Module M3301
Partie 2 - Méthodologie de construction du logiciel
Modélisation et construction des logiciels complexes

J. Christian Attiogbé

11/2014, maj 2016

Motivations

Exercice

Construisez un logiciel/système de réservation de ressources (places dans un avion, train, ...).
Le logiciel doit gérer des usagers qui s'y connectent pour réserver/acheter des places initialement disponibles.
A l'épuisement des places disponibles le système s'arrête.
Déclinez ce projet de construction, selon différentes solutions ou architectures de développement
(quelles sont les différentes solutions ?)

Méthodes

Quelles méthodes possibles ? quelles méthodes choisir ?
Quelles technologies ? quelles plateformes (multi-plateformes) ?

Motivations

D'autres exemples de projets de construction de logiciels

- Système de *chat* sur internet
- PaceMaker (embarqué, critique)
- WebService (à la *Amazone*) ; interfacer une appli existante,
- Application Web + Bases de Données
  (architecture classique d'une application BD-Web ?)
- jeux vidéo
- Applications pour appareils mobiles ?
- Contrôle domotique
- Applications pour systèmes embarqués/connectés...
- Projet d'applis scientifiques.

Introduction

Employer les bonnes méthodes pour construire le logiciel qui répond à des besoins spécifiques.

- Quelles méthodes employer pour produire/construire quels logiciels ?
- Quelles techniques ?
- Quels langages ?
- quels outils ?

Le logiciel dans son environnement : système informatique.

Introduction

Avez-vous une intuition de ce que c'est ?

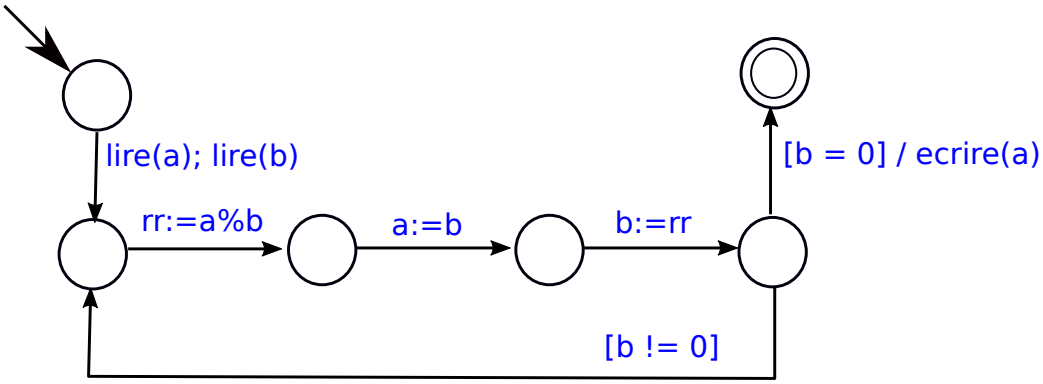


Figure: Exemple de modèle

Introduction

Avez-vous une intuition de ce que c'est ?

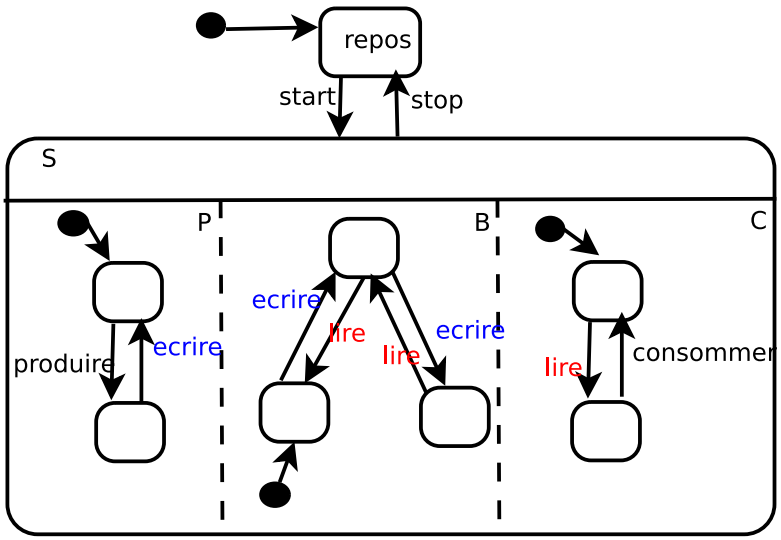


Figure: Exemple de modèle

Software Engineering Models and Methods (anticipons !)

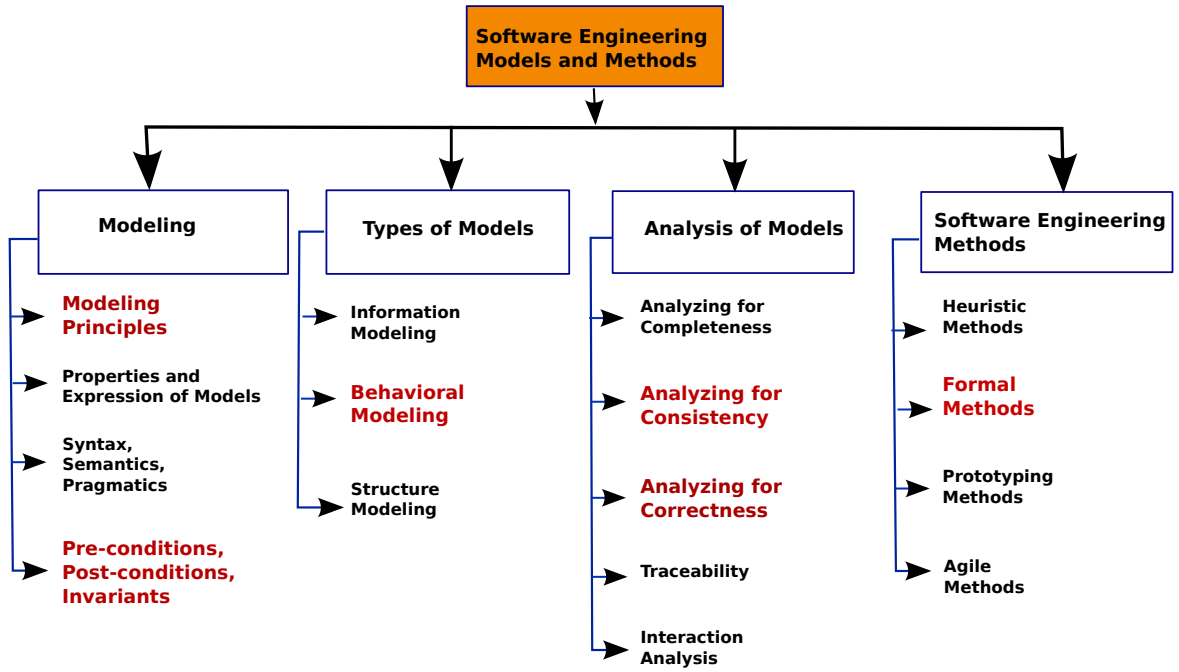


Figure: Software Engineering Models and Methods (source ieee swebok)

Exercices - devoirs de maison

- Etudier (réviser) les automates à états.
- Etudier (réviser) les logigrammes (organigramme de programmation).
- Etudier (réviser) les flux de processus (*workflows*).

Plan du cours

Fondamentaux en ingénierie

Définitions : Exigences, besoins, spécifications, cahier de charges, ...

Besoins, Needs, *Requirement* - côté client

Requirements are the users' description of what the finished product should do.

Spécification des besoins - côté technique

A specification is an explicit set of requirements to be satisfied by the software.
A specification is the technical description of the software, covering the requirements and much more (cost, technicalities, problems, ...).

Fondamentaux en ingénierie

Notions fondamentales

Cahier de charges, Spécifications fonctionnelles, Architecture des applications logicielles, Cycles de vie

Analyse des besoins

Exigences fonctionnelles Elicitation, analyse, spécification, vérification, validation des besoins d'un (projet) logiciel.

Ingénierie système

Décomposition fonctionnelle, Décomposition modulaire, Construction des composants, Assemblage des composants

Conception et réalisation de produits industriels

En sciences de l'ingénieur la démarche scientifique pour réaliser un produit industriel passe par deux principales phases : la conception puis la fabrication du produit.
Les étapes du processus de réalisation sont :

- l'élaboration du cahier des charges fonctionnel,
- l'analyse,
- la conception,
- la réalisation de prototypes,
- la mise au point produit-process,
- l'industrialisation.

Ce sont les mêmes étapes qui sont suivies pour la réalisation de logiciels ou de systèmes contenant du logiciel.

Etapes de conception et de réalisation de produits

- La durée des étapes est plus ou moins longue, selon les contextes, les méthodes, les outils.
- L'analyse des systèmes consiste à décomposer le/s produits en composants (sous-systèmes)
- L'étape de mise au point se fait généralement par simulation et essais.
  Certaines étapes/phases, comme la simulation, sont elles même assistées par ordinateur.
- Lorsqu'on adopte une démarche rigoureuse, bâtie sur des méthodes formelles de modélisation et d'analyse, on allonge le temps de l'analyse conception et on réduit considérablement le temps des mises au point. On idéalise la construction correcte immédiate.

Conception de produit (ou analyse fonctionnelle)

Objectif

L'objectif de la conception est de prévoir ce que sera le comportement du produit lors de son utilisation en conditions réelles.

Réponse d'un produit

Le produit interagit avec son environnement, ce qui entraîne une modification de certaines grandeurs physiques pouvant caractériser soit l'environnement, soit le produit lui-même. Une telle modification est appelée réponse du produit.

Conception

La tâche du concepteur est de déterminer les réponses, ou du moins celles qui font partie des objectifs de l'étude, afin de s'assurer que les performances réalisées seront bien conformes aux performances attendues.

Etude d'un système

Dans les sciences de l'ingénieur, deux approches scientifiques permettent de déterminer et prévoir les réponses et les performances réalisées par un produit, à partir des informations dont on dispose sur l'environnement.

- l'approche par simulation (on travaille dans le virtuel) : elle est basée sur une modélisation mathématique
- l'approche par essai ou mesure (on travaille dans le réel) : consiste à réaliser un prototype/une maquette du produit, le mettre dans les conditions réelles ou identiques, puis de mesurer (en s'aidant d'instruments) les réponses et les modifications des grandeurs de l'environnement.

Diagnostic en conception

La conception est validée lorsque l'écart entre les résultats/comportements/performances attendus et ceux observés est le plus faible possible.
De fortes hypothèses :

- le modèle pour la simulation est fidèle au produit.
- l'essai peut venir confirmer les résultats de la simulation
- les conditions de l'essai sont représentatives des situations réelles

Lorsque l'écart est grand, on doit effectuer un diagnostic pour trouver les causes et les corriger.

Ramassé

- Chaîne : algorithmique, programmation, compilation
  Algorithmique avancée (Cormen &Al ; Knuth, ...)
  Ingénierie de la compilation : logique, langages, automates
- Programmation modulaire/structurée : développement de bibliothèques
- Analyse structurée (SA), fonctionnelle, relationnelle (BD)
- Divers cycles de vie (Cascade/*waterfall*, Spirale ! autour de prototype ; incrémental ; cyce de Balzer) - Sciences de l'ingénieur : cycle d'un produit
- Conception et programmation orientée Objets (1990) : développement de bibliothèques, de langages
- Programmation fonctionnelle
- Réseaux, Internet, Services : Middleware
- Environnements d'ingénierie - IDE - Open Source

Cycle de vie

Cycle de vie = toutes les étapes du développement logiciel, de sa conception à sa construction, sa maintenance, et à son abandon ou disparition.

Le cycle de vie est constitué des étapes suivantes :

- Définition des objectifs et des finalités du logiciel ou du système global comprenant le logiciel
- Analyse des besoins : recueil et formalisation des besoins d'un client, et des contraintes qui sont liées aux besoins et au contexte
- Spécification (*Software requirement analysis document*, le quoi du logiciel à construire), les fonctionnalités, et les contraintes
- Conception générale puis détaillée : architecture globale du projet/logiciel ; puis détail de chaque composant ; c'est le comment.
- Implantation : c'est la phase de réalisation (dans un langage et un environnement précis) du logiciel élaboré lors de la conception.

Etapes du cycle de vie (suite)

- Test unitaire des composants
- Test global (ou d'intégration)
- Validation de la conformité de la réalisation aux objectifs initiaux (spécifications)
- Mise en production : déploiement auprès du client, et validation
- Maintenance (corrective et évolutive)

Plusieurs cycles de vie

Il y a plusieurs cylces de cie du logiciel. Par exemple les cycles :

- cycle en Cascade (1970),
- cycle en V (années 1980),
- cycle en Spirale (Boehm 88),
- cycle de Balzer (Balzer, 1989),
- cycle itératif (1990, utilisé dans les méthodes *RAD Rapid Application Development*)
- Méthodes agiles (issues des pratiques de ces précédents cycles, spirale, RAD, XP)
- etc

Ils sont choisis et appliqués selon les besoins du projet à développer.

Cycles de vie

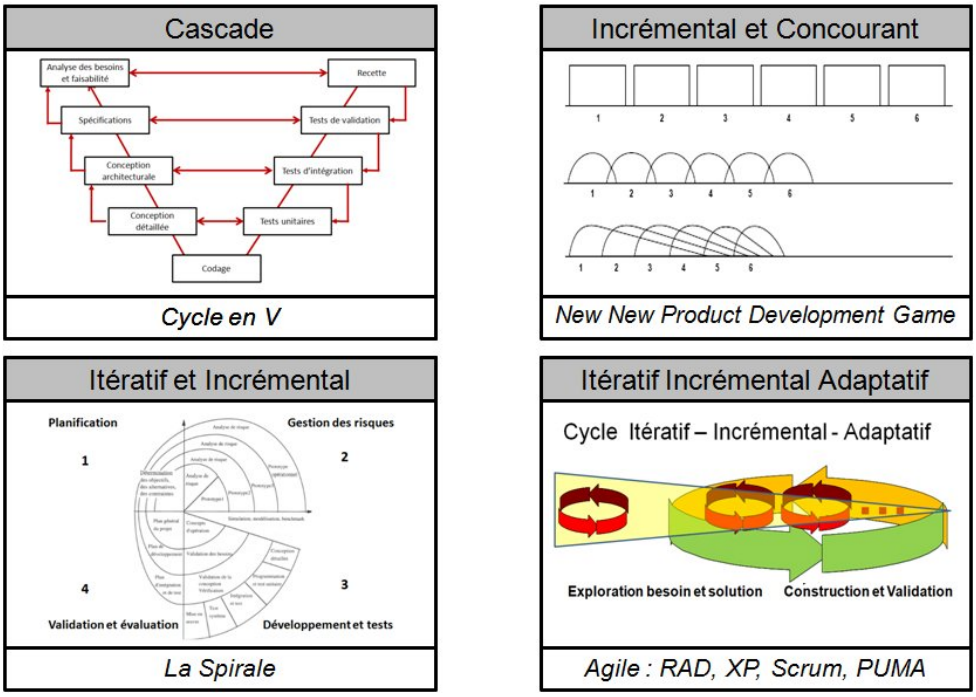


Figure: Cycle de vie (source :wikipedia)

Cycles de vie ou de développement

Tous les cycles de vie classiques ne marchent pas avec les méthodes formelles. Il faut les adapter.
Le cycle de vie de Balzer représente une nouvelle famille de cycles de vie adaptés au développement formel (ou rigoureux).

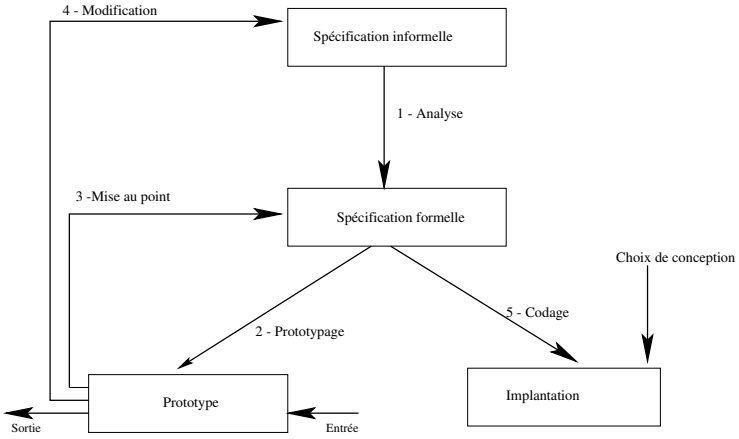


Figure: Cycle de vie de Balzer

[Balzer et al., 1989] Balzer, Robert, Begeman, Michael, Garg Pankaj K., Schwartz, Meyer, and Shneiderman, Ben, Panel Discussion on Hypertext and Software Engineering, Proceedings of Hypertext '89, ACM Press, 1989

Références

- INCOSE, Systems Engineering Handbook: A Guide for System Life Cycle Processes and Activities, version 3.2.2, International Council on Systems Engineering, 2012
- Software Engineering, Ian Sommerville
- Software Engineering Body of Knoweledge (Swebok V3)
- Acm SE 2004 `http://sites.computer.org/ccse/`

Catégories/types des systèmes

Applis	Types
Bases de données, Web, synthèse d'images, calcul numérique, contrôle de procédé industriel, système bancaire, enchères, réservation de ressources, gestion, télécommunication, etc	Systèmes interactifs, réactifs, répartis, transformationnels, embarqués, temps-réels, concurrents, ...

Pour chaque catégorie : des concepts, méthodes, techniques, outils

Catégories/types des systèmes

Exercice : relier les applis et les types !

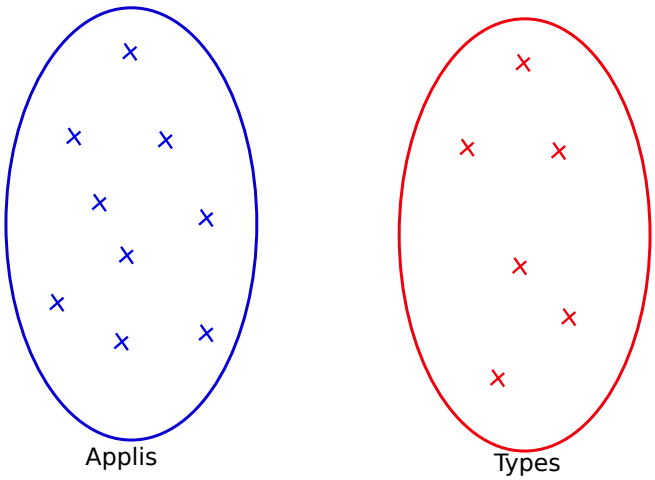


Figure: Relation (à trouver) entre les applis et leurs types

Diverses technologies pour la construction de logiciels

	Technologies
Programmation Objets, composants	C++, J2EE, Scala, Python, etc
Programmation par assemblage de composants, Architecture	Corba, EJavaBeans, .Net, ICE JavaEE, etc
Programmation multi-paradigme	OCaml, Haskell, JavaScript, Scala, etc
Patrons de conception	Spring (frameworks), MVC, CakePHP, etc
Applications Web	PHP, Javascript, Ajax (client) LAMP, WAMP

Linux Apache MySql PhP - Windows Apache MySql PhP

Diverses technologies pour la construction de logiciels

	Technologies
Architectures orientées services	Axis, Java Web Services Development Pack (JWSDP), JBossWS, Rest, BPMN, etc
Outils supports de développement	Les IDE ; ANT, Maven, Sonar (test, qualité), Jenkins, etc
Serveurs d'applications	Java EE, .Net, Tomcat, etc
Ingénierie des modèles	MDE : UML/XML, MOF, QVT, ATL, Acceleo ; MDA, etc
Méthodes et qualité logicielle	agiles (Scrum, ...), CMMI, ITIL, etc
Normes, gouvernance	CMMi, ITIL, etc

Devoir de maison

Se renseigner sur les normes pour la construction de logiciels

Normes	Domaines	Caractéristiques?
IEC 62304	Systèmes embarqués, médical	
IEC 62279	Railway applications	
IEC 61506	Industrial-process measurement and control	

Devoir de maison

- Remplissez le tableau suivant,
  en vous servant des définitions données dans la suite.

Catégories des systèmes informatiques/logiciels

Nature	Caractéristiques?	Méthodes/Langages /Techniques ?
Système séquentiel		
Système autonome *autonomous (transformational)*		
Système centralisé *centralised*		
Système réactif *reactive*		
Système temps-réel *real-time system*		
...		

Catégories des systèmes informatiques/logiciels

Nature	Caractéristiques?	Méthodes/Langages /Techniques ?
système parallèle *parallel system*		
Système parallèle et concurrent *parallel and concurrent*		
Système réparti *distributed*		
Système embarqué *embedded*		
Protocoles communication *communication protocols* ...		

⇒ plusieurs types de systèmes informatiques, nécessité de méthodes variées

Systèmes autonomes transformationnels

Ils transforment leurs entrées en (résultats en) sortie.
Exemples : tri, édition, calculs, planification, etc

- Concepts : séquence, transition d'états, processeur unique
- Modélisation : Algorithmique, automates, hiérarchies fonctionnelles, UML, modèles relationnels, etc
- Etude : Logique, Automate, ...
- Programmation : langages de programmation classiques (impératifs, fonctionnels, objets, ...)

Systèmes concurrents

Un système dans lequel plusieurs tâches (identiques ou différentes) s'exécutent en même temps.
Exemples : producteurs-consommateurs de ressources

- Concepts : processus, concurrence, synchronisation, multi-processeurs
- Modélisation : automates, réseaux de Petri, algèbres de processus
- Etude : automates, logique, preuve, ...
- Programmation : processus, threads, primitives de synchronisation

Communication entre les tâches, accès concurrents aux données, cohérences des données, ...

Systèmes parallèles

Un système dans lequel plusieurs processeurs contribuent à exécuter la même tâche.
Exemples : imagerie, calculs numériques, parallélisation de divers programmes (tri dichotomique, etc, recherche web, multiplication de matrices ...),

- Concepts : multi-processeurs, synchronisation,
- Modélisation : automates, algèbres de processus,
- Etude : logique, automate, ...
- Programmation : primitives, multitache, processus, threads, multi-threads,

Systèmes répartis (*distributed systems*)

Type de systèmes concurrents où plusieurs ordinateurs coopèrent sans un programme central.
Exemples : réservation de places dans le transport, applications/services web, etc

- Concepts : asynchrone, non-déterminiseme, causalité, consensus, messages, événements, mémoire partagée, (cf LAMPORT)
- Modélisation : Algorithmique distribué, Automates communicants, Réseaux de Petri, Algèbres de processus, logiques temporelles
- Etude : automates, réseaux de Petri, algèbres de processus,
- Programmation : Threads, API spécifiques dans les langages

Systèmes temps-réels

Exemples : contrôle de procédés industriels, robotique, aéronautique,

- Concepts : délais, *timeout*, synchronisation, préemption, ordonnancement
- Modélisation : Automates temporisés, moniteurs
- Etude : automates temporisés, réseaux de Petri, ...
- Programmation : *Threads*, C dédié, ...

Systèmes réactifs

Ce sont des systèmes qui réagissent ou répondent continuellement à des événements externes (de leur environnement).
Exemples : système de contrôle (de lumière, présence, etc) ; systèmes de détection d'objets, d'alarme...

- Concepts : événement, signal, réaction
- Modélisation : Automates de Mealy, Statecharts, approche ingénierie système
- Etude : basé sur les automates
- Programmation : langages synchrones (signal, esterel, scade) ; API spécifiques java

Les **systèmes interactifs** sont des cas particuliers de systèmes réactifs.

Systèmes embarqués (*embedded systems*)

Ce sont les systèmes informatiques avec une forte intégration et interaction entre une partie matérielle et une partie logicielle pour effectuer une tache spécifique.
Caractéristiques : faible quantité de ressources de calcul et de stockage, faible consommation d'énergie, taille réduite pour le système (ils sont dédiés à des tâches spécifiques).
Exemples : assistance dans les véhicules, robotique, domotique, appareils médicaux, objets divers (montres, lecteur MP3, etc)

- Concepts : matériel-logiciel, interruption, préemption, contraintes de temps, performance,
- Modélisation : automates [temporsés], Matlab/simulink, Scilab, MARTE, UPPAAL, AADL, Papyrus
- Etude : Logique, automates, etc
- Programmation : Ada, C spécifique, (Environnements VisSim, etc)

Références

- Heath, Steve (2003). Embedded systems design
- ACM SE2014 `https://www.acm.org/education/se2014.pdf`
- Modeling SOftware Software Behabiour, Jorgensen, CRC Press
- Software Engineering Design, Otero, CRP Press
- Embedded Systems Development, Sangiovanni&al, Springer
- *12 Best Software Development Methodologies with Pros & Cons* http://acodez.in/12-best-software-development-methodologies-pros-cons/

Notions de bases: état, transition, comportement

Le comportement d'un système (quel qu'il soit), peut être vu comme une suite des différents états du système.
Les changements d'états ou transitions entre états peuvent être dus à des événements externes au système.

Une **suite d'états décrit un comportement**, donc des transitions entre des états.

Modélisation

En modélisation et en construction de logiciels en particulier, on prévoit le comportement des logiciels. On fait pour cela des **modèles**. On modélise ainsi ce qu'on va construire (aussi bien les les données que les traitements). On peut ainsi prédire les comportements corrects et incorrects (*bugs*) ; il faut pour cela que les modèles soient les plus précis (mathématiquement) possibles.

Un modèle simule mathématiquement le (comportement d'un) système. En ce sens un automate à états sert de modèle d'un système ; il permet d'en simuler le comportement.

Automates à états

Un automate à états est défini formellement par un n-uplet : $\langle S, A, \delta, S_0, S_f \rangle$

- Ensemble d'états fini: $S = \{S_0, \cdots, S_f, \cdots\}$
- Un alphabet d'actions ou d'étiquettes : A
- Une relation de transition δ définie sur $(S \times A)$ et S : $S \times A \leftrightarrow S$
- Un état initial S_0 (élément de S)
- Un ou des états finaux S_f (éléments de S)

Automate déterministe

L'automate est déterministe lorsque la relation de transition est une fonction plutôt qu'une relation, auquel cas il est non déterministe.

Terminologie

- Diagramme Etat/Transition (*State/Transition Diagram*)
- Automate (*Automaton, Automata*)
- Machine à états finis (*Finite State Machine*)
- Système de transitions (étiquetées) *Labelled Transition System*
- Espace d'états (*state space)*

- **Alphabet** d'actions ensemble des actions effectuées par un système
- **Espace d'états** : ensemble des états (possibles) d'un système
- **Relation de transition, Fonction de transition**
- **Graphe d'états**.

Exemple d'automate

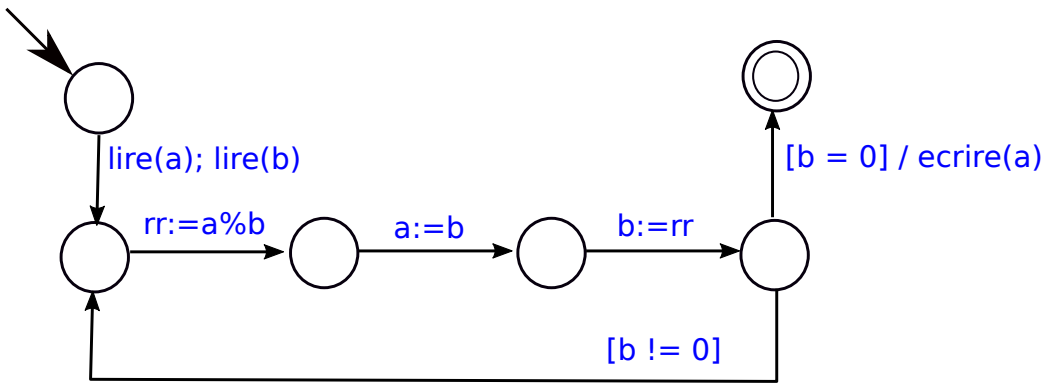
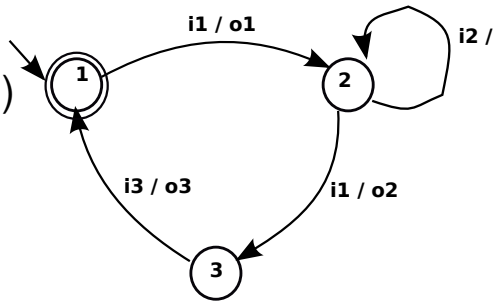


Figure: Exemple de modèle

Machine à états de Mealy

Un automate de Mealy est défini par un n-uplet
$\langle S, A = In \cup Out, \delta, \delta_o, S_0, S_f \rangle$

- Ensemble d'états : S
- Ensemble d'entrées : In
  (événements d'entrées, conditions)
- Ensemble de sorties : Out
  (actions de sortie, traitements)
- Un état initial : S_0
- Deux relations :
  - transition $\delta : S \times In \rightarrow S$
  - sortie $\delta_o : S \times In \rightarrow Out$
- Etat final S_f



Machines de Mealy : notation

- Notation des transitions : $Ss \ \ —i/o \rightarrow \ St$
- si l'entrée i est reçue alors que le système est dans l'état Ss, la sortie o est produite et le nouvel état du système est St
- i est aussi appelé le déclencheur (*trigger*).

La notation des transitions est étendue de la façon suivante : les transitions entre états ont la forme evt [garde] / actions*.
evt est l'entrée reçue/lue.
[garde] : garde est une condition sur l'état du système après l'événement evt.
action* est une suite de 0 ou plusieurs actions.

Caractéristiques d'une machine de Mealy

- **non hiérarchique**
- un état dénote l'état complet du système
- le système est dans **un seul état à la fois**
- une **transition est atomique** ; elle ne peut être décomposée.

Généralisation des machines à états

Il y a plusieurs modèles et familles de modèles à états. Plusieurs langages ou formalismes de modélisation utilisent ces modèles à états.

- **System Design Language (SDL)** (normalisé par la *International Telecommunication Union* (ITU), 1988, très utilisé en Télécoms et ailleurs)
- **Statecharts** de David Harel (hiérarchiques) 1987, utilisé pas exemple dans papyrus, ...
- **Réseaux de Petri** de Carl Adam Petri (1962, 1969)
- **Algèbres de processus**, telles que *Calculus of Communicating Systems* (CCS, 1980) de R. Milner, *Communicating Sequential Processes* (CSP, 1978) de T. Hoare. C'est la famille la plus expressive des modèles à états.

Outils et modèles du génie logiciel

Généralités

Variétés de systèmes et de méthodes

La nature des systèmes complexes oblige le développeur à utiliser
les formalismes,
méthodes,
techniques et
outils appropriés
au bon moment.

Nous allons nous focaliser sur les automates comme base de nombreux formalismes et méthodes

Généralités

- Approche TOP-DOWN (descendant)
  On procède par **décomposition**
  - Analyse globale (étude système, ingénierie de système)
  - Architecture de logiciel
    ↓
  - Codage des composants
    - Programmation directe ou
    - Développement formel (synthèse, raffinement)

- Approche BOTTOM-UP (ascendant)
  On procède par **composition** de composants élémentaires.
  - Etude des composants disponibles
  - Composition, Réutilisation

Mais il faudra modéliser les composants et les construire !

Construction du logiciel

Le terme construction de logiciel (*software construction*) fait référence à la création détaillée d'un logiciel opérationnel à travers la combinaison du codage, de la vérification, des tests unitiares, des tests d'intégration et du déboggage.
La construction de logiciels est étroitement liée aux phases de conception et de test.

Positionnement dans les cycles

Le résultat de la phase de conception (*design*) est l'entrée de la phase de construction (*implementation*).
Le résultat de la phase de constrution est l'entrée de la phase de test (*tesing*).

Software Engineering Models and Methods

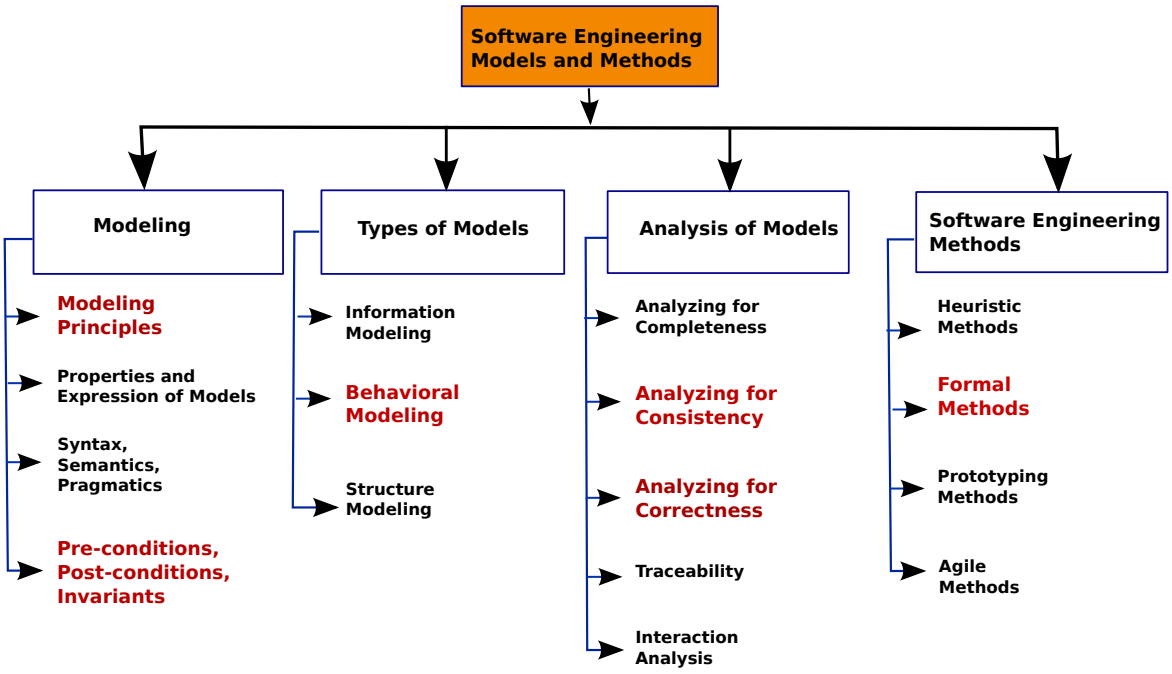


Figure: Software Engineering Models and Methods (source ieee swebok)

Software Construction

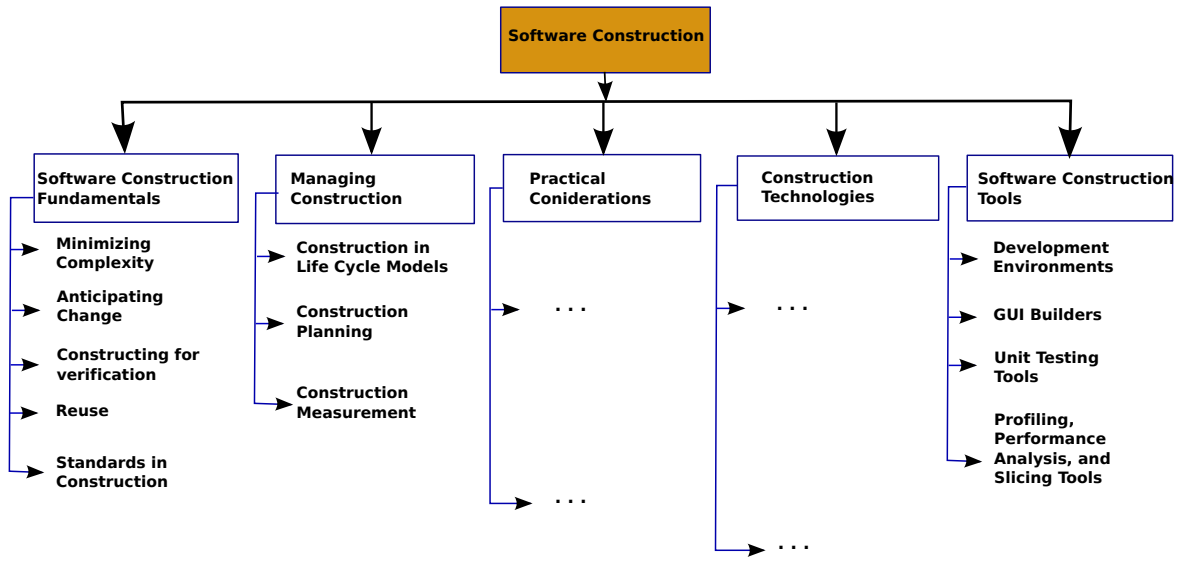


Figure: Software Construction (source ieee swebok)

Software Construction (1st and 2nd blocks)

Software Construction Fundamentals

- Minimizing complexity
- Anticipating Change
- Constructing for verification
- Reuse
- Standards in construction

Managing Construction

- Construction in Life cycle
- Construction planning
- Construction Measurement

Software Construction (3th block)

Practical Considerations

- Construction design
- Construction languages
- Coding
- Construction testing
- Construction for reuse
- Construction with reuse
- Construction quality
- Integration

Software Construction (4th block)

Construction technologies

- API Design and Use
- Object-oriented Run-time issues
- Parameterization and generics
- Assertions, design by contract, defensive programming
- Error handling, Exception handling, fault tolerance
- Executable models
- State-based and Table driven construction techniques
- Run-time configuration and internationalization
- Grammar-based input processing
- Concurrency primitives
- Middleware
- Construction methods for distributed software
- Constructing heterogeneous systems
- Performance analysis and tuning
- Plateform standards
- Test-first programming

Software Construction Tools (5th block)

- Development environments
- GUI builder
- Unit testing tools
- Profiling, performance analysis, slicing tools

Méthodes de construction de composants logiciels

Construction des briques qui seront assemblées.

Méthode B : construction formelle de composanrts et logiciels

Références

- Software Engineering, Ian Sommerville
- Software Engineering Body of Knoweledge (Swebok V3)
- Acm SE 2004 `http://sites.computer.org/ccse/`
- ACM SE2014 `https://www.acm.org/education/se2014.pdf`
- Ihe International Council on Software and Systems Engineering (INCOSE) INCOSE,
  Systems Engineering Handbook: A Guide for System Life Cycle Processes and Activities, version 3.2.2, International Council on Systems Engineering, 2012

Ingénierie système (*Systems Engineering*)

Ingénierie système (*Systems Engineering*)

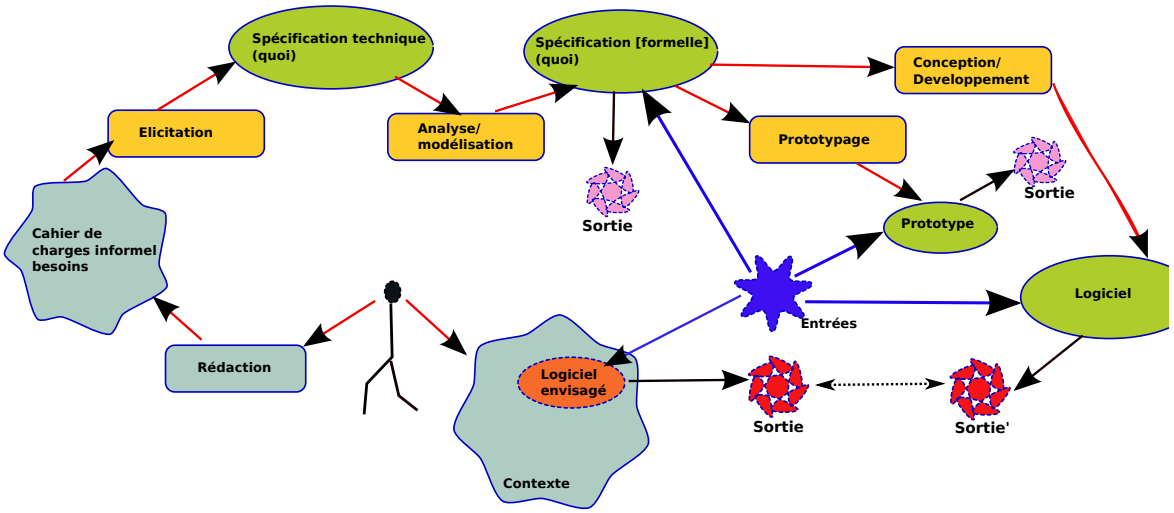


Figure: Construction du logiciel

Ingénierie système (*Systems Engineering*)

Système

Un système est un ensemble de composants en interaction ; le comportement du système dépend non seulement des comportements individuels des composants, mais surtout de l'interaction entre les composants.

- la notion de système se décline en plusieurs niveaux : un composant peut être vu à son tour comme un système et ainsi être décomposé en d'autres sous-composants (sous-systèmes).
- Un composant peut être physique (matériel) ou logiciel.
- L'ingénierie des systèmes offre un cadre adapté à l'étude des produits pluritechnologiques.
- Des langages comme SysML, AADL permettent de décrire un système avec ses composants.

Ingénierie système

Systems Engineering: the Process (*System Engineering*, Boeing)
A process that transforms an operational need or market opportunity into a system description to support detail design.

- Requirements Analysis
- Functional Analysis
- Synthesis
- Systems Analysis / Management

Ingénierie système

Systems Architecture: a Product (*System Engineering*, Boeing)
The design team's interpretation / implementation of customer requirements communicated through:

- System usage scenarios (i.e., Use Cases)
- Functional components & interrelationships
- Physical subsystems & interfaces
- Etc

Ingénierie système

Benefits of Architecting (*System Engineering*, Boeing)

- Identifies all system elementsEarlier vs. Later
- Matches function to requirements
- Capture & communicate key concepts
- Results in one design
- Manages increasing complexity
- Allows modular design

Ingénierie système

Summary

- Increasingly complex systems drive a need for better, clearer design descriptions
- Architectures convey the system designer's interpretation of the requirements
- Architectures may be presented by a variety of views which collectively describe the system
- As part of the systems engineering process, systems architecting defines and manages development of a system

Les méthodes en ingénierie

Domaines variés, tailles variables (petites ou grandes)
Projets informatiques complexes ⇒ méthodes, techniques, outils

- Méthodes d'analyse,
- Méthodes de conception,
- Méthodes de développement,

Il y a des Outils à tous les niveaux.

Méthodes d'analyse

Quelques méthodes semi-formelles

- Analyse fonctionnelle (SADT par exemple),
- Analyse structurée (SA, SSADM), SA-RT (Temps-Réel),
- Entités/Associations, Merise, Axiale,
- JSD/JSP,
- Analyse Orientée Objet, OMT, UML,
- Architecture de logicielle (*System Level*),
- etc

Méthodes : motivations

Besoin de méthodes rigoureuses pour certains domaines :

- Sécurité, Certification, Coût, Maintenance
- ITSEC (Information Technology Security Evaluation Criteria) exigent l'usage de méthodes **formelles**
- Echec (d'un vol) de ARIANE !, Erreur du Pentium, etc, etc
- Milieux hostiles à l'homme (nucléaire, chimie, marin, etc)
- Systèmes embarqués (véhicules, équipements, etc)
- Automates (domaine médical, etc)
- etc

Analyse des exigences (*Software requirements analysis*)

Analyse des exigences

Les exigences du logiciel (*Software requirements*) expriment les besoins et les contraintes liés à un logiciel qui (peut) contribue(r) à la solution d'un problème réel.

On a des exigences sur le produit/logiciel ou des exigences sur la démarche de réalisation.

Une propriété essentielle de toute exigence, est qu'elle soit vérifiable en tant que caractéristique élémentaire, comme exigence fonctionnelle ou comme exigence non-fonctionnelle au niveau global d'un système.

Elicitation, analyse, spécification, vérification, validation des besoins d'un (projet) logiciel.

Fondamentaux

Les exigences fonctionnelles décrivent les fonctions que le logiciel doit fournir.

Les exigences non-fonctionnelles sont celles qui contraignent la solution fournie par le logiciel.

Les propriétés non-fonctionnelles sont souvent vues commes des contraintes ou des exigences de qualité.

Les propriétés non-fonctionnelles peuvent être classées en plusieurs sous-catégories :

- exigences de performance, exigences de maintenabilité, exigences de sureté,
- exigences de fiabilité, exigences de sécurité, exigences d'interopérabilité,
- etc

Sources des exigences et élicitation des exigences

Les sources des exigences d'un logiciel peuvent être nombreuses et variées ;
il faut les identifier pour en faire un recueil approprié (élicitation des besoins).
Il y a différentes techniques d'élicitation des besoins :

- Interviews, scenarios (tels que les *use case de UML*),
- Réunions de groupe, observation, *User stories*, ...
- Prototypes/maquettes

Réaliser ensuite un cahier de charges (spécification informelle des besoins).

Analyse des exigences (*Requirements Analysis*)

L'analyse des exigences permet

- de détecter et résoudre d'éventuels conflits entres les exigences.
- de découvrir les limites du logiciel et comment il interagit son environnement,
- d'élaborer les exigences systèmes pour en dériver les exigences du logiciel.

Les approches classiques d'analyses des exigences intègrent la modélisation (conceptuelle) en utilisant des méthodes telles que l'analyse structurée. On peut y ajouter la classification des exigences.

Classification des exigences (*Requirements Classification*)

- Exigences fonctionnelles ou non-fonctionnelles (délais/temps, sécurité, qualité).
- Exigences sur le produit (logiciel) ou exigences sur la procédure (ITIL, SIL, ...)

Les exigences sur la procédure peuvent contraindre le choix des contractants, la démarche d'ingénierire logicielle à adopter, les normes ou méthodes à utiliser, etc.

Méthodes/Outils d'analyse des exigences

KAOS : *Knowledge Acquisition in automated specification* ou *Keep All Objectives Satisfied*
Tables de Parnas : formalisme rigoureux pour expliciter les besoins
SCR (Software Cost Reduction),
RSML *(Requirements State Machine Language)*

Modélisation conceptuelle (*Conceptual Modeling*)

L'élaboration de modèles pour un problème réel est fondamentale pour l'analyse des exigences du logiciel.

Les modèles aident à comprendre les situations dans lesquelles le problème survient, aussi bien que pour esquisser une solution.

Ainsi les modèles conceptuels comprennent les modèles des entités du domaine du problème, configurés pour reflecter leurs inter-relations et dépendances.

Modélisation conceptuelle (*Conceptual Modeling*)

Les modèles incluent par exemple :
des diagrammes de cas d'utilisation,
des modèles de flots de données,
des modèles à états,
des modèles orientés buts,
des interactions entre usagers,
des modèles à objets, etc.

Parmi les facteurs qui influent sur le choix du formalisme de modélisation : il y a notament la nature du problème.

Analyse formelle

L'expression formelle des exigences requiert l'emploi d'un langage avec une sémantique précise.
Le recours à l'analyse formelle des exigences permet :

- de les spécifier précisement et sans ambiguités évitant ainsi les mauvaises interprétations
- de pouvoir raisonner sur les exigences, de prouver les propriétés du logiciel spécifié.

L'analyse formelle requiert des spécifications formelles comme point de départ.

Des considérations pratiques

Le traitement des exigences couvre tout le cycle de vie du logiciel.

- Changement des besoins initiaux
- Maintenance des exigences dans l'état qui réflettent le logiciel à construire, ou du logiciel qui est construit.
- Tracabilité des exigences.

Software Requirements Tools

- Tools for modeling
- Tools for managing requirements.

References

- Parnas's Tables
  https://cs.uwaterloo.ca/~jmatlee/talks/parnas01.pdf
- Beck, A., Boeing, G., & Shannon, D. *Systems and Methods for Analyzing Requirements. US Patent 8650186*, 2014
- Chemuturi, M. (2013). *Requirements Engineering and Management for Software Development Projects*, ISBN 978-1-4614-5376-5, 2013
- Software Development Process–activities and steps
  http://www.uacg.bg/filebank/acadstaff/userfiles/publ_bg_397_SDP_activities_and_steps.pdf