Introduction

PDL

1 - Introduction

Abstract

Definition

Table des matières

1. L’ingénierie logicielle : définitions et concepts fondamentaux

L’ingénierie logicielle se distingue par la nature abstraite et intangible des systèmes qu’elle produit. Comme l’explique Ian Sommerville : *“Les systèmes logiciels ne sont pas contraints par les propriétés matérielles, ni gouvernés par des lois physiques ou des processus de fabrication”*. Cette caractéristique permet une grande liberté créative, mais entraîne également une complexité potentiellement incontrôlable.

1.1. Définitions clés

L’ingénierie logicielle peut être définie comme :

- Une discipline d’ingénierie concernée par tous les aspects de la production logicielle
- Une méthode pour résoudre des problèmes réels via des solutions logicielles
- Une approche permettant de spécifier, construire, distribuer et maintenir des logiciels de qualité, à coûts contrôlés et dans des délais garantis

1.2. Qualités du logiciel

Qualités internes :

- Maintenabilité (corrective, adaptative, perfective)
- Réparabilité
- Évolutivité
- Réutilisation
- Portabilité
- Interopérabilité

Qualités externes :

- Correction
- Fiabilité
- Robustesse
- Performance
- Ergonomie

2. La notion de projet en ingénierie logicielle

2.1. Contraintes économiques

Contrairement aux projets de construction où les coûts sont standardisés (environ 800 CHF/m³ en Suisse), les projets logiciels sont principalement contraints par les coûts, qui sont difficiles à estimer. Dans le cycle de vie d'un bâtiment, environ 20% des coûts sont liés à la construction et 80% à la maintenance. De façon similaire, la maintenance d'un logiciel est souvent plus coûteuse que son développement initial.

2.1.1. Coût d'une ligne de code

Si l'on considère qu'un développeur :

- Coûte environ 200′000 CHF par an
- Code en moyenne 20 lignes productives par jour
- Est absent 1/5 du temps
- Travaille 250 jours ouvrables par an

On peut estimer un coût par ligne de code, bien que cette métrique soit simpliste et ne reflète pas la complexité réelle du développement logiciel.

```
// Calcul simplifié du coût par ligne de code
let cout_developpeur_annuel = 200000; // CHF
let jours_travailles = 250 * (1 - 1/5); // jours effectifs
let lignes_par_jour = 20;
let lignes_par_an = jours_travailles * lignes_par_jour;
let cout_par_ligne = cout_developpeur_annuel / lignes_par_an;

// Résultat : environ 50 CHF par ligne de code
```

2.2. Diversité des projets

Il n'existe pas de processus universel en ingénierie logicielle en raison de la grande diversité des projets :

- Logiciel embarqué
- Logiciel concurrent
- Logiciel distribué
- Logiciel graphique
- Logiciel de traitement des données
- Logiciel de simulation

3. Les échecs des projets en ingénierie logicielle

3.1. Petits projets vs grands projets

Petits projets :
- Cahier des charges court et facile à comprendre
- Coût faible et facile à calculer
- Conception mentale
- Développement par une personne
- Peu de planification et de communication
- Dépendance forte au développeur

Grands projets :
- Cahier des charges volumineux et incomplet
- Méconnaissance du métier du client
- Coûts difficiles à évaluer
- Choix technologiques périlleux
- Travail en équipe
- Documentation indispensable

3.2. Exemples d'échecs notables

- **FoxMeyer** : 4ème distributeur de médicaments aux USA en 1993, faillite en 1996 due à l'incapacité de son nouveau système ERP à gérer le volume de transactions
- **Equifax** : Faille de sécurité majeure due à une vulnérabilité connue mais non corrigée
- **Volkswagen** : Logiciel frauduleux détectant les tests d'émissions

```
// Exemple illustratif du problème Volkswagen
function mesureEmissions(mode) {
  if (detectTestEnCours()) {
    // Activer le système de réduction des émissions
    return reductionEmissions();
  } else {
    // En conditions normales, priorité aux performances
    return emissionsReelles(); // Bien au-dessus des limites légales
  }
}
```

3.3. Statistiques d'échecs

Selon le Chaos Report du Standish Group :
- En 1994 : 31% des projets arrêtés sans résultats, 52% avec dépassements majeurs, 16% de succès
- En 2015 : amélioration, mais toujours des taux d'échec significatifs

4. Causes des échecs en ingénierie logicielle

4.1. La loi de Brooks

“Ajouter des personnes à un projet en retard le rend encore plus en retard”

Cette loi s’explique par :
- Le temps d’adaptation nécessaire aux nouvelles personnes
- L’augmentation exponentielle des canaux de communication entre les membres de l’équipe

```
// Illustration des canaux de communication
function calculCanaux(n) {
  // Formule : n(n-1)/2
  return (n * (n - 1)) / 2;
}

// Pour 3 personnes : 3 canaux
// Pour 5 personnes : 10 canaux
// Pour 10 personnes : 45 canaux
// Pour 50 personnes : 1225 canaux
```

4.2. Causes selon Ian Sommerville

1. **Complexité croissante du logiciel** : Les nouvelles techniques nous permettent de construire des systèmes plus grands et plus complexes, mais les exigences évoluent constamment.
2. **Méthodes d’ingénierie logicielle largement ignorées** : De nombreuses entreprises développent des logiciels sans utiliser de méthodes formelles.

4.3. La complexité selon John Ousterhout

La complexité est définie comme “tout ce qui rend un système logiciel difficile à comprendre et à modifier”.

Deux approches pour combattre la complexité :
1. **Éliminer la complexité** en rendant le code plus simple et plus évident
2. **Encapsuler la complexité** pour que les développeurs n’y soient pas exposés en permanence

```
// Exemple d'encapsulation de la complexité
class GestionDonnees {
  // Interface simple exposée au reste du système
  function recupererDonnees(id) {
    return this.processComplexeInterne(id);
  }

  // Complexité isolée dans des méthodes privées
  private function processComplexeInterne(id) {
    // Vérification de cache
    // Connexion à la base de données
    // Gestion des erreurs
    // Transformation des données
    // etc.
  }
}
```

5. Solutions en ingénierie logicielle

5.1. Évolution historique des méthodes

- **Programmation structurée** (1966) : Théorème de Böhm-Jacopini, “Go To Statement Considered Harmful” de Dijkstra
- **Software engineering** (1968) : Terme introduit lors de la conférence OTAN
- **Software development life cycle** (années 1960)
- **Analyse et conception structurée**
- **Méthodes de développement globales** : OMT, Booch, Fusion… puis UP
- **Programmation orientée objet**
- **Méthodes Agiles**

```
// Exemple de programmation structurée vs non structurée
// Non structuré (avec GOTO)
function calculNonStructure() {
  let total = 0;
  let i = 1;
debut:
  if (i > 10) goto fin;
  total += i;
  i += 1;
  goto debut;
fin:
  return total;
}

// Structuré (avec boucle)
function calculStructure() {
  let total = 0;
  for (let i = 1; i <= 10; i++) {
    total += i;
  }
  return total;
}
```

6. L’éthique du logiciel

6.1. Importance sociétale

L’ingénierie logicielle est devenue essentielle au fonctionnement :

- Des gouvernements
- Des sociétés
- Des entreprises
- Des institutions

6.2. Principes éthiques

- Le génie logiciel ne se résume pas à l’application de connaissances techniques
- Les ingénieurs doivent faire preuve d’honnêteté et se comporter de manière éthique
- Un comportement éthique va au-delà du respect des lois

6.3. Questions éthiques majeures

- Confidentialité
- Compétences
- Propriété intellectuelle
- Utilisation malveillante

```
// Exemple d'implémentation éthique de collecte de données
function collecterDonnees(utilisateur) {
  // Vérifier le consentement explicite
  if (!utilisateur.aConsentement()) {
    return null;
  }

  // Minimiser les données collectées
  let donneesMinimales = extraireDonneesNecessaires(utilisateur);

  // Anonymiser les données sensibles
  return anonymiser(donneesMinimales);
}
```