Rigi: A Visualization Environment for Reverse Engineering

Margaret-Anne Storey
Kenny Wong
Hausi Muller
Outline

• Rigi Project
• Rigi System
• SHriMP Views
• User Studies
• Conclusion
The Rigi Project

- Discover meaningful abstractions in large software systems and transfer this information into the minds of software engineers
- Develop an environment for research and practice in program understanding
Rigi Approach

• Program understanding through:
  • structural redocumentation
  • domain independent reverse engineering
Structural Redocumentation

• Extract software artifacts
• Build a graph of software structure
• Identify and form subsystems in the graph
• Analyze and explore the graph
• Produce “views” as documentation
Rigi Graph Model

composite node

composite arc

level arcs

subsystem

subsystem
Domain Independent Reverse Engineering

- General graph editor
- Define domain model
- Programmable (Tcl)
- Customizable user interface (Tk)
- Rigi Command Library (RCL)
- Rigi Standard Format (RSF)
Reverse Engineering Process

- [Tilley94]
- Data gathering
- Knowledge organization
- Information exploration, analysis and presentation
Exploring and Presenting Software Structures

- Software visualization tools must:
  - support various comprehension strategies
  - facilitate exploration
  - provide orientation cues
  - reduce disorientation
Rigi Exploration and Presentation

- Multiple, separate windows:
  - overview
  - parent/child
  - projection
  - selection
Rigi Graph Algorithms

- Selection
- Layout
- Filtering
Problems with Multiple Windows

- Large software systems generate large graphs
- Partitioning the graph into separate windows results in hidden information
- Users have to conceptualize the implicit relationships between windows
SHriMP Views

- Simple Hierarchical Multi-Perspective Views
- Prototype interface for exploring software structures and browsing code
- Single window user interface
- More effective use of limited screen area
SHriMP Presentation

- Nested graphs for presenting hierarchical structure
SHriMP
Exploration and Presentation

- Fisheye views — detail with context
- Pan and zoom — detail without context
- Integrated code browsing
- Composite arcs
Empirical Studies

• To study the effectiveness of program understanding tools

  1 Pilot study, Spring 1996

  2 Recent study, Spring 1997
Study 1 Design

- Compared Rigi, SHriMP and vi/grep for browsing source code
- 12 subjects, with each subject using all three tools
- Simple tasks
- Questionnaire and interview
Study 1 Results

- In Rigi, multiple windows were confusing due to hidden information
- In SHriMP, too much information was visible at times
- 8 users preferred SHriMP, 3 users preferred Rigi and one user preferred vi/grep
Study 1 Results

- For larger programs, vi/grep was not as effective as Rigi and SHriMP
- For smaller programs, vi/grep and Rigi were both less effective than SHriMP
Study 2 Design

- Compared Rigi, SHriMP and SNiFF+
- 30 subjects, with each subject using only one tool
- Broader and more realistic maintenance tasks
- Questionnaire and interview
Example Tasks

- How many players can play at once?
- Does the game support a computer mode?
- Verify that the game supports a limited number of hotels and houses.
- Implement a new rule so that a player in jail cannot collect rent.
- Overall, what was your impression of the structure of the program?
Study 2 Results

- In Rigi and SNiFF+, multiple windows were disorienting
- In Rigi and SHriMP, the lack of searching tools for source code was a problem
- In SHriMP, *sightseeing* behaviors were observed
- In SHriMP, fisheye views were rarely used
Conclusion

• Most users were able to solve the tasks in spite of the assigned tool
• Subsystem hierarchies were useful for aiding exploration of the software
• More experiments are needed to determine the most effective tool features for program understanding
Web Site

- http://www.rigi.csc.uvic.ca/