Slide 1

Software Design

(Guide to the SWEBOK)

Guy Tremblay
Dépt. d'informatique
UQAM

MIG8500 — Hiver 2002

Slide 2

Aperçu

1. Qu'est-ce que le *Software Design*?
2. Comme le *SD Knowledge Area* du guide est-il organisé?
3. Pourquoi ce *KA* est-il organisé ainsi?
4. Quelles sont les limites de cette description du *Software Design*?
   Comment se compare-t-elle avec d'autres descriptions?

Slide 3

1 Définition du *Software Design*

Software Design = Conception logicielle

Définition du *design* selon l'IEEE (1990)

- "*the process of defining the architecture, components, interfaces, and other characteristics of a system or component*"
- "*the result of that process*"

Donc, la *conception* peut être vue autant comme

- un produit = description de l'architecture du système, comment il est décomposé en composants indépendants, l'interface et le comportement de ces composants
- un processus = étape du cycle de vie où les exigences et spécifications sont analysées dans le but de produire une description de l'organisation interne d'un système

Slide 4

En termes d'un processus classique de développement (*ISO/IEC 12207 Software life cycle processes*), on retrouve ainsi les étapes suivantes :

1. Conception architecturale
2. Conception détaillée

Rôle clé de la conception = génèrer des *modèles* qui

- décrivent la solution avant qu'elle soit mise en oeuvre (*blueprints*)
- pouvent être analysés et évalués
- pouvent servir comme base pour les tests
- sont utilisables pour la planification des étapes subséquentes

Slide 5

Portée du *Software Design KA* (terminologie de De Marco):

D-design = Decomposition design = décomposition d'un composant en sous-composants

FP-design = Family Pattern design = identification des aspects communs à un groupe de logiciels

I-design = Invention design = conceptualisation et spécification d'un système dans le but d'identifier les besoins et exigences

Le KA n'inclut pas non plus, de façon spécifique, de concepts liés aux items suivants:

- Conception de systèmes en temps réel ;
- Conception d'interfaces personne-machine ;

Slide 6

2 Description du *Software Design KA*

Vue d'ensemble :

I. Concepts de base ;

II. Enjeux fondamentaux ;

III. Structure et architecture ;

IV. Analyse et évaluation de la qualité ;

V. Notations ;

VI. Stratégies et méthodes ;

Slide 7

I. Basic concepts

Concepts formant une *fondation* pour comprendre le rôle et la portée du SD :

- Concepts généraux sur la conception
- Le contexte du SD (dans le cycle de vie)
- Le processus de SD (architecturale vs. détaillée)
- *Enabling techniques* pour le SD :
  - Abstraction
  - Couplage et cohésion
  - Décomposition et modularité
  - Encapsulation et dissimulation de l'information
  - Interface vs. mise en oeuvre
  - *Sufficiency, completeness and primitiveness*

Slide 8

II. Key issues in *Software Design*

Key issue = enjeu clé = aspect global du système qui va au delà d'une simple fonctionalité

Semblable à la notion moderne d'*aspect* :

- "*some aspect of the system's behaviour that is not in the application domain, but which adresses some of the supporting domains*"
- "*aspects cross-cut the system's functionality*"

En d'autres mots, un *aspect* est quelque chose qui ne correspond pas de façon directe à une fonctionalité ou à un composant spécifique du système. Un *aspect* a plutôt un effet sur de nombreux composants du système.

Slide 9

Quelques enjeux clés :

- Concurrence
- Contrôle et traitement des événements
- Distribution
- Traitement des exceptions
- Interactions
- Persistence

Ces différents aspects sont rarement abordés de façon explicite dans les livres modernes sur la conception (sauf par Meyer et dans les livres spécialisés portant sur l'un de ces aspects)

Cause : absence de maturité du domaine?

Slide 10

III. Software Structure and Architecture

Au sens premier du terme: *une* architecture logicielle (*a software architecture*) = une description des sous-systèmes et composants formant *un* système logiciel ainsi que des inter-relations entre ces divers éléments.

Mais, le champ du *Software Architecture* s'est élargi pour devenir un champ quasi-indépendant, qui étudie les structures et les architectures de logiciels de façon plus générique et abstraite.

Les architectures peuvent décrire différents niveaux d'abstraction ou de généralisation . . .

- Famille de programmes (lignes de produits) ;
- Styles architecturaux (macro-architecture) ;
- Patrons de conception (micro-architecture) ;

Slide 11

III-1. Structures et vues architecturales

Différentes vues (structures) d'un logiciel doivent être décrites et documentées. On ne peut donc pas parler de "*la structure*" d'un logiciel ;(

Chaque structure :

- représente "un aspect partiel d'une architecture logicielle qui décrit certaines propriétés spécifiques d'un système".
- est une abstraction par rapport à certains critères.
- est un *blueprint* indépendant, avec ses propres notations, styles, etc.

Exemples:

- Vues logique, physique, des processus, de développement ;
- Vues comportementale, fonctionnelle, structurale, de modélisation des données ;
- . . .

Encore dans ce cas: pas de consensus sur les vues à décrire

Slide 12

Les principales structures selon Bass, Clements et Kazman 1998 ("Software Architecture in Practice")

Structure	**Composants**	**Utilisations**
Module	*Work assignment*	Allocation des ressources, planification, etc.
Conceptuel	Fonctions	Compréhension du problème
Processus	Programmes	Ordonnancement, analyse de performance
Physique	Matériel	Performance, sécurité, disponibilité
Utilisations	Programmes	Sous-ensembles et extensions
Appels	Programmes	Profil de performance, goulot d'étranglement
Flux de données	Tâches fonctionnelles	*Traceability*
Flux de contrôle	Modes ou états	Simulation
Classes	Objets	Gabarits

Slide 13

III-2. Styles architecturaux (*macro*-architecture)

Style architectural

= “ensemble de contraintes sur une architecture qui définit un ensemble ou une famille d’architectures qui satisfont ces contraintes”.

= méta-modèle qui décrit l’organisation de haut niveau (macro-architecture) d’un logiciel.

Exemples:

- Général: en couches, pipelines et filtres, *blackboard*, etc. ;
- Systèmes distribués: client-serveur, à trois-*tiers*, etc. ;
- Systèmes interactifs: MVC, PAC ;
- Systèmes adaptables: micro-noyau, réflection ;
- Autres: en lot, interpréteur, à base de règles ;

Slide 14

III-3. Patrons de conception (*micro*-architecture)

Patron de conception = “*a common solution to a common problem in a given context*”

- Style architectural = patron de haut-niveau (macro-architecture)
- Patron de conception = patron de bas niveau (micro-architecture)

Exemples:

- Patrons de création: usine, prototype, singleton, etc. ;
- Patrons structurels: adapteur, pont, composite, décorateur, façade, mandataire, etc. ;
- Patrons comportementaux: commande, interpréteur, itérateur, observateur, état, stratégie, visiteur, etc. ;

Slide 15

III-4. Familles de programmes et cadres de conception

Famille de programmes

= *ligne de produits* logiciels (*software product line*)

= ensemble de systèmes partageant un nombre de propriétés en commun, mais aussi avec certaines différences :

- *Commonalities* = ce qui est commun aux différents membres de la famille ;
- *Variabilities* = ce qui varie entre les différents membres ;

Cadre de conception = *framework*

= sous-système logiciel partiellement complet qui peut être étendu en instantiant divers points chaud (*hot spots*, *plug-ins*)

= *un* mécanisme particulier permettant de réaliser une famille de produits

Slide 16

IV. Software Design Quality Analysis and Evaluation

- Attributs décrivant la qualité d'une conception (*ilities*) ;
- Analyse de la qualité et outils d'évaluation :
  - Revues de conception ;
  - Analyses statiques ;
  - Simulation et prototypage ;
- Mesures :

  Pour estimer, de façon quantitative, divers aspects de la taille, structure ou qualité d'une conception :
  - Mesures pour approches fonctionnelles (structurées) ;
  - Mesures pour approches OO ;

Slide 17

V. Software Design Notations

Diverses notations et techniques (micro-méthodes) utilisées pour la description de différents aspects de la conception :

- Descriptions structurales

  = structure et organisation des composants et de leurs inter-relations

  => vue statique.

- Description comportementales

  = description du comportement (*behavior*) des composants ou de leur interactions (protocoles)

  => vue dynamique.

Slide 18

Descriptions structurales :

- ADL (*Architecture Description Languages*) ;
- Diagrammes de classes et d'objets ;
- Diagrammes de composants ;
- Cartes CRC ;
- Diagrammes de déploiement ;
- Diagrammes entités-relations ;
- IDL (*Interface Definition Language*) ;
- Diagrammes de Jackson ;
- Diagrammes hiérarchiques ;

Slide 19

Descriptions comportementales :

- Diagrammes d'activités ;
- Diagrammes de collaboration ;
- Diagrammes de flux de données ;
- Tables et diagrammes de décision ;
- Organigrammes ;
- Langages formels de spécification ;
- Pseudo-code et PDL (*Program Design Language*) ;
- Diagrammes de séquences ;
- Diagrammes de transitions et *Statecharts* ;

Slide 20

VI. Software Design Strategies and Methods

Stratégies vs. méthodes :

- Une stratégie aide à guider le processus de conception, souvent de façon assez générale (heuristiques).
- Une méthode est plus spécifique et fournit habituellement :
  1. Un ensemble de notations à utiliser avec la méthode ;
  2. Une *prescription* du processus à suivre pour utiliser la méthode ;
  3. Un ensemble de règles et d'heuristiques qui aident à utiliser la méthode ;

Rôle clé d'une méthode = mécanisme de *transfert de connaissances*

1. Stratégies générales : raffinements successifs, diviser-pour-régner, dissimulation de l'information, etc.
2. Conception structurée (à la Yourdon et Constantine) : analyse des transformations vs. des transactions, heuristiques sur le *fan-in/fan-out*, sur la portée de l'effet vs. du contrôle, etc.;

Slide 21

3. Conception orientée-objets : abstraction de données, approche basée sur les responsabilités, etc.
4. Conception basée sur les structures de données (méthode de Jackson) ;
5. Autres méthodes : méthode rigoureuse de développement, transformationnelle, . . . ;

3 Justifications (*Rationale*)

- Connaissances généralement acceptées . . . dans une fenêtre de 3–5 ans :
  => Inclusion des styles architecturaux et lignes de produit ;
- Indépendant de tout domaine d'applications :
  => rien de spécifique sur les systèmes en temps réel ;

Slide 22

- Indépendant de toute technologie :
  => pas de mention explicite de UML . . . même si la plupart de ses éléments sont présents ;
  – "Notations" vs. "Stratégies et méthodes"
- *As inclusive as possible* :
  => Présentation de certaines méthodes moins connues en Amérique du Nord (par ex., Jackson), ou moins à la mode (conception structurée) ;
  – "*Key issues in SD*" ;

- Utilisant la terminologie généralement acceptée :

  Une exception importante mais intéressante : *Enabling techniques* ;

  "*To enable* = "*to give the means to do something*"

Autres caractéristiques :

- Présentation des notations de façon indépendante des méthodes ;
- Présentation des méthodes selon diverses grandes classes, indépendantes des stratégies générales ;
- Pas de discussion des éléments suivants :
  - *User interface design* ;
  - *Database design* ;
  - *Participatory design* ;
  - *Collaborative design* ;

Slide 23

4 Critiques et comparaisons

4.1 Critiques

Certains points mentionnés par les réviseurs :

- Pas de discussion des systèmes en temps réel ;
- Pas de discussion de la conception des interfaces personne-machine ;
- Il faudrait ajouter la référence "XXX" ;
- Pas de discussion explicite de la *conception détaillée* ;

Slide 24

Slide 25

Les points *faibles*, selon mon opinion :

- *II. Key Issues* : incomplète, un peu confuse = état de l'art ; (
- *III. Software Structure and Architecture* :
  - Présentation ne suivant pas les niveaux d'abstraction ;
  - Pas de consensus sur les différentes vues à décrire ;
  - Notion de *framework* au même niveau que *program family* ;
  - Rien sur "*component-based design*" ;
  - = Domaine du *FP-design* (*Family Pattern*) encore embryonnaire ; (
- Pas de place précise pour la conception détaillée et procédurale (*Related discipline*?) ;

. . . mais voir la conclusion

Slide 26

4.2 SWEBOK vs. SWE-BOK (1999)

SWE-BOK = April 1999, Software Engineering Institute.

2.2 Software Design

- 2.2.1 *Architectural Design* : développement d'une structure modulaire et représentation des relations de contrôle existant entre les modules ;
- 2.2.2 *Abstract Specification* : production des spécifications abstraites des services fournis par chaque module et des contraintes sous lesquelles le module doit opérer ;
- 2.2.3 *Interface Design* : conception et documentation de l'interface entre les sous-sytèmes et entre le système et l'usager ;
- 2.2.4 *Data Structure Design* : spécification et conception des structures de données à utiliser dans la mise en oeuvre d'un logiciel ;
- 2.2.5 *Algorithm Design* : spécification et conception détaillée des algorithmes utilisés pour réaliser les fonctionalités et services logiciels ;

Slide 27

4.3 SWEBOK vs. “Software Design” (D. Budgen, 1994)

I. The role of software design

1. *The nature of the design process*
2. *The software design process*
3. *Design in the software development process*
4. *Design qualities*
5. *Expressing ideas about a design*
6. *Some design representations*
   - *DFD*
   - *ERD*
   - *Structure chart*
   - *Structure graph*
   - *Jackson structure diagram*
   - *Pseudocode*
   - *State transition diagram*
   - *Statechart*
   - *Petri net*

Slide 28

II. Design practices

7. *The rationale for a method*
8. *Design strategies*
9. *Jackson structured programming*
10. *Structured system analysis and structured design*
11. *Jackson system development*
12. *Object-oriented design and object-based design*
13. *Some other systematic approaches to design*
14. *A formal approach to design*
15. *The evolution of software practice*

5 Conclusion

The "*Guide to the SWEBOK*" est un guide au corpus des connaissances, c'est-à-dire, un ensemble de pointeurs *vers* les connaissances, *pas le corpus lui-même*.

SWE-BOK (SEI 1999) et "*Software design*" (Budgen 1994) me semblent moins complets, mais (force et faiblesse) peut-être plus directement utilisables pour un premier ou deuxième cours de conception.

Slide 29

Donc :

- Le *Software Design KA* semble fournir une bonne description de l'"état de l'art" en 2001 :)
- L'état de l'art est encore incomplet et immature :(
- La structure d'un cours (même avancé) sur la conception ne devrait pas être basée, de façon directe, sur la structure du *Software Design KA* ;(