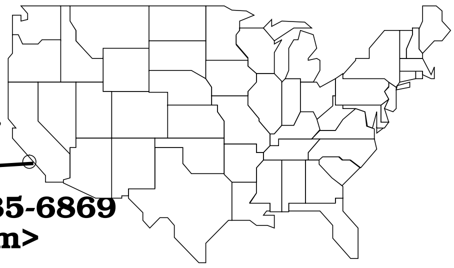
CE QUE TOUT MANAGER DOIT SAVOIR SUR LA RÉUTILISABILITÉ

Bertrand Meyer

ISE Inc.



Adresse de l'auteur:

Interactive Software Engineering Inc.
270 Storke Road, Suite 7
Santa Barbara, CA 93117 USA
Téléphone 805-685-1006, Fax 805-685-6869
Courrier électronique <info@eiffel.com>

FTP anonyme: eiffel.com — WWW: http://www.eiffel.com

CE QUE TOUT MANAGER DOIT SAVOIR SUR LA RÉUTILISABILITÉ

RÉFÉRENCES

Cette présentation est en partie fondée sur les deux livres suivants:

- ***Object Success: A Manager's Guide to Object Technology and its Impact on the Corporation***

  Bertrand Meyer, Prentice Hall, 1995

- ***Reusable Software: The Base Object-Oriented Libraries***

  Bertrand Meyer, Prentice Hall, January 1995

Expérience support: construction de milliers de composants logiciels réutilisés depuis 1974.

EXPÉRIENCE SUPPORT

Simula 67: Programmation par objets, quelques classes réutilisables.

Bibliothèques numériques: IMSL, Linpack, NAG etc.

Conception et implémentation des bibliothèques ISE Eiffel:

- **Base – Structures de données et algorithmes fondamentaux. Environ 150 classes.**
- **Vision – Graphique et interfaces utilisateur, sous forme portable (indépendante de la plate-forme). Environ 550 classes.**
- **Lex, Parse – Analyse lexicale et syntaxique de langages formels.**
- **Store – Accès portable (indépendant de la plate-forme) aux bases de données; support pour les objets rémanents («persistants»).**
- **Math — Bibliothèque mathématique (intégration, équations aux dérivées partielles, calcul matriciel...) utilisant à fond les concepts objets et Eiffel, s'appuyant de façon interne sur NAG.**

(Environ 2000 classes, largement réutilisées dans de nombreuses applications.)

Également: bibliothèques d'application développées en collaboration avec des clients.

PRENDRE LA RÉUTILISABILITÉ AU SÉRIEUX

La production de logiciel apparaît aujourd'hui, dans l'échelle de l'industrialisation, quelque part en dessous des industries de la construction les plus rétrogrades. Elle me semble pouvoir prétendre à une place beaucoup plus élevée, et je me propose [dans cet article] d'étudier les perspectives offertes par les techniques de production de masse en logiciel.

...Quand on entreprend d'écrire un compilateur, on commence par se poser la question «Quel mécanisme de gestion de tables allons-nous écrire?»; au lieu de: «Quel mécanisme allons-nous utiliser?».

...Ma thèse est que l'industrie du logiciel repose sur des bases fragiles [en partie à cause de] l'absence d'une sous-industrie de composants... Une industrie de composants pourrait avoir un succès immense.

Construction de composants en masse
(Mass-Produced Software Components)
Doug McIlroy (ATT Bell Laboratories)
Conférence OTAN sur le Génie Logiciel, Garmisch-Partenkirchen, 1968.

Pourquoi la réutilisation n'a-t-elle pas réussi à plus grande échelle?

LA RÉUTILISABILITÉ: POURQUOI?

Avantages attendus:

- **Diminution des efforts et coûts de développement, amélioration de la productivité. (C).**
- **Diminution des efforts et coûts de maintenance (C).**
- **Fiabilité (C).**
- **Efficacité (C).**
- **Interopérabilité (C / P).**
- **Engranger la connaissance et le savoir-faire collectifs de l'entreprise. (P).**

Les deux aspects de la réutilisabilité:

- **Vue du consommateur (C).**
- **Vue du producteur(P).**

CE N'EST PLUS UN RÊVE

La réutilisation existe en pratique, en partie grâce à la technologie objet.

Mais encore très loin de ce qu'elle devrait et pourrait être.

PRINCIPE 1:

Le produit

La réutilisation en logiciel
est la réutilisation du code.

LES DEUX MYTHES DE LA RÉUTILISABILITÉ EN LOGICIEL

Mythe 1:

- **Les aspects logistiques (bases de données de composants, systèmes de recherche et d'accès, etc.) constituent les principaux problèmes de la réutilisabilité.**

Mythe 2:

- **Un obstacle majeur à la réutilisabilité est le syndrome du NII (*Non Inventé Ici).***

Exemples de décisions prises sur la base de ces mythes:

- **Récompenser les réutilisateurs (!).**
- **"Évangélistes de la réutilisation".**
- **Décréter que tout programme sera réutilisable.**

PRINCIPE 2: CONSOMMATEURS ET PRODUCTEURS

Les problèmes difficiles en réutilisabilité apparaissent du côté des producteurs, non des consommateurs.

Une politique de réutilisabilité qui se concentre sur les consommateurs est une perte de temps.

APPRENDRE AVANT DE SAUTER

Être consommateur avant de devenir producteur.

La réutilisabilité au sens du producteur s'apprend par imitation et des efforts importants.

Piles ou consommateurs?

Les «objets d'affaires» (*business objects*) viendront ensuite.

Le principe du pragmatisme en réutilisation:

NE PAS MÉPRISER LES PILES!

PRINCIPE 4: LES PRIORITÉS

Le but d'une politique de réutilisabilité est de satisfaire les *consommateurs* (les réutilisateurs potentiels des composants réutilisables) en agissant sur les *producteurs* (les auteurs de ces composants) pour qu'ils effectuent le meilleur travail possible.

PRINCIPE 4: LE CHAMP D'ACTION

La réutilisabilité est avant tout un problème technique

Pour le point de vue adverse: voir Will Tracz, *Confessions of a Used Program Salesman: Institutionalizing Software Reuse*, Addison-Wesley, 1995.

UNE POLITIQUE DE RÉUTILISABILITÉ

Peut être initialement modeste, mais doit exister.

Règles:

- **Il doit y avoir une politique d'inclusion dans la bibliothèque, avec des critères d'acceptation précis.**
- **Il doit y avoir un responsable de la réutilisabilité.**
- **Combinaison de composants externes et internes (consommateur / producteur).**
- **Extraire des produits à partir des projets.**
- **Généraliser.**

Critères pour un responsable de la réutilisabilité:

- **Doit avoir une solide expérience de la programmation.**
- **Doit avoir l'expérience des bibliothèques réutilisables.**
- **Très bien organisé.**
- **Capable d'écrire une partie de la documentation.**

LA GESTION D'UNE BIBLIOTHÈQUE DE COMPOSANTS

Deux notions distinctes:

- **Bibliothèque.**
- **Répertoire.**

Les critères d'acceptation pour la bibliothèque doivent être stricts.

Traiter la bibliothèque comme un projet logiciel.

«Taxe à la réutilisabilité».

LES TÂCHES DU RESPONSABLE DE LA RÉUTILISABILITÉ

(et de l'équipe support)

1. Définir les critères de qualification.

2. Appliquer le processus de qualification.

3. Aider les candidats à affiner et améliorer leurs propositions.

4. Maintenir les composants acceptés.

INNÉ OU ACQUIS?

Les arguments de l'école a priori (inné):

- **La réutilisabilité ne peut pas être ajoutée après coup.**
- **Les seuls composants réutilisables sont ceux qui sont nés réutilisables.**

Les arguments de l'école a posteriori (acquis):

- **Aucun logiciel n'est réutilisable la première fois.**
- **Un logiciel n'est pas réutilisable s'il n'a pas été réutilisé.**

Laquelle de ces deux opinions est la bonne?

NIVEAUX DE RÉUTILISABILITÉ

0. Utilisé dans un système.

1. Utilisé dans plusieurs systèmes construits par la même personne.

2. Utilisé dans plusieurs systèmes construits par la même équipe, la même société, ou des équipes en liaison étroite avec les développeurs.

3. Utilisé dans plusieurs systèmes construits par des équipes qui sont en contact avec les développeurs.

4. Utilisé par des équipes inconnues des développeurs.

LA GÉNÉRALISATION:

DES ÉLÉMENTS DE PROGRAMME

AUX COMPOSANTS LOGICIELS

Pour obtenir la qualité:

- **Le faire tourner.**
- **Le faire tourner vite.**
- **Le faire tourner sous peu de mémoire.**
- **Le rendre général.**

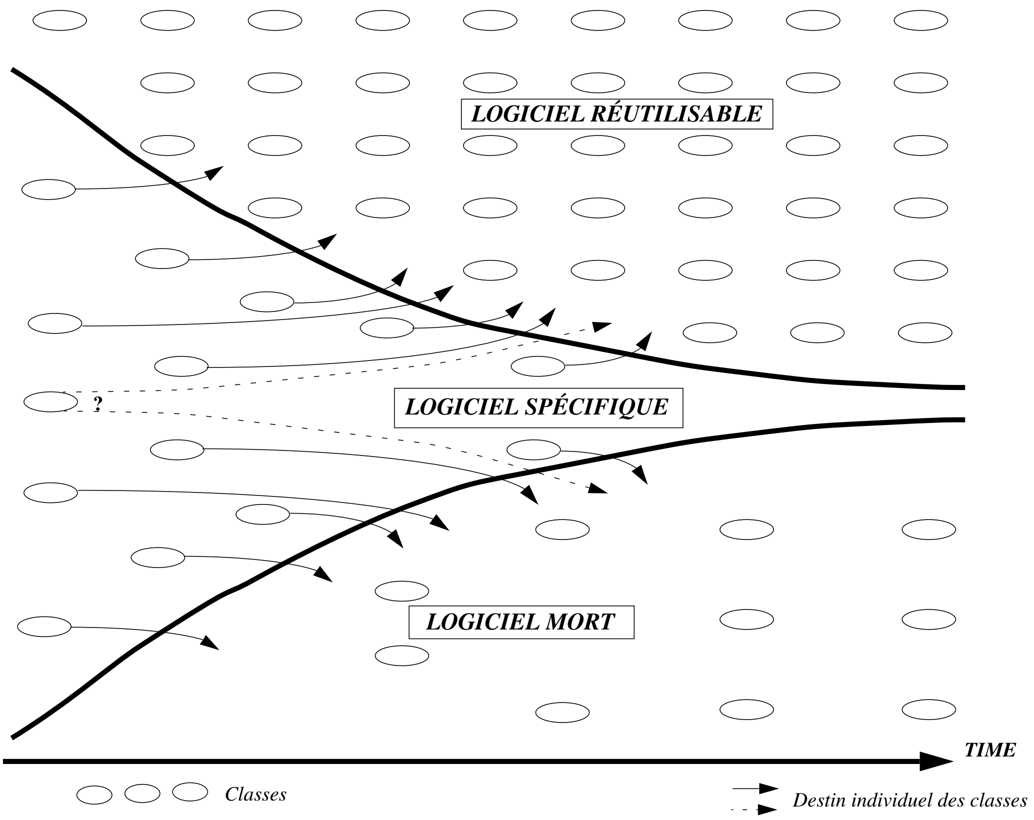
Résister à la tentation de spécialiser.

Nommer un gestionnaire de la réutilisabilité.

Définir des critères d'inclusion dans la bibliothèque.

La réutilisabilité est une culture.

LE MONDE ET SON AVENIR



CE QUE TOUT MANAGER DOIT SAVOIR SUR LA RÉUTILISABILITÉ

PRINCIPE 5:

La technique

GENERALISEZ!

L'inutile

Standards définis par des comités

CONDITIONS POUR UNE INDUSTRIE DE COMPOSANTS RÉUTILISABLES

(et pourquoi cela ne se produira pas avec C++ ou en Smalltalk).

Structure modulaire stricte. Pas de fuites.

Spécifications précises par assertions (préconditions, postconditions, invariants): contrats.

Cohérence entre les assertions et l'héritage.

La liaison dynamique comme seule sémantique des appels

Mécanismes automatiques pour l'élimination de code mort, l'expansion en ligne etc.

Mécanisme d'indexation intégré et flexible.

Mécanismes d'abstraction.

Efficacité similaire ou supérieure en moyenne à celle des solutions non réutilisables sur le marché.

CONDITIONS (suite)

Typage statique, sans tricherie.

Covariance.

Héritage multiple.

Solide base de composants standardisés (cf. ELKS en Eiffel).

Résolution des conflits de noms pour les primitives ET les class.

Superbes outils de fouinage (browsing).

Possibilité de cacher les implémentations de certaines classes tout en publiant la spécification (le contrat).

Ramassage de miettes impeccable, rendant effectivement la mémoire au système d'exploitation.

Gestionnaire de licenses.

Domaine d'application parfaitement compris, avec une théorie.

Démarche totalitaire quant au style de conception et de programmation, aux conventions de passage des arguments, au nommage des classes et des primitives, au masquage de l'information, aux commentaires visibles, à l'emploi de l'héritage, à la méthodologie...

Le projet le plus passionnant aujourd'hui:

CONSTRUIRE UNE BIBLIOTHÈQUE D'APPLICATION

fondée sur un bon environnement à objets