiert un temps minimal.
- **Fiabilité** : fonctionnement du logiciel sans erreurs dans une période de temps bien déterminée.
- **Performance** : bon résultat de l'exécution du logiciel.

Critères de qualité d'un logiciel

- **Satisfaction** : confort et évaluation subjective de l'interaction pour l'utilisateur.
- **Interopérabilité** : possibilité d'interaction avec d'autres logiciels.
- **Portabilité** : possibilité d'installation sur différents systèmes d'exploitation.
- **Réutilisabilité** : utilisation dans d'autre contextes.
- **Facilité de maintenance** : facilité de correction, de modification et d'ajout de fonctionnalités.

Plan du cours

Cycle de vie du logiciel

- **Étude de faisabilité** : budget, compétences, ressources...
- **Spécification des besoins** : Déterminer les besoins de l'utilisateur et par la suite les fonctionnalités du logiciel.
- **Conception** : Déterminer l'architecture globale et détaillée du logiciel.
- **Codage** : implémentation des différentes fonctionnalités du logiciel.
- **Tests** : Essayer le logiciel sur des données d'exemple pour s'assurer qu'il fonctionne correctement.
- **Installation** chez le client.
- **Maintenance** : modification si nécessaire.

→ Il existe différents **processus de développement** d'un logiciel (processus séquentiel y compris le modèle en cascade et le modèle en V, processus itératif...). Ces processus décrivent l'organisation des étapes du cycle de vie d'un logiciel.

Plan du cours

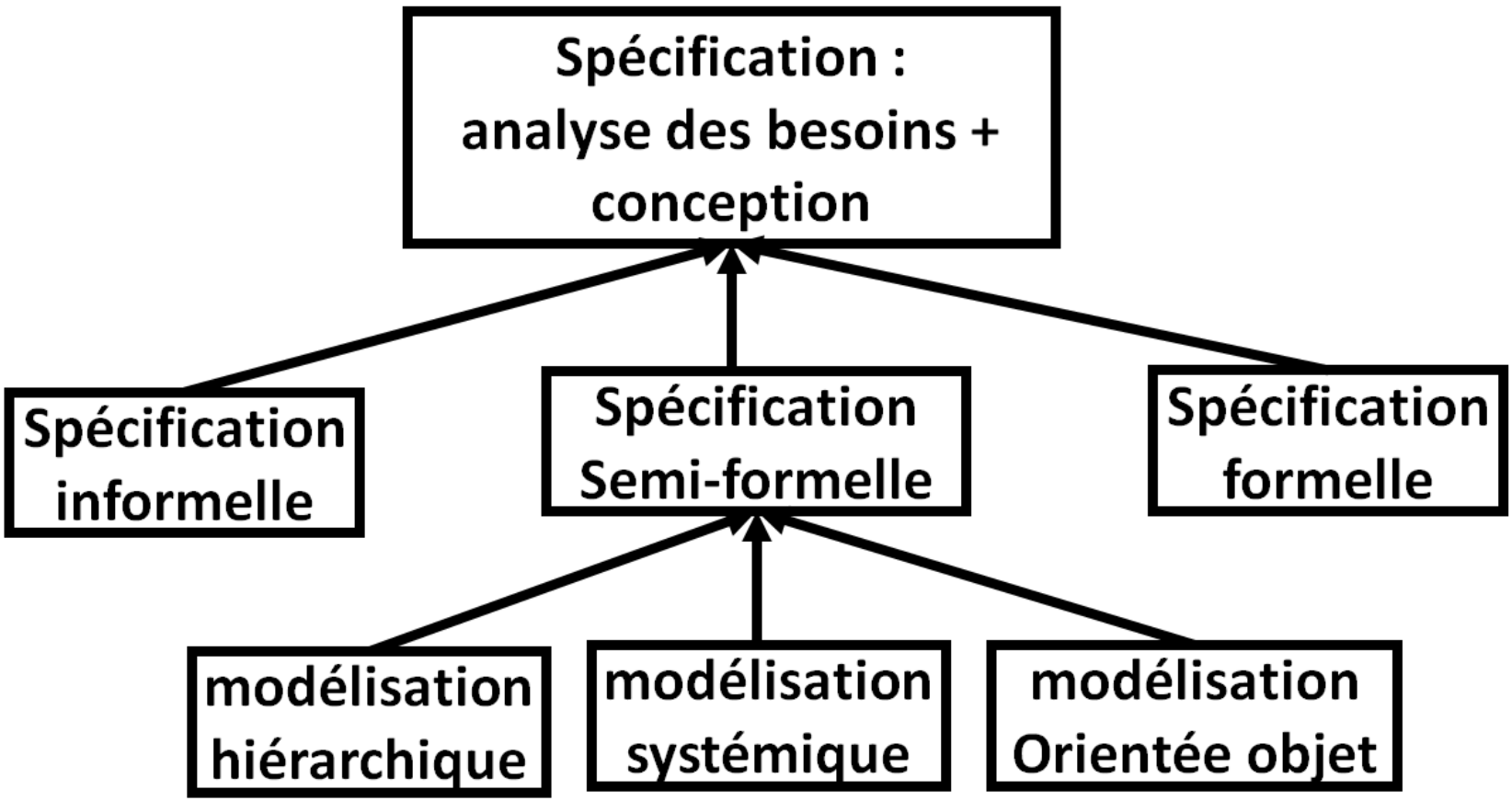
Modélisation du logiciel

- Le cycle de vie d'un logiciel est composé **globalement** de :
  - → Une étape de **pré-production** : étude de faisabilité/planification, spécification des besoins et conception.
  - → Une étape de **production** : codage.
  - → Une étape de de **post-production** : test/validation, installation et maintenance.
- L'étape de **pré-production** est caractérisée par la **modélisation** du logiciel.
- Cette modélisation peut être appelée aussi **spécification** et s'applique précisément à l'étape de **spécification des besoins** et à l'étape de **conception** du cycle de vie d'un logiciel.

Spécification/Modélisation du logiciel

- La **modélisation** d'un logiciel permet de **décrire** :
  - les **fonctionnalités** attendues du logiciel (exp : diagramme de cas d'utilisation),
  - les scénarios possibles de son **comportement** (exp : diagramme de séquences système),
  - son **architecture** (exp : les patrons d'architecture comme le modèle MVC),
  - sa **composition** interne (exp : diagramme de classes),
  - le **fonctionnement** des objets qui le composent (exp : diagramme d'activités),
  - les **états** possibles des objets qui le composent (exp : diagramme d'états-transitions),
  - les **interactions** entre les objets qui le composent (exp : diagramme d'interactions),
  - les relations entre les **données** qu'il gère (exp : diagramme entité-association)...

Les classes de spécification/modélisation



Les classes de spécification/modélisation

- **Spécification informelle** : le logiciel est décrit en langage **naturel**.

→ cette description reste **non précise** et peut donner lieu à plusieurs interprétations.

- **Spécification semi-formelle** : le logiciel est décrit graphiquement avec des **diagrammes** qui peuvent représenter différentes "**vues**/projections" du logiciel (structure, fonctionnalités, temps/comportement). Ces différents modèles graphiques forment ensemble la "vue 3D" **complète** du logiciel.

→ représentation graphique **compréhensible** par le client/utilisateur.

→ la description **syntaxique** est bien précise mais la description **sémantique** reste ambigüe : possibilité de traduction en différente versions du code.

Les classes de spécification/modélisation

- **Spécification formelle** : modélisation du logiciel à l'aide de la **logique mathématique** (calcul des propositions/prédicats) raffiné en **pseudo-code** :
  - qui peut être **traduit** directement sans ambiguïté en langage de programmation.
  - qui peut être **prouvé** directement dés la phase de spécification permettant d'éviter la phase de test après le codage.

→ **Exemples** de méthodes de spécification formelle : Méthode B (1996), Langage Z (1980) ...

→ **Exemples** d'outils d'aide à la spécification formelle : ACL2, AtelierB, CADP, Coq, Murϕ, Prototype Verification System, SMV, nuSMV, zChaff...(`https://fr.wikipedia.org/wiki/M%C3%A9thode_formelle_(informatique)`)

Les classes de spécification/modélisation

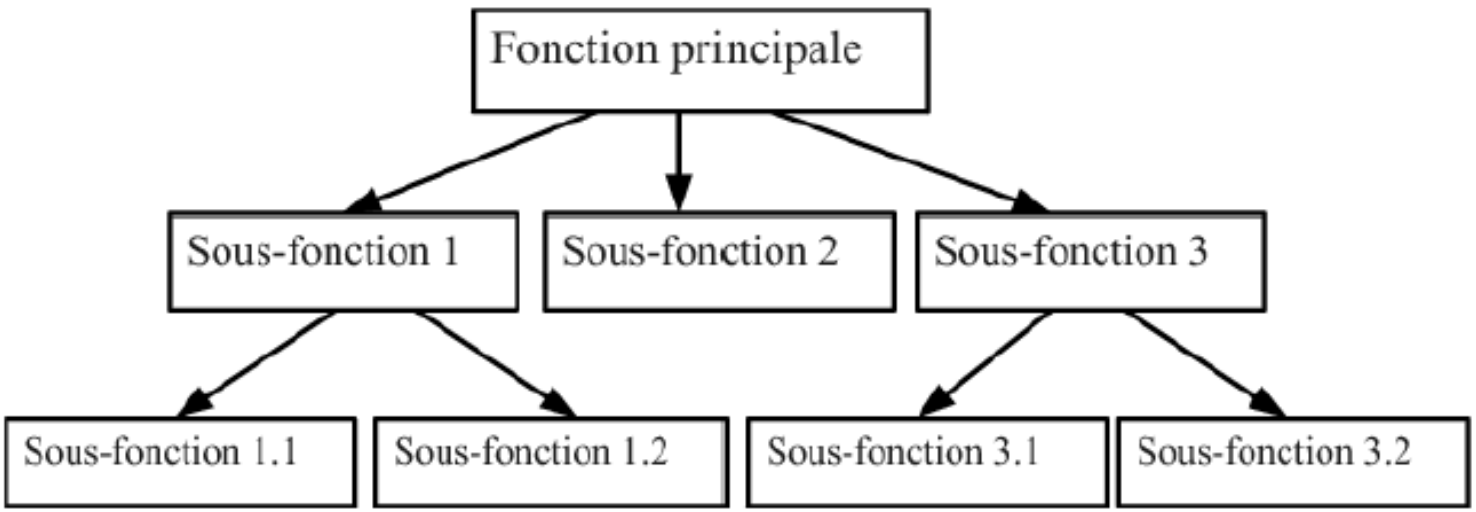
→ La spécification formelle est :

- généralement **coûteuse** en ressources humaines (compétences en logique mathématique et méthodes formelles) et matérielles (outils d'aide payants en général)
- réservée aux logiciels les plus critiques en **sécurité** et en complexité.
- **non compréhensible** par le client/utilisateur.

→ la spécification **semi-formelle** semble la plus adéquate pour la majorité des logiciels.

Les classes de spécification/modélisation semi-formelle

- 1ère **Génération : méthodes de modélisation hiérarchique** : modéliser hiérarchiquement les **fonctionnalités** du logiciel : le plus haut niveau représente la fonctionnalité principale du logiciel qui sera décomposée en sous-fonctions d'une manière **descendante**.

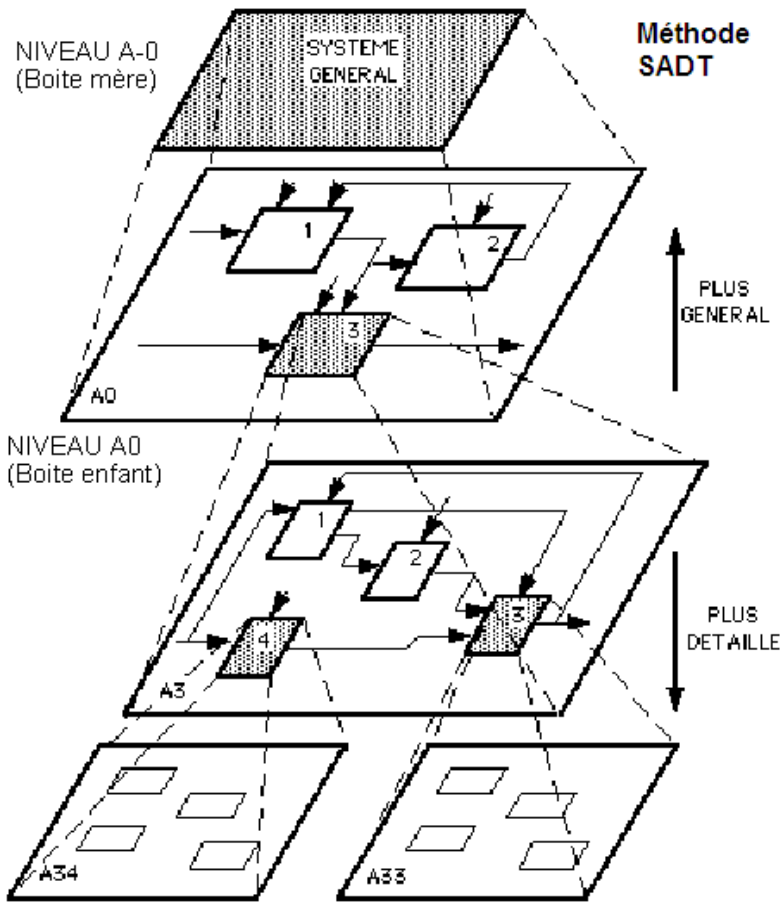


Pierre Gérard, "*Introduction à UML 2 Modélisation Orientée Objet de Systèmes Logiciels*", Université de Paris 13.

Les classes de spécification/modélisation semi-formelle

Exemple de méthodes de modélisation **hiérarchique** :

- **Warnier**/Orr diagram : *Logical Construction of Programs*, 1974
- JSP : ***Jackson*** *Structured Programming*, 1975
- **SADT** : *Structured Analysis and Design Technique*, 1977
- **De Marco** : *Structured analysis and system specification*, 1979
- ...



http://wikimeca.org/index.php?title=M%C3%A9thode_SADT

Les classes de spécification/modélisation semi-formelle

- $2^{ème}$ **Génération : méthodes de modélisation systémiques** :
  - modéliser les données et le traitement séparément avec plusieurs types de **diagrammes complémentaires**.
  - introduire des **niveaux d'abstraction** dans le processus de développement du logiciel (niveau conceptuel, niveau logique et niveau physique).
  - Ce type de méthodes est souvent **spécialisé** pour la conception d'un certain type de systèmes.
  - Ce type de méthodes s'inspirent de la **modélisation des systèmes vivants** (biologiques, sociales...)

Les classes de spécification/modélisation semi-formelle

- $2^{ème}$ **Génération : méthodes de modélisation systémiques** :
  - Cette approche met en valeur les **échanges** entre le système (logiciel) et son **environnement** ainsi que les échanges entre les différentes **parties** du système.
  - Cette approche est influencée par les systèmes de gestion de **bases de données** (utilisation de la notion **entité-association**).
  - Ces méthodes peuvent aller jusqu'à modéliser tout le **cycle de vie** du logiciel.
  - **Exemple** de méthodes concurrentes de modélisation systémique :
    - **AXIAL** : *Analyse et Conception de Systèmes d'Information Assistées par Logiciels* : conçue au sein de IBM France en 1986.

Les classes de spécification/modélisation semi-formelle

- $2^{ème}$ **Génération : méthodes de modélisation systémiques** :
  - **Exemple** de méthodes concurrentes de modélisation systémique :
    - **SSADM** : *Structured Systems Analysis and Design Method* :

  → méthode de modélisation anglaise,
  → conçu depuis 1980,
  → regroupe plusieurs types de diagrammes (y compris le diagramme de modélisation des flux de données DFD largement connu),
  → plusieurs versions se sont apparues jusqu'à l'année 2000, se base sur les méthodes hiérarchiques comme SADT.

Les classes de spécification/modélisation semi-formelle

- $2^{ème}$ **Génération : méthodes de modélisation systémiques** :
  - **Exemple** de méthodes concurrentes de modélisation systémique :
    - **MERISE** : *Méthode d'Etude et de Réalisation Informatique pour les Systèmes d'Entreprise* :

→ inventée en 1978 par H. Tardieu et A. Rochfeld
→ a évoluée jusqu'au 1992 avec Merise 2,
→ la méthode la plus utilisée en informatique de gestion en France et grande partie de l'Europe,
→ introduit un quatrième niveau d'abstraction : le niveau organisationnel,
→ modélisation des traitements à l'aide du formalisme issu des réseaux de Pétri (un modèle mathématique traduit par un graphe orienté modélisant des évènements et des transitions utile pour simuler des système informatique, industriel, biologique...).

Les classes de spécification/modélisation semi-formelle

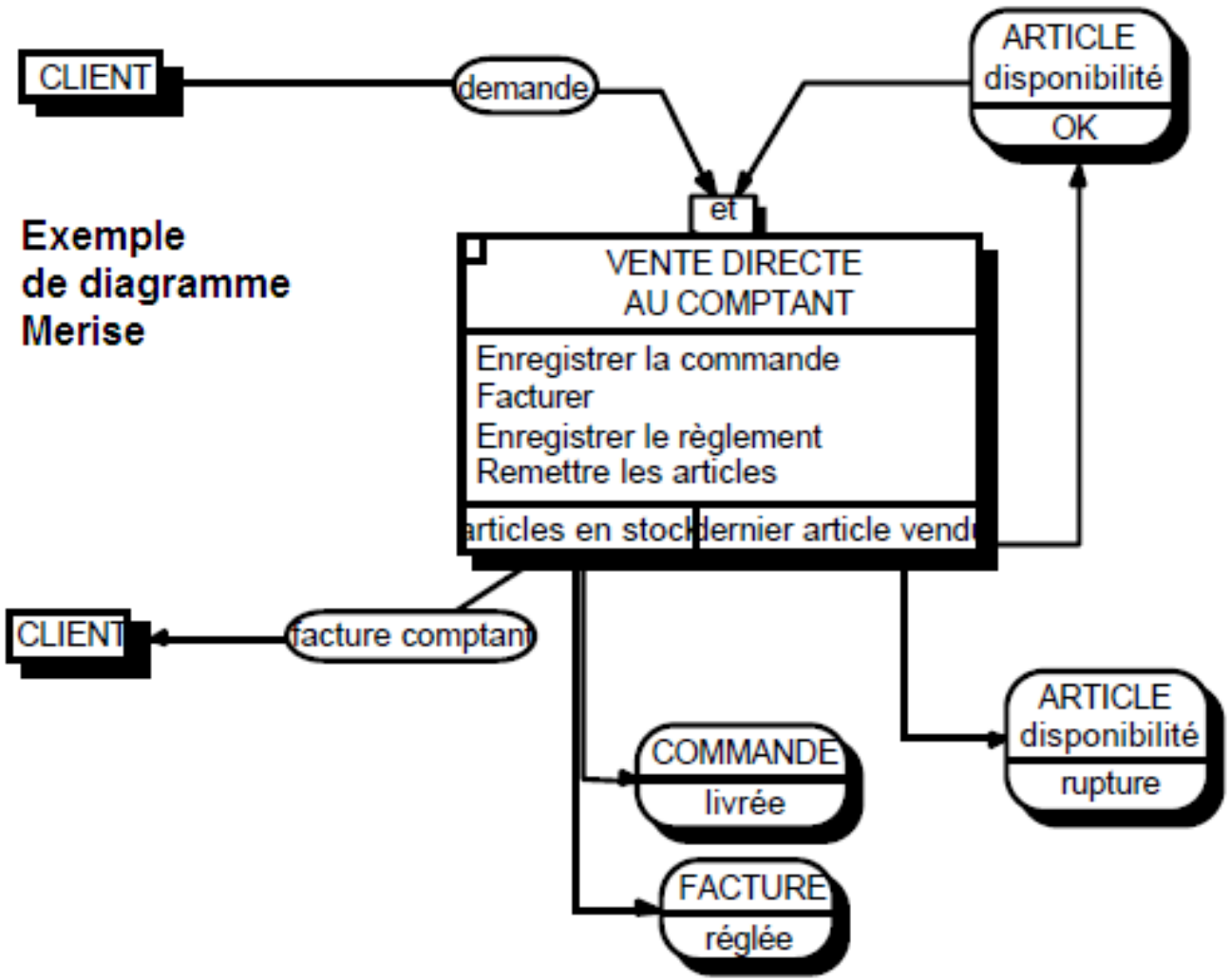
Les Modèles de Merise

	Données	Traitements
SIO — CONCEPTUEL et ORGANISATIONNEL — SYSTEME D'INFORMATION ORGANISATIONNEL	Modèle Conceptuel des Données **MCD** *Signification des informations sans contrainte technique ou économique*	Modèle Conceptuel desTraitements **MCT** *Activite du domaine sans préciser les ressources ou leur organisation*
	Modèle Organisationnel des Données **MOD** *Signification des informations avec contrainte organisationnelles et économique*	Modèles Organisationnels des Traitements **MOT** *Fonctionnement du domaine avec les ressources utilisées et leur organisation*
SII — LOGIQUE — SYSTEME D'INFORMATION INFORMATISE	Modèle Logique des Données **MLD** *Description des données tenant compte de leurs conditions d'utilisation par les traitements*	Modèles Logique des Traitements **MLT** *Fonctionnement du domaine avec les ressources et leur organisation informatiques*
PHYSIQUE	Modèle Physique des Données **MPD** *Description de la ou des bases de données dans la syntaxe du logiciel SGF ou SGBD*	Modèle Physique des Traitements **MPT** *Architecture technique des programmes*

Bernard ESPINASSE, "MERISE : une méthode systèmique de conception de SI", Université d'Aix-Marseille.

Les classes de spécification/modélisation semi-formelle

- Exemple de diagramme Merise au niveau du MCT (*Modèle Conceptuel de Traitement*) :

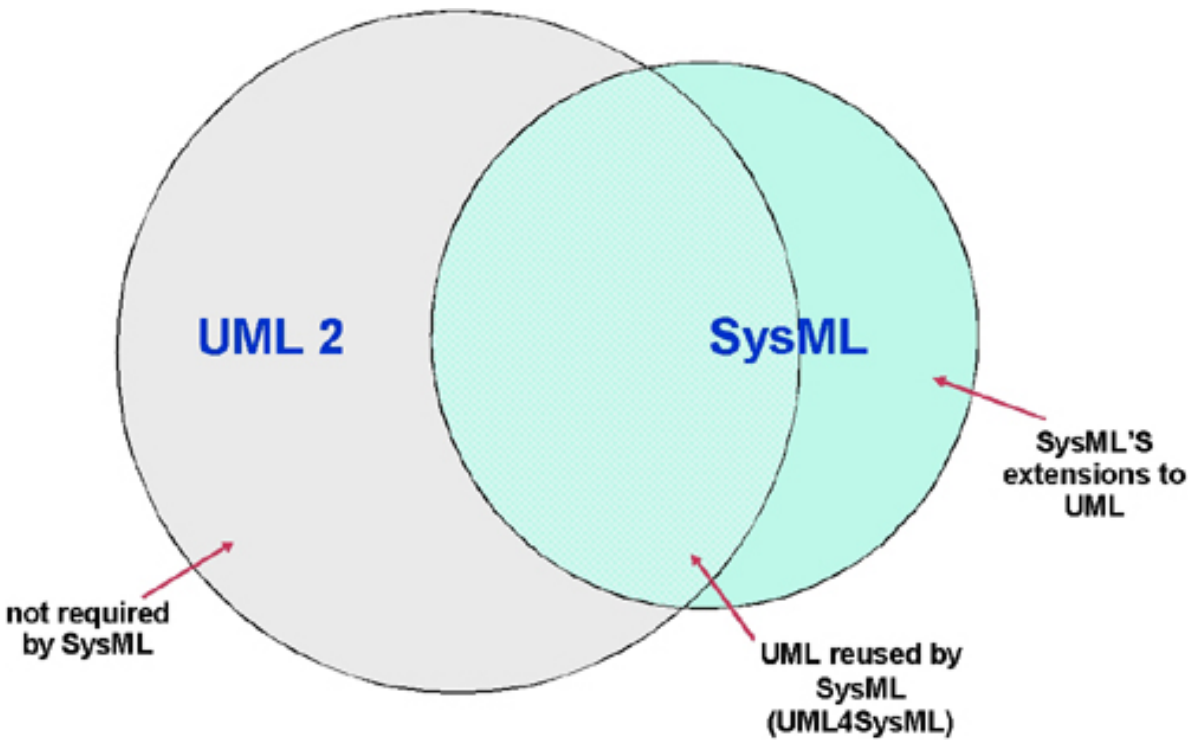


Les classes de spécification/modélisation semi-formelle

- $3^{\text{ème}}$ **Génération : méthode de modélisation orientées objet** :
  - modéliser le logiciel sous forme d'un ensemble d'**objets**.
  - Chaque objet contient des données sous forme d'**attributs** représentant son **état** ou ses propriétés et un traitement sous forme de **fonctions** représentant son **comportement**.
  - Ces objets sont typés par des **classes** et ces classes peuvent être regroupés dans des ***packages*** $\rightarrow$ la modélisation orienté objet est **ascendante**.
  - Ce type de modélisation est généralisable : les composants du logiciel peuvent être **réutilisables**.
  - **Exemple 1** de méthodes de modélisation orientées objet : **UML** (*Unified Modeling Language*, 1997).

Les classes de spécification/modélisation semi-formelle

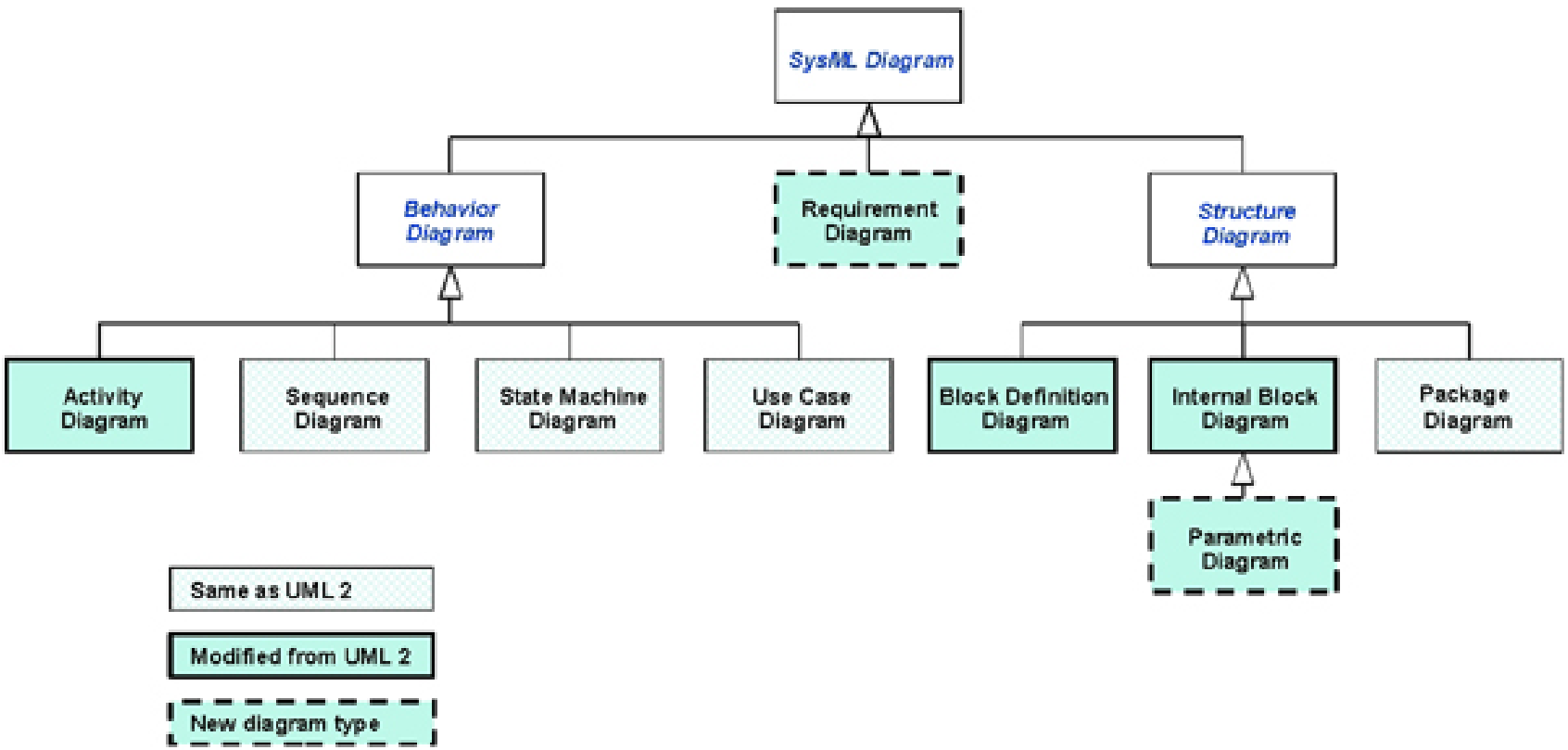
- 3ème **Génération : méthode de modélisation orientées objet** :
  - **Exemple 2** de méthodes de modélisation orientées objet : **SysML** (*Systems Modeling Language*, 2007) : le SysML est un langage de modélisation similaire à l'UML, il reprend certains diagrammes, modifie d'autres diagrammes et ajoute de nouveaux diagrammes par rapport aux diagrammes UML.



https://www.omgsysml.org/what-is-sysml.htm

Les classes de spécification/modélisation semi-formelle

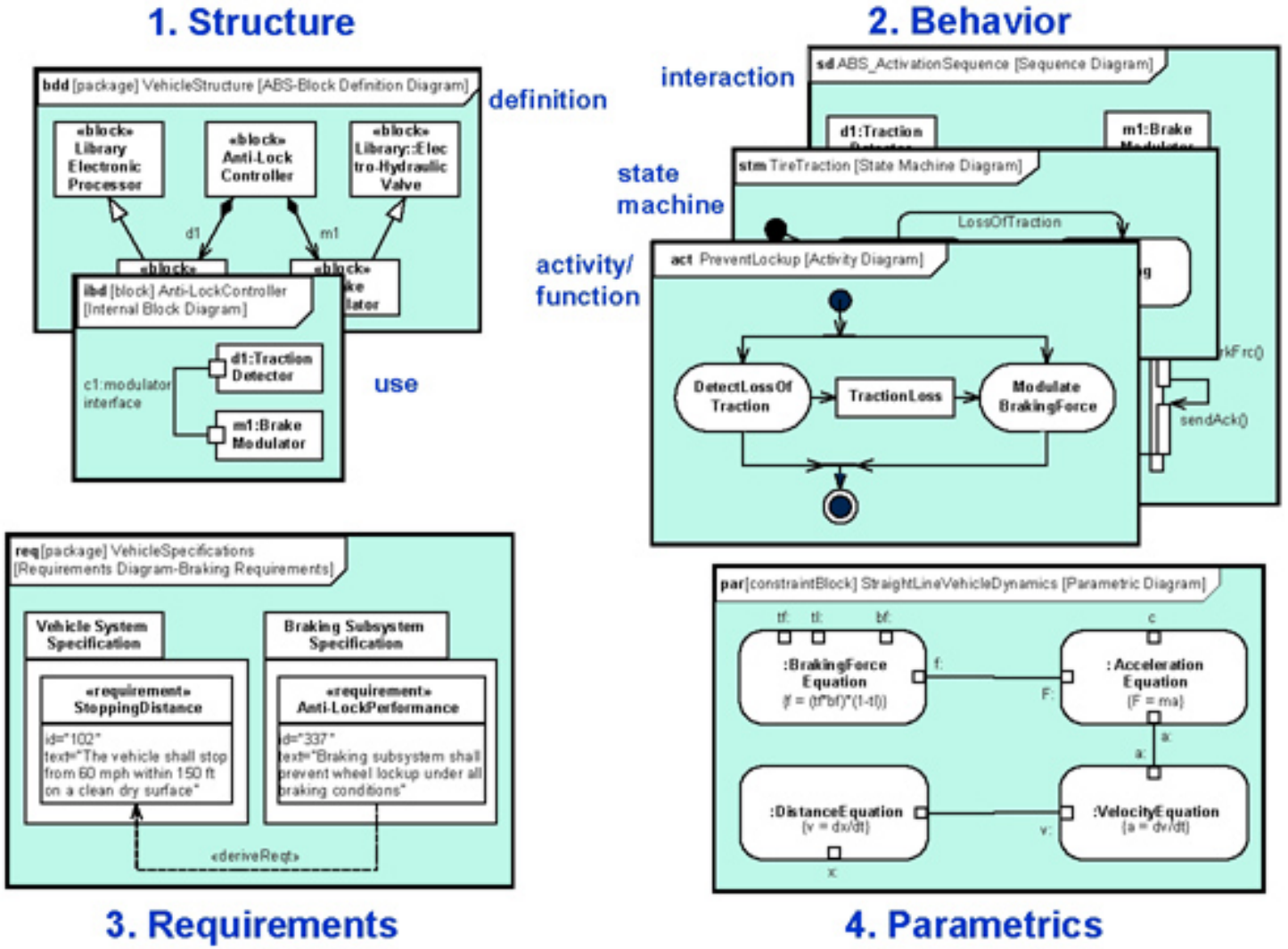
- 3ème **Génération : méthode de modélisation orientées objet** :
  - Le **SysML** est conçu pour modéliser des produits informatiques et non informatiques et peut être généralisé pour modéliser de larges systèmes d'informations complexes.



https://www.omgsysml.org/what-is-sysml.htm

Les classes de spécification/modélisation semi-formelle

- 3ème **Génération : méthode de modélisation orientées objet : SysML**



Note that the Package and Use Case diagrams are not shown in this example, but are respectively part of the structure and behavior pillars

https://www.omgsysml.org/what-is-sysml.htm

Références

- Pierre Gérard, "*Introduction à UML 2 Modélisation Orientée Objet de Systèmes Logiciels*", Université de Paris 13 - IUT Villetaneuse.
- Guillaume Laurent,"*Introduction au génie logiciel*", École nationale supérieure de mécanique et des microtechniques, 2007.
- The Standish Group, "*Chaos Report*", 2014.
- Yassine Jamoussi, "*Introduction au Génie Logiciel*", ENSI-Manouba.
- Djamel-Abdelhak SERIAI, "*Développement de Systèmes Informatiques Orientés objets en UML*", ENSM-DOUAI, Lille.
- Olivier Guibert, "*Cours d'Analyse et Conception des Systèmes d'Information*", Université Bordeaux 1, 2007.
- Bernard ESPINASSE, "MERISE : une méthode systèmique de conception de SI", Université d'Aix-Marseille.
- E. Bourreau, M. Leclère, T. Libourel et C. Nebut, "*Spécification et qualité du logiciel*" LIRMM.
- R.Beltaifa Hajri, "*Outils formels de développement*", ENSI, Laboratoire RIADI.