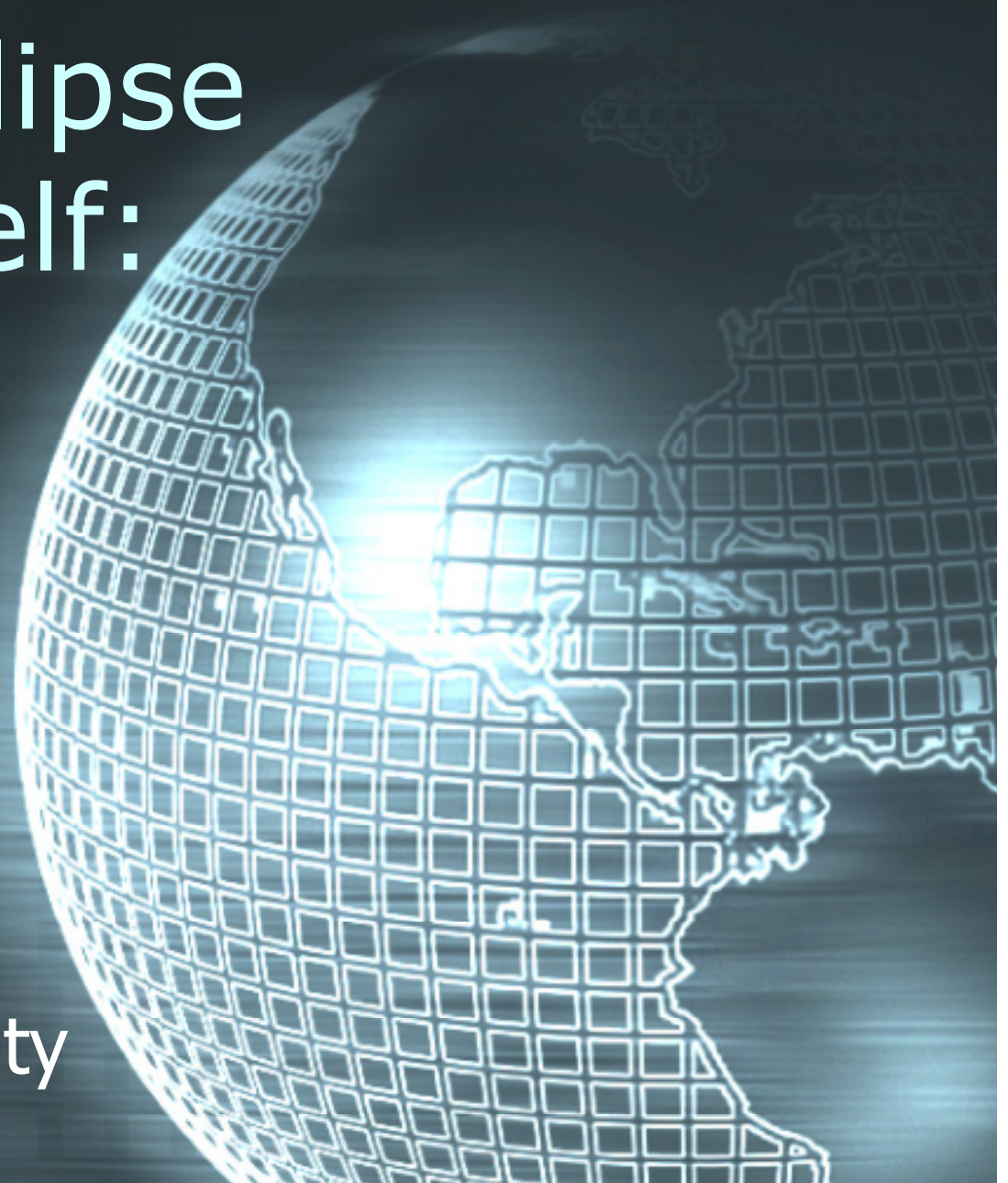


Turning Eclipse Against Itself:

Finding Errors in
Eclipse Sources

Benjamin Livshits

Stanford University



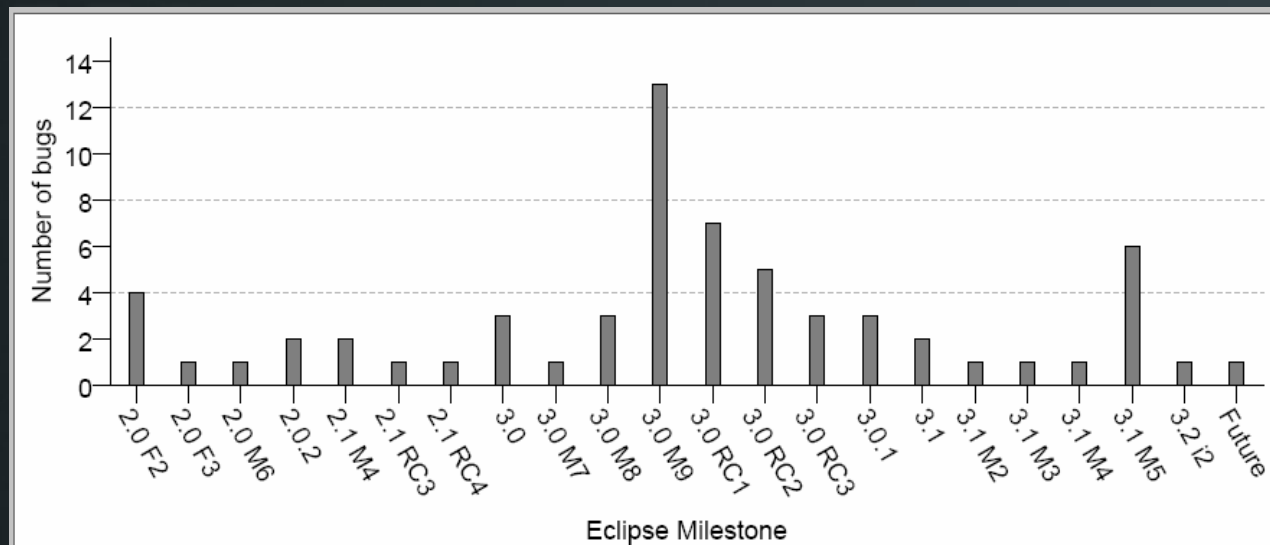
Summary



- We want to **find errors** in **Eclipse**
 - Implemented a tool called **Checkclipse** (plugin)
 - Uses **lightweight static analysis**
- Looks for violations of **3 design rules**
 - One **API usage** rule
 - Two **resource management** rules
- Preliminary results are **encouraging**
 - Ran Checkclipse on Eclipse sources
 - Found a total of **68** likely errors
 - Checkclipse checks multiple plugins in **minutes**

Eclipse Code Base...

- Eclipse:
 - One of the biggest Java projects ever written
 - Very robust
- Still, a multitude of bugs exist
 - bugs.eclipse.org – hundreds of known errors
 - Think about *unknown* ones!
 - Certain types of errors are repeatedly introduced over and over
 - “Lapsed listener” errors discovered for different Eclipse releases



Error Patterns to the Rescue



- Lots of API-specific coding patterns
 - Patterns are “specified”
 - using comments
 - not at all
 - Misuse of these patterns leads to errors
- This is great news for us!
 - Discover what the error patterns are
 - Find and report *pattern violations*
 - Can do so using *dynamic* or *static* analysis
- On to the error patterns...
 - 3 patterns evaluated
 - Many more remain – looking to expand the scope of Checkclipse

Error pattern #1: Call super

- A common rule of thumb in Eclipse code
 - For many methods m
 - A subclass implementing method m must call $super.m(...)$ inside method m

Must call `super`,
but don't

Checkclipse
super checker

Suspicious call	Category	Project	File
<input type="checkbox"/> PlatformActivator.stop	org.eclipse.core.runtime.Plugin.stop(BundleContext)	org.eclipse...	PlatformAct...
<input type="checkbox"/> ProgressViewer.hookControl	org.eclipse.jface.viewers.ContentViewer.hookControl(Control)	org.eclipse...	org.eclipse...
<input type="checkbox"/> AbstractElementListSelectionDialog.create	org.eclipse.jface.window.Window.create()	org.eclipse...	org.eclipse...
<input type="checkbox"/> CheckedTreeSelectionDialog.create	org.eclipse.jface.window.Window.create()	org.eclipse...	org.eclipse...
<input type="checkbox"/> ElementTreeSelectionDialog.create	org.eclipse.jface.window.Window.create()	org.eclipse...	org.eclipse...
<input type="checkbox"/> DetachedWindow.configureShell	org.eclipse.jface.window.Window.configureShell(Shell)	org.eclipse...	org.eclipse...
<input type="checkbox"/> Control.releaseChild	org.eclipse.swt.widgets.Widget.releaseChild()	org.eclipse...	org.eclipse...
<input type="checkbox"/> Shell.releaseChild	org.eclipse.swt.widgets.Widget.releaseChild()	org.eclipse...	org.eclipse...
<input type="checkbox"/> AnnotationRulerColumn.createCanvas	org.eclipse.swt.widgets.Control.addMouseListener(MouseListener)	org.eclipse...	org.eclipse...
<input type="checkbox"/> AbstractSourceContainer.dispose	org.eclipse.debug.core.sourcelookup.ISourceContainer.dispose()	org.eclipse...	AbstractSo...
<input type="checkbox"/> EncodingActionGroup.dispose	org.eclipse.ui.actions.ActionGroup.dispose()	org.eclipse...	org.eclipse...

Error pattern #2: Failing to Dispose

- OS resource leaks are common:
 - Many classes define method *dispose()*
 - SWT design rule: dispose what you create
 - Interesting special case: maps
 - Need to clear most class-allocated maps in *dispose()*
 - Failing to clear the maps, causes OS resource leaks

Checkclipse
dispose checker

Cleared

Locally allocated
maps

Not cleared

```
Extend super violation... | Dispose checker viewer | Lapsed listeners | Error Log | Problems | Javadoc | Declaration | Console | Progress | Search | Call Hierarchy
```

- LaunchViewContextListener.dispose
 - + modelsToContexts
 - + modelsToActivities
 - + contextViews
 - fContextSubmissions
 - List submissions=(List)fContextSubmissions.get(launch);
 - fContextSubmissions.put(launch, submissions);
 - List submissions=(List)fContextSubmissions.remove(launches[0]);
- + TableRenderingContentProvider.dispose
- + VariablesView.dispose
- + ContextualLaunchAction.dispose
- + CodeAssistConfigurationBlock.dispose
- FoldingConfigurationBlock.dispose
 - + fProviderDescriptors
 - + fProviderPreferences
 - + fProviderControls

Error pattern #3: Lapsed Listeners

- Memory leaks exist in Java, despite GC!
- Common case of memory leaks:
 - Listeners are used to register handlers for events, such as mouse clicks, etc.
 - Not un-registering listeners properly leads to memory leaks
 - Memory leaks lead to crashes and instability

Not unregistered listener

Properly registered and unregistered listener

```
Extend super violation... Dispose checker viewer Lapsed listeners Error Log Problems Javadoc Declaration Console Progress Search Call Hierarchy  
- TableWithTotalView  
  - createPartControl  
    - getTableListener()  
    - viewer.addSelectionChangedListener(getTableListener())  
- VariablesView  
- AbstractDebugView  
- AbstractInfoView  
  - createPartControl  
    - getSite().getWorkbenchWindow().getPartService().addPartListener(fPartListener)  
  - dispose  
    - getSite().getWorkbenchWindow().getPartService().removePartListener(fPartListener)  
    - provider.removeSelectionChangedListener(fCopyToClipboardAction)  
- BreakpointsView  
  - createPartControl  
    - DebugPlugin.getDefault().getBreakpointManager().addBreakpointManagerListener(this)
```

Checking for Pattern Violations: How?



- Runtime or dynamic approaches
 - Aspects allow run time checking of rules
 - Memory profilers and debuggers
 - Custom-made tools:
 - *sleak* by Steve Northover, the architect of SWT
 - a tool to check for memory leaks in Eclipse code
 - But: violations need to be triggered during a particular execution!
- Instead, we analyze the Java source code of the plugins
- Advantages:
 - No need to consider a particular execution
 - So, can find all potential pattern violations

Static Analysis State of the Art



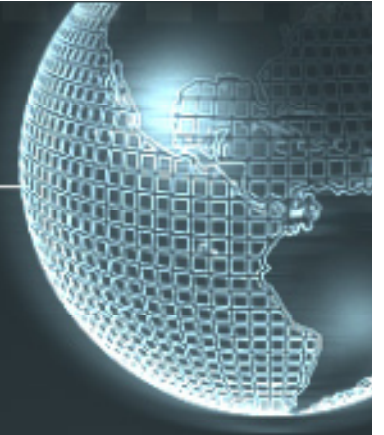
- Sound and complete analysis approaches
 - Suffer from imprecision – false positives
 - Don't scale to code bases the size of Eclipse
- We use unsound lightweight static analysis
 - Runs fast – took several minutes to analyze 20 core Eclipse plugins
 - Produces false positives
 - Takes time to filter the false positives
 - May miss errors

Why Lightweight Static Checking?



- Overall goal:
 - Address whole *classes* of problems
 - Better target debugging efforts of Eclipse developers
- Make it fast and easy to audit potential errors
 - User is presented with three custom viewers
 - One for each error pattern
 - Extend super viewer
 - Dispose rule viewer
 - Lapsed listener viewer
- Can run analysis and fix the errors without ever leaving Eclipse
- So, what did we find?

We Find...



EXTEND SUPER	
methods that require super to be called	38
calls to these methods	390
filtered calls	19
potential errors (methods not calling super)	13
DISPOSAL RULES	
dispose methods checked	794
filtered methods	51
potential errors (leaking dispose methods)	42
LAPSED LISTENERS	
subclasses of ViewPart checked	81
subclasses with matched listeners	6
subclasses not using listeners	53
subclasses with mismatched listeners	22
potential errors (classes with lapsed listeners)	13
TOTAL ERRORS	
	68

Status

Implemented Checkclipse

- Working tool
- Runs fast
- Available for download
suif.stanford.edu



Find 68 *likely* errors

- Many are hard to evaluate
- Used our best judgement to determine what is an error
- Need a strong knowledge of APIs



Future Work

Happy to pass the errors over to IBM engineers

Have a project to find and correct similar error patterns **dynamically**

Looking for new patterns to check

