Une Introduction à la Compréhension des Logiciels

Abdelhak-Djamel Seriai

Parmi les sources utilisées :

- S. Rugaber « Program Comprehension ».
- U. Akhlaq et al. « Impact of Software Comprehension inSoftware Maintenance and Evolution »
- M. Ward « Program Comprehension ».
- M.-A. Storey « Theories, Methods and Tools in Program Comprehension:Past, Present and Future »

Définitions

- Compréhension du programme = Compréhension du code source.
- Est le processus :
  - D'examen et de compréhension d’u système.
  - D'acquisition de connaissances sur un programme informatique.
  - De développement de modèles mentaux de l'architecture, du sens et du comportement des systèmes logiciels.

Les besoins

- Emergence de grands systèmes logiciels composés de millions de lignes de code:
  - Les développeurs ont de plus en plus de mal à avoir la compréhension nécessaire pour réaliser efficacement leurs tâches de développement.
  - Pendant la maintenance et l'évolution, les ingénieurs logiciels consacrent 60 à 90% de leur temps à la compréhension du programme.
- La plupart des activités d'ingénierie logicielle, comme la réutilisation, la refactorisation et les tests, impliquent le besoin de comprendre comment le code fonctionne, comment les modules interagissent, quel code implémente quelles exigences, etc.

Objectifs de la compréhension du programme

- Récupérer des informations (Modèle) de haut niveau sur un système. Ceci inclut :
  - Sa structure : les composants et leurs interrelations.
  - Ses fonctionnalités : quelles opérations sont effectuées et sur quels composants.
  - Son comportement dynamique : comment l'entrée est transformée en sortie.
  - Sa raison d'être : comment a été le processus de conception et quelles décisions ont été prises.
  - Sa construction, ses modules, sa documentation et ses suites de tests.

Modèles statiques et dynamiques des logiciels

- Un modèle statique est une collection d'informations qui peut être découverte en examinant le code source du système et des documents connexes.
    - Généralement valables pour toutes les exécutions possibles du programme.
- Un modèle dynamique est toute collection d'informations qui peut être découverte en exécutant le logiciel et en examinant les données d'entrée, les données de sortie, les traces d'exécution et toutes les autres données produites par l'exécution du programme.
    - Peuvent uniquement être valides pour l'entrée ou l'ensemble d'entrées particulier utilisé pour générer les données.

Pourquoi la compréhension du programme est-elle difficile?

- Besoin de combler différents écarts entres domaines .
  - L'écart entre un problème dans un domaine d'application et une solution dans un langage de programmation.
  - L'écart entre le monde concret des machines physiques et des programmes informatiques et le monde abstrait des descriptions de conception de haut niveau.
  - L'écart entre la description cohérente et hautement structurée souhaitée d'un système telle qu'initialement envisagée par ses concepteurs et le système réel dont la structure peut s'être désintégrée au fil du temps.
  - L’écart entre le monde hiérarchique des programmes et la nature associative de la cognition humaine.
  - L'écart entre l'analyse ascendante du code source et la synthèse descendante de la description de l'application.

Pourquoi la compréhension du programme est-elle difficile?

- Certains facteurs peuvent rendre la compréhension encore plus compliquée:
  - Le code d'un système hérité est dans de nombreux cas très spécifique et non générique.
  - Le code peut contenir des erreurs.
  - Le résultat du processus d'ingénierie inverse peut inclure de nouvelles erreurs.

Approches de compréhension de programme

- Approche descendante:
  - Commencer par les concepts de domaine de problème les plus abstraits et tenter de les mapper sur le code source.

- Approche ascendante:
  - Se concentrer sur la compréhension du comportement de petits morceaux de code et, plus tard, combiner ces informations en plus grandes abstractions.

- Approche opportuniste:
  - Le programmeur / mainteneur bascule de haut en bas et de bas en haut pendant le processus de compréhension. La commutation dépend de la connaissance initiale.

Approches de compréhension de programme: approche descendante

- Essaie de reconstruire des correspondances entre le domaine de problème et le domaine de programmation :
  - Le programmeur crée des hypothèses basées sur les connaissances acquises ou existantes pour arriver à une compréhension.
  - Les hypothèses sont vérifiées par rapport au code source pour prouver leur validité.
  - Les balises sont des endroits dans le code source qui prouvent ou falsifient une hypothèse.
- Un exemple d'hypothèse de haut niveau est: «Ce programme produit des factures». Cette hypothèse mappe le domaine métier (facturation) au domaine de programmation (le programme lui-même).

Approches de compréhension de programme: approche ascendante

- Généralement utilisé lorsqu'on n'est pas familier avec le code / l'application
  - Rechercher des idiomes (patterns/patrons/motif) reconnaissables dans le code
    - Exemple.
      - Le motif : "swap" *t: = x; x: = y; y: = t;*
      - Le motif "accumulation":

while $F(i)$ **do**
    total := total + $A[i]$;
    $i := i + 1$ **od**;

- Combiner des unités identifiées pour comprendre des sections de code toujours plus grandes: par exemple, reconnaître que le «swap» fait partie d'un processus de «tri».

Approches de compréhension de programme: Approche opportuniste

- Les programmeurs basculent fréquemment entre les approches descendantes et ascendantes:
  - Exemple : Commencez du haut vers le bas, obtenez un aperçu des fonctions du programme. Puis, appliquez sélectivement des stratégies ascendantes lorsque on s'approche du niveau du code.
  - Utiliser l'information issue de l'analyse ascendante pour vérifier les hypothèses résultant de la lecture descendante.

Compréhension du programme et outils

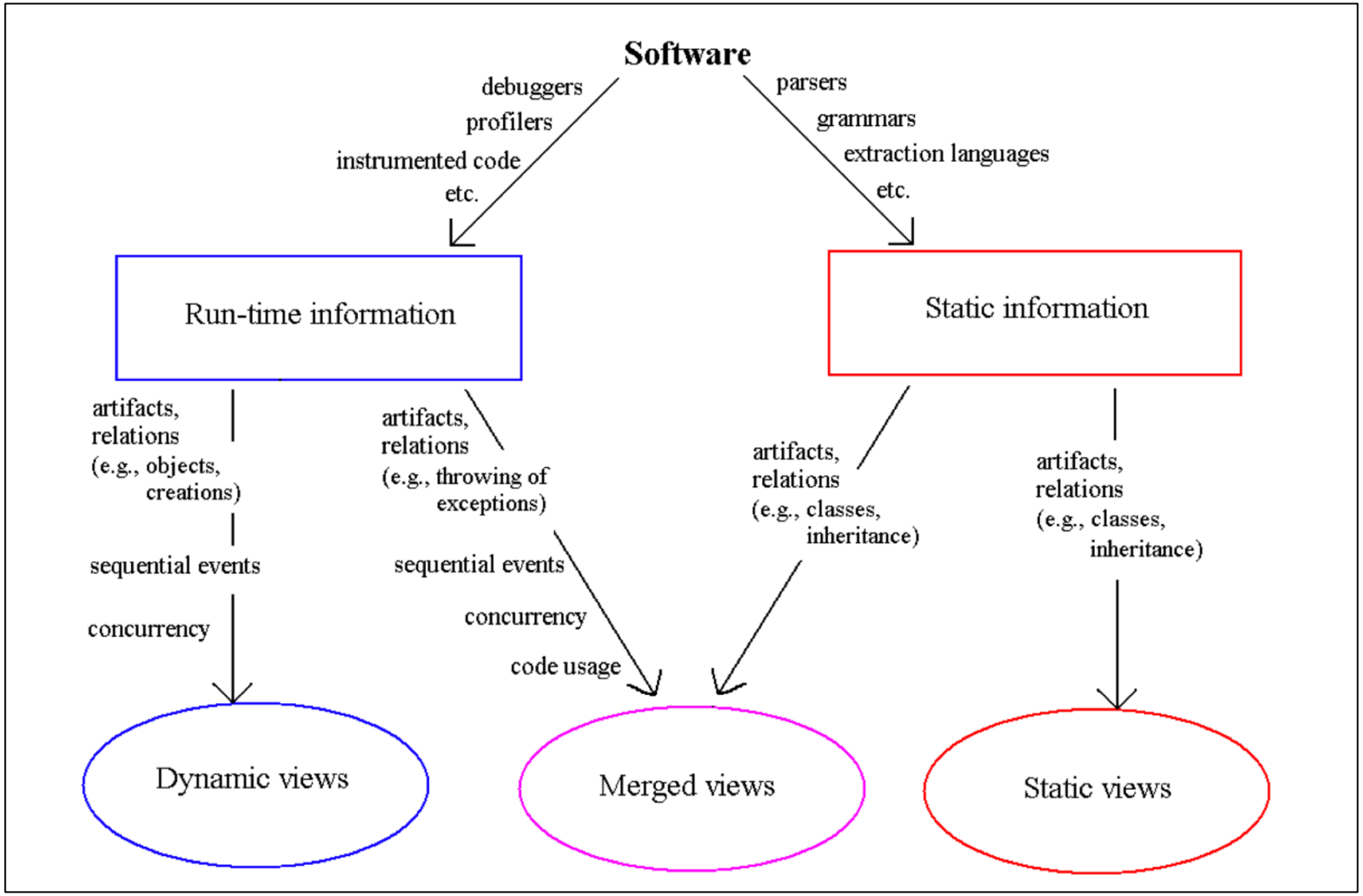
- Le code source ou binaire est souvent la seule source d'information pour comprendre les programmes.

- Collecte d'informations:
  - Parseurs, débogueurs, profileurs, enregistreurs d'événements, …

- Informations sur les modèles abstraits:
  - Faire des modèles compréhensibles et de haut niveau, …

- Informations de navigation (visualisation):
  - Des outils tels que des trancheuses (slicing) interactives, des affichages graphiques, des éditeurs, …

Compréhension du programme et outils



@M. Ward

Code source vs binaires

- Code source:
  - Meilleure forme de représentation.
  - Pas toujours disponible.
  - Le résultat dépend de l'analyseur
    - La vue extraite par l'analyseur peut être différente des compilateurs

- Binaires:
  - Collecte plus rapide d'informations (par exemple, code octet Java).
  - Problèmes de légalité.
  - Problème d’obfuscation.
  - Perte d'information: noms de variables, commentaires, structure.

Modèles statiques

- Découvrir la structure statique, l'architecture:
  - Code (en utilisant un analyseur)
  - Documents.
  - Entretiens.
  - Tranchage (slicing) statique.

- Visualisation:
  - Diagrammes de classe.
  - Graphiques d'appel.
  - Contrôler les graphiques de flux.
  - Graphiques de flux de données.
  - Graphiques de dépendance de programme.

Modèles dynamiques

- Découvrir le comportement d'exécution des logiciels:
  - Débogueur.
  - Profiler.
  - Instrumentation de code source.
  - Exécution et test: profilage, test et observation du comportement du programme.
  - Tranchage (slicing) dynamique.

- Visualisation:
  - Scénarios (diagrammes de séquence).
  - Diagrammes d'état.
  - Graphiques hiérarchiques.

Extraction des modèles : Manuelle versus automatique

- Extraction automatique:
  - En utilisant la structure du langage de programmation .
  - En utilisant des mesures (métriques).
    - Par rapport à la structure.
    - Par rapport au comportement.
    - Par rapport au vocabulaire.
    - Par rapport à l'usage.
    - Etc.

Extraction d'architecture logicielle

- Vise à présenter les systèmes logiciels existants au niveau architectural.

- Un processus de reconstruction d'architecture consiste normalement en 4 étapes:

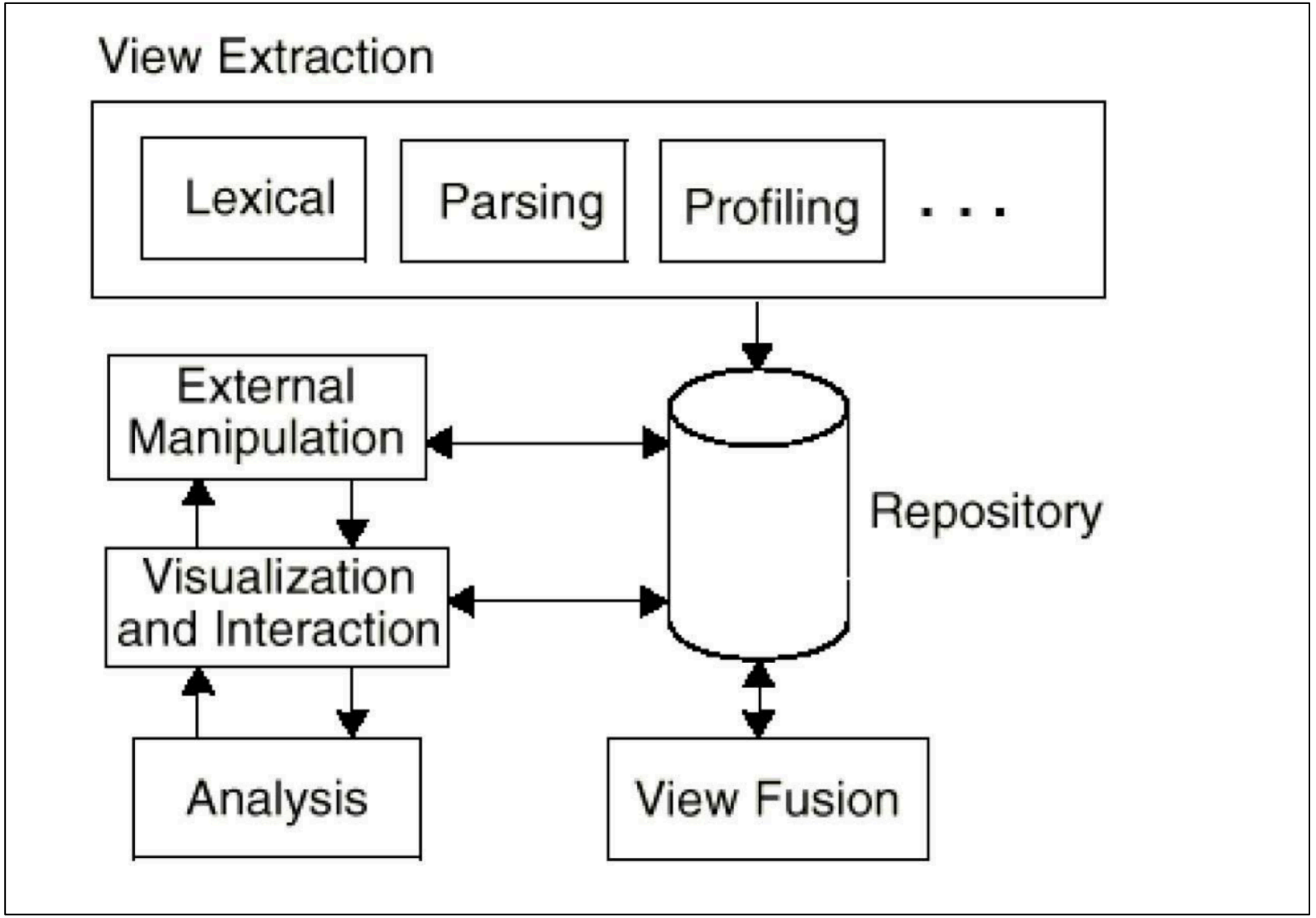
  - Définition de concepts architecturaux significatifs.

  - Collecte de données, dans laquelle un modèle de système est construit en termes de concepts définis à l'étape 1.

  - Abstraction, dans laquelle le modèle est enrichi d'abstractions (spécifiques au domaine) qui mènent à une vision plus élevée du système.

  - Présentation de l'architecture reconstruite dans une série de formats, tels que des graphiques, des diagrammes UML et des diagrammes de séquences de messages, en tenant compte de la vue architecturale requise (logique, processus, physique, développement).

Extraction d'architecture logicielle



@M. Ward

Extraction d'architecture logicielle