Interactive Exploration of Compacted Visualizations for Understanding Behavior in Complex Software
Making sense of complex code

The two challenges:

- Showing the ‘behavior’ of the code
- Making sure that users don’t get overwhelmed
Showing the ‘behavior’ of the code

- How are code components in a system actually used?
  - The lifecycle of code
  - Interactions among code
  - The flow of control
  - ...
  - ...
Showing the ‘behavior’ of the code

- How are code components in a system actually used?
  - The lifecycle of code
  - Interactions among code
  - The flow of control
  - ...
  - ...

- Sequence diagram based visualizations –
  - Demonstrate overviews of behavior
  - Provide easy analysis of a developer’s logic
Making sure that users don’t get overwhelmed

- Growing abundance of software systems with complex codebases
- Diagrams produced by traditional sequence diagramming tools quickly become large and unmanageable when dealing with complex code
Making sure that users don’t get overwhelmed

- Growing abundance of software systems with complex codebases
- Diagrams produced by traditional sequence diagramming tools quickly become large and unmanageable when dealing with complex code

How do we make sure the diagrams stay useful when the code gets complicated?

- Presenting users with minimal, manageable amount of information
- Helping user to interact with and manipulate a diagram
- Integration and automation
Demo 1: Exploration

- Support the incremental exploration of a large or complicated project via a user’s expansion of components
Demo 1: Exploration

- Support the incremental exploration of a large or complicated project via a user’s expansion of components
- Icons and locations of exploration handles indicate the type of interactions that they can be used to display
- Easily exploring called methods shows how and why code is used
- Methods that override a superclass method show alternate implementations
Demo 2: Preventing and eliminating information overload

- Filter out less relevant information to emphasize key and pertinent parts of a system
- Condense and compact automatically
- Also allow for easy compression, hiding, or removal of components directly by user
- Use unobtrusive visual components whenever possible to convey information
Demo 2: Preventing and eliminating information overload

- Filter out less relevant information to emphasize key and pertinent parts of a system
- Condense and compact automatically
- Also allow for easy compression, hiding, or removal of components directly by user
- Use unobtrusive visual components whenever possible to convey information

- Hiding and prevention of addition of library code
- “Backward” messages
- Conditional blocks
- “Chained calls”
- Deletion
Demo 3: Automation & integration

- Makes use familiar, intuitive, and minimizes learning curve
- Eases navigation process while allowing users to maintain control
Demo 3: Automation & integration

- Makes use familiar, intuitive, and minimizes learning curve
- Eases navigation process while allowing users to maintain control

- Display all interactions among components with a single button push
- Generate sequence diagram when the debugger is launched. Stepping with debugger automatically updates the diagram, adding the new methods executed
- Generate a diagram of the most recent navigations: refocuses the user
- Convert diagram type: allows analysis of different properties and aspects of code
Questions, comments?

We want your feedback!

http://www.architexa.com
info@architexa.com

“Interactive Exploration of Compacted Visualizations for Understanding Behavior in Complex Software”