Program Comprehension and Software Migration Strategies

Hausi A. Müller
University of Victoria

IWPC-2000
Limerick, Ireland, June 11, 2000
Outline

- Reengineering categories
- Comprehension strategies
- Migration strategies
- Language migration
- Program comprehension education
- Mt. St. Helens Theory
- Key research pointers
- Conclusions
Research Support

NSERC CRSG

IBM Centre for Advanced Studies

IRIS

Consortium for Software Engineering Research
Consortium de recherche en génie logiciel

National Research Council Canada
Conseil national de recherches Canada

Engineering University of Victoria

ICSE 2000 Roadmap
The Horseshoe Model of Software Migration
Reengineering Categories

- Automatic restructuring
- Automatic transformation
- Semi-automatic transformation
- Design recovery and reimplementation
- Code reverse engineering and forward engineering
- Data reverse engineering and schema migration
- Migration of legacy systems to modern platforms
The Horseshoe Model

Components

Semi-automatic

Middleware

Abstract system

Existing system

New system

Automatic

Reverse engineering

Forward engineering
Reengineering Categories...

■ Automatic restructuring
  • to obtain more readable source code
  • enforce coding standards

■ Automatic transformation
  • to obtain better source code
  • HTML’izing of source code
  • simplify control flow (e.g., dead code, goto’s)
  • refactoring and remodularizeing
  • Y2K remediation
Reengineering Categories...

- Semi-automatic transformation
  - to obtain better engineered system (e.g., rearchitect code and data)
  - semi-automatic construction of structural, functional, and behavioral abstractions
  - re-architecting or re-implementing the subject system from these abstractions
Design Recovery
Levels of Abstractions

■ Application
  • Concepts, business rules, policies

■ Function
  • Logical and functional specifications, non-functional requirements

■ Structure
  • Data and control flow, dependency graphs
  • Structure and subsystem charts
  • Architectures

■ Implementation
  • AST’s, symbol tables, source text
Synthesizing Concepts

- Build multiple hierarchical mental models
- Subsystems based on SE principles
  - classes, modules, directories, cohesion, data & control flows, slices
- Design and change patterns
- Business and technology models
- Function, system, and application architectures
- Common services and infrastructure
The Ubiquitous Graph Model

- Composite node
- Generalization arcs
- Subsystem
- Composite arc
- Aggregation arcs
- Subsystem
- Classification
- Typed nodes and arcs
Program Comprehension Technology

- Program understanding technology
  - Cognitive models
  - Levels of abstraction
  - Synthesizing concepts
  - Filtering information
  - Slicing and dicing

- Comprehension environment
  - Parsers and lightweight extractors
  - Repository and conceptual modeling
  - Visualization engines (graph and web based)
The Big-Bang Comprehension Problem

- What can we do during evolution to ease future understanding and migration of information systems?
- We know the knowledge we need but it is difficult to obtain from scratch
- “Big-bang” comprehension when the system becomes “critical” is high-risk
- Analysis paralysis
The Understanding Gap

needed overall understanding

useful, known overall understanding

[Wong99]
Continuous Program Comprehension

- Apply program understanding continuously and incrementally during evolution of the software system
- Use software reverse engineering to re-document existing software
- Insert reverse engineering techniques into development [Wong99]
- Symbiosis: models and code [Jackson00]
Evaluating Reverse Engineering Tools

- The purpose of most reverse engineering tools is to increase the understanding an engineer has of the subject system
- No agreed-upon definition or test of understanding
- Several types of empirical studies that are appropriate for studying the benefits of reverse engineering tools
Program Understanding

Theses
An Emerging Discipline

- Domain retargetable reverse engineering [Tilley95]
- Cognitive design elements for software exploration tools [Storey98]
- Continuous understanding Reverse Engineering Notebook [Wong99]
- Integrating static and dynamic reverse engineering models [Systa2000]
- Architectural Component Detection for Program Understanding [Koschke2000]
Outline

- Reengineering categories
- Comprehension strategies
- Migration strategies
- Language migration
- Program comprehension education
- Mt. St. Helens Theory
- Key research pointers
- Conclusions
Migration Theses

- Management of uncertainty and inconsistency in database reengineering [Jahnke99]
- Integration and migration of information systems to object-oriented platforms [Koelsch99]
- Migrating C++ to Java [Agrawal99, Wen2000]
- An Environment for Migrating C to Java [Martin2000]
Migration Objectives
Evolving Business Requirements

- Adapt to e-commerce platform
- Adapt to web technology
- Reduce time to market
- Support new business rules
- Allow customizable billing
- Adapt to evolving tax laws
- Reengineer business processes
Migration Objectives ...
Software Evolution Requirements

- Higher productivity
- Lower maintenance costs
- Move to object-oriented platforms
- Inject component technology
- Adapt to modern data exchange technology
- Leverage modern methods and tools
Migration Objectives ...

Software Architecture Requirements

- Move to network-centric platforms
- Integrate cooperative information systems
- Leverage centralized repositories
- Move from hierarchical to relational db
- Take advantage of web user interfaces
- Provide interoperability via buses and gateways among applications
- Move to client-server architectures
Common Requirements
Migration

- Ensure continuous, safe, reliable, robust, ready access to mission-critical functions and information
  - Migrate in place
- Minimize migration risk
  - Reduce migration complexity
  - Make as few changes as possible in both code & data
  - Alter the legacy code to facilitate and ease migration
  - Concentrate on the most important current and future requirements
Common Migration Requirements ... 

- Minimize impact on
  - users
  - applications
  - databases
  - operation

- Maximize benefits of modern technology
  - user interfaces, dbs, middleware, COTS
  - automation, tools
Dimensions of Migration
Methods and Tools

Automation

User involvement

manual

automatic

10K

generic

10M

specific

Scale

Domain

ICSE 2000 Roadmap
Resistance to Change

- Are some systems more difficult to change, evolve, reengineer than others?
- Can we define a measure resistance based on business value, existing technology, new technology, evolution pace?
- We need empirical studies ...
Separable Tiers

- Decompose legacy system into three layers or application tiers
  - Presentation (interfaces: user and APIs)
  - Processing (application code, functions, business rules, policies)
  - Data services (database)
- Promotes interoperability, reuse, flexibility, distribution, separate evolution paths
Application Layers

- User Objects
- Processing Objects
- Data Objects
- Infrastructure
Classification of LIS Architectures

- Decomposable
  - Separation of concerns
  - Interfaces, applications, db services are distinct components
  - Functional decomposition
  - Ideal for migration

There is nothing more difficult to arrange, more doubtful of success, and more dangerous to carry through than initiating changes.

—N. Machiavelli
Classification of IS Architectures ...

- **Semidecomposable**
  - Applications and db services are not readily separable
  - System is not easily decomposable

- **Nondecomposable**
  - No functional components are separable
  - Users directly interact with individual modules

- [BS95]
Migration Strategies

- **Ignore**
  - retire, phase out, let fail

- **Replace with COTS applications**

- **Cold turkey**
  - rewrite from scratch
  - high risk

- **Integrate and access in place**
  - integrate future apps into legacy apps without modifying legacy apps
  - IS-GTP [Koelsch99]
Data Warehousing

- Data is needed for several distinct purposes
  - on-line transaction processing (access in place)
  - data analysis for decision support applications (extraction of data into an application specific repository)
- Creates duplicate data
- Popular approach
Gradual Migration or “Chicken Little”

- Rearchitect and transition the applications incrementally
- Replace LIS with target application
- Language migration
- Schema and data migration
- User interface migration
- GTE [BrSt95]
Chicken Little ...

- The intent is to phase out legacy applications over time
- In place access is not economical in the long run
- More effective, less risky than cold turkey
- Allows for independent user interface and database evolution
- Incremental
Chicken Little ...

- Legacy and target applications must coexist during migration
- A gateway to isolate the migration steps so that the end users do not know if the info needed is being retrieved from the legacy or target system
- Development of gateways is difficult and costly
Opportunistic Migration Method

- Combination of forward and reverse migration strategies
- Forward or reverse migration path per
  - operation
  - application
  - interface
  - database
  - site
  - user
- More complex gateways are needed
Migration Research Method

- Perform a concrete case study with an industrial software system
- Investigate methods and tools to automate the process adopted in the case study
- Conduct user experiments to improve the effectiveness of the developed methods and tools
- Investigate tool adoption problems
Language Migration—A Case Study

- Subject system is a 300 KLOC legacy software system of highly optimized code written in PL/IX

- Can the system incrementally be translated to C++?
  - Transliteration versus object-oriented design

- Develop tools which semi-automate the translation process to C++

- The translated code must perform as well as the original code
Manual Migration

- First migration and integration effort was completed by hand by an expert [Uhl97]
- 10 person-weeks to migrate 7.8 KLOC
- Successfully passed all regression tests
- Built C++ and Fortran compilers with it
- It works …
  … but migrated C++ code was 50% slower than original PL/IX code
Performance Evaluation

- Expert identified performance bottlenecks
- Hand-optimized migrated code
- Optimized version performed better than the original version [Martin98]
  - Up to 20% better than the original code
  - Now IBM was interested …

- Results
  - Correct, efficient
  - Translation, integration, optimization heuristics
  - Incremental process
Can the translation, integration, and optimization heuristics discovered by experts be integrated into an automated tool?

How would it affect the performance?

What existing tools could be leveraged to build such a tool?

Solution

- Use Software Refinery, Reasoning Systems
Transformation Process

- Transform PLI/IX artifacts to their corresponding C++ artifacts
- Generate support C++ libraries
  (macros for reference components; class definitions for key data structures)
- Generate C++ source code that is structurally and behaviorally similar to the legacy source code
- CASCON98 Best Paper
  [Kontogiannis98]
Results, Morale & Lessons Learned

- Semi-automatic transformation of large volume of code is feasible
- Migrated code suffers no deterioration in performance
- Incremental migration process feasible
- Technique readily applicable to other imperative languages
- Tool reduces migration effort by a factor of 10 over manual migration
- CTAS—C++ to Java [Jackson2000]
Outline

- Reengineering categories
- Comprehension strategies
- Migration strategies
- Language migration
- Program comprehension education
- Mt. St. Helens Theory
- Key research pointers
- Conclusions
Teaching program understanding

- How many teach 4th year or graduate courses in software evolution, program understanding, comprehension, reverse engineering, reengineering?
- How many teach program understanding or program reading in 1st year?
Challenges and Aspirations

- 1. Discriminate among different software development roles
- 4. Integrate an engineering point of view into CS and IS undergraduate curricula
- 6. Exploit our own technology in support of education
Discriminate among different software development roles

- Available knowledge about software exceeds what any one person can know
- Specializing roles
- Comprehension versus coding skills
- Developing the role of a reverse engineer, program comprehender
- Software inspection expert
Integrate an engineering point of view into undergraduate curricula

- Study good examples of software systems and develop program understanding skills
- Teach back-of-the-envelope estimation using reverse engineering technology
- Teach students how to investigate non-functional requirements using program comprehension technology
Exploit our own technology in support of education

- Employ software exploration and reverse engineering tools in 1st year
- Integrated environments such as VA Java or J++ do not provide facilities to explore and record mental models
- Familiarize students with software exploration and conceptual modeling tools
- Restructure curricula to teach both fresh creation and evolutionary change

ICSE 2000 Roadmap
Mt. St. Helens Theory

- May 18, 1980
  Mt. St. Helens self-destructed, setting off the biggest landslide in recorded history and losing 400 meters of its crown

- Forests and meadows, and mountain streams were transformed into an ash-gray wasteland

- Ecologists dogma—nature recreates ecosystems in a predictable fashion
A decade later

- A decade later even on the most sterile of landscapes brave little vegetative beachheads are formed.
- The unpredictability of recolonization and the pivotal importance of chance in rebuilding of biological communities.
- Wildflower gardens, which are mixes of lupine, Indian paintbrush, pearly everlasting, and fireweed, are emerging.
Encourage island-driven research

- Is program comprehension research becoming too predictable?
- Do we need a cataclysmic event to rejuvenate comprehension research?
- There are many vegetative beachheads in the community
- But they tend to gravitate towards established research and tools
- Particularly the tools arena needs new beachheads
Outline

- Reengineering categories
- Comprehension strategies
- Migration strategies
- Language migration
- Program comprehension education
- Mt. St. Helens Theory
- Key research pointers
- Conclusions
Key Research Pointers

- Investigate infrastructure, methods, and tools for continuous program understanding to support the entire evolution of a software system from the early design stages to the long-term legacy stages
  - Reverse engineering notebook
Key Research Pointers ...

- Instrument design architecture to ease extraction of understanding architecture
- Store architecture artifacts in schema-based repository and as unstructured or Web-based text to ease searching
- Allow for incomplete semantics and partial extraction of artifacts
Key Research Pointers ...

- Allow user to build virtual, multiple architectures, perspectives, and views
- Provide tools to compare virtual and code-centric architectures (e.g., reflection models [Murphy98])
- Make architecture extraction tools end-user programmable and extensible
Key Research Pointers ...

- Develop methods and technology for computer-aided data and database reverse engineering
  - Integrate code and data reverse engineering methods and tools
  - Leverage synergy between code and data reverse engineering communities
Key Research Pointers ...

- Develop tools that provide better support for human reasoning in an incremental and evolutionary reverse engineering process that can be customized to different application contexts
  - End-user programmable tools
  - Domain retargetable reverse engineering
Key Research Pointers ...

- Concentrate on the tool adoption problem by improving the usability and end-user programmability of reverse engineering tools to ease their integration into actual development processes
  - Start with a web-based user interface
  - Conduct user studies
Conclusions

- **Mission statement**
  - Researchers in software design and formal methods should concentrate on software evolution rather than construction
  - Program understanding and analysis experts should teach their methods in 1st-year

- **Plenty of research problems**
- **Wonderful case studies**
- **Exciting research!!!!**
Canada
May 12-19, 2001

23rd International Conference on Software Engineering

May 12-19, 2001
Toronto, Ontario, Canada
The Westin Harbour Castle

www.csr.uvic.ca/icse2001/