



THE UNIVERSITY  
OF QUEENSLAND

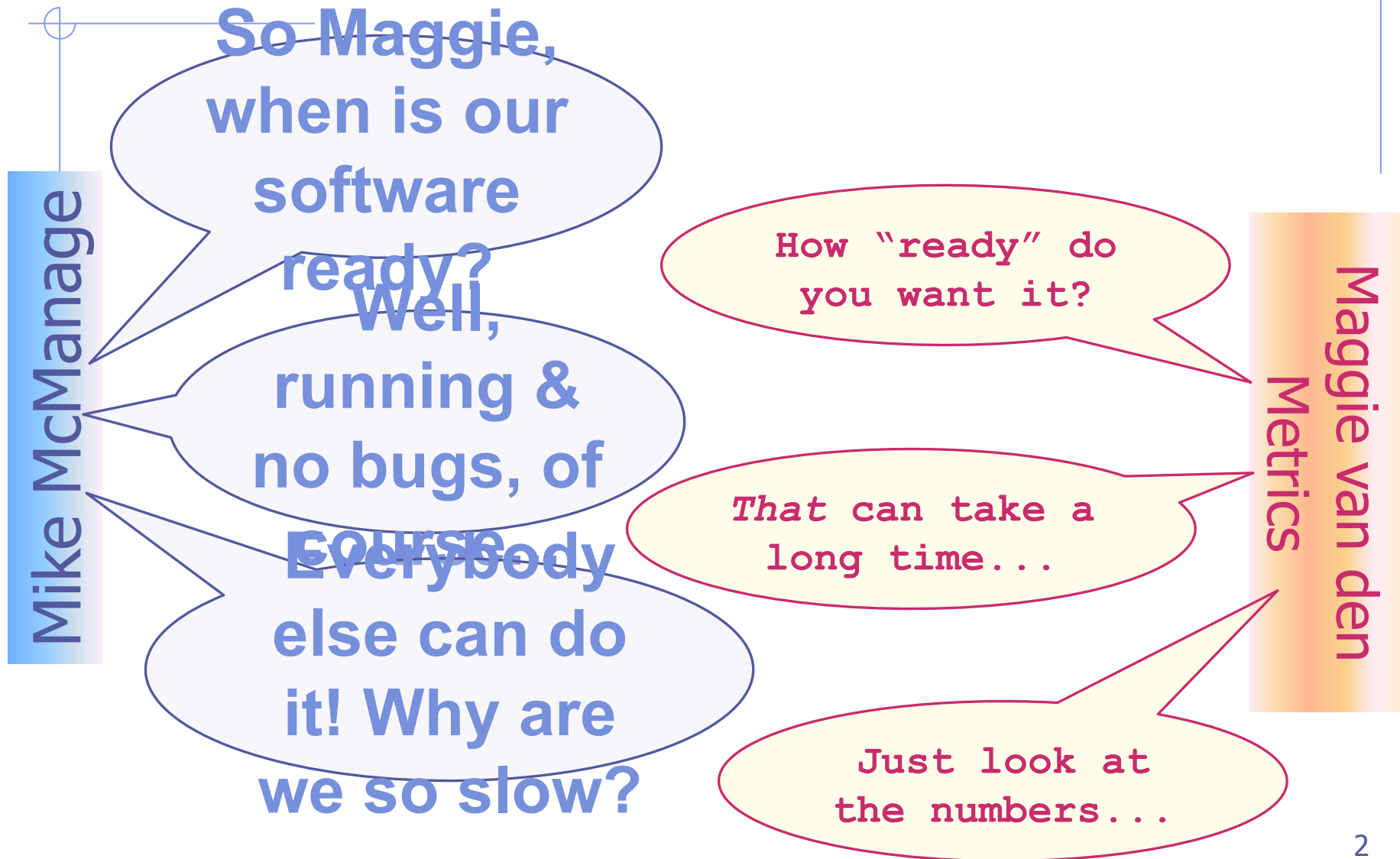
# **CSSE2003**

## **Software Engineering Studio**

Semester 2, 2009

### **10: Analysis Tools: Strength in Numbers**

# A High-Quality Conversation



# “Good Enough” Software

- ◆ Hardly any software is beyond improvement
  - ◆ If it never ships, it will never be useful
  - ◆ Pareto Principle:  
80% of the effects come from 20% of the causes
- The art is to understand what is **good enough** to release.
- ◆ Many measures: Bug counts, repository volatility...
  - ◆ In this lecture: Tools for assessing **Quality of Code**

# Three Flavours of Bugs

- ◆ **Macroscopic: Package Level → JDepend**
  - Tangled Package Dependencies
  - Complex Code (high nesting)
- ◆ **Idiomatic Style: Class Level → PMD**
  - Inappropriate Naming
  - Bad use of Libraries
  - Violated Style conventions
- ◆ **Bad Practice: Class Level → FindBugs**
  - Unsafe Multithreading Constructs

# The Example: A Timetable Formatter

- ◆ Formats Microsoft Outlook 'Virtual Free/Busy' (.vfb) files as HTML pages and uploads them
- ◆ Numerous Libraries and Tools used
  - Parser Generator
  - HTML Validator and Formatter
  - FTP/SSH Uploader
  - Registry-Access
  - Template Engine
  - ...

# JDepend – User Interface



**JDepend**

**File**

**Depends Upon - Efferent Dependencies (6 Packages)**

- epayment.adapters (CC: 2 AC: 0 Ca: 0 Ce: 4 A: 0 I: 1 D: 0)
  - com.abc.epayment
  - com.xyz.epayment
  - epayment.framework
  - epayment.response
    - epayment.framework
- epayment.commands (CC: 5 AC: 0 Ca: 0 Ce: 1 A: 0 I: 1 D: 0)
- epayment.framework (CC: 1 AC: 5 Ca: 5 Ce: 0 A: 0.83 I: 0 D: 0.17)
- epayment.processor (CC: 2 AC: 0 Ca: 0 Ce: 1 A: 0 I: 1 D: 0)
- epayment.request (CC: 1 AC: 0 Ca: 0 Ce: 1 A: 0 I: 1 D: 0)
- epayment.response (CC: 1 AC: 0 Ca: 1 Ce: 1 A: 0 I: 0.5 D: 0.5)

**Used By - Afferent Dependencies (8 Packages)**

- com.abc.epayment (CC: 0 AC: 0 Ca: 1 Ce: 0 A: 0 I: 0 D: 1)
  - epayment.adapters
- com.xyz.epayment (CC: 0 AC: 0 Ca: 1 Ce: 0 A: 0 I: 0 D: 1)
- epayment.adapters (CC: 2 AC: 0 Ca: 0 Ce: 4 A: 0 I: 1 D: 0)
- epayment.commands (CC: 5 AC: 0 Ca: 0 Ce: 1 A: 0 I: 1 D: 0)
- epayment.framework (CC: 1 AC: 5 Ca: 5 Ce: 0 A: 0.83 I: 0 D: 0.17)
  - epayment.adapters
  - epayment.commands
  - epayment.processor
  - epayment.request
- epayment.response (CC: 1 AC: 0 Ca: 1 Ce: 1 A: 0 I: 0.5 D: 0.5)
  - epayment.processor (CC: 2 AC: 0 Ca: 0 Ce: 1 A: 0 I: 1 D: 0)
  - epayment.request (CC: 1 AC: 0 Ca: 0 Ce: 1 A: 0 I: 1 D: 0)
  - epayment.response (CC: 1 AC: 0 Ca: 1 Ce: 1 A: 0 I: 0.5 D: 0.5)

**epayment.adapters (CC: 2 AC: 0 Ca: 0 Ce: 4 A: 0 I: 1 D: 0)**

# JDepend - Metrics

- ◆ **CC** - Concrete Class Count
- ◆ **AC** - Abstract Class (and Interface) Count
- ◆ **Ca** - Afferent Couplings ( $C_a$ )
- ◆ **Ce** - Efferent Couplings ( $C_e$ )
- ◆ **A** - Abstractness (0-1)
- ◆ **I** - Instability (0-1)
- ◆ **D** - Distance from the Main Sequence (0-1)
- ◆ **V** - Volatility (0-1)
- ◆ **Cyclic** - If the package contains a dependency cycle



# JDepend - Integrations

- ◆ Integrations
  - Eclipse
  - Ant
  - Junit
  - ...
  
- ◆ Live Demo ...



# Pretty Much Done ... Targets



- ◆ Possible bugs
  - empty try/catch/finally/switch statements
- ◆ Dead code
  - unused local variables, parameters and private methods
- ◆ Suboptimal code
  - wasteful String/StringBuffer usage
- ◆ Overcomplicated expressions
  - unnecessary if statements,
  - for loops that could be while loops
- ◆ Duplicate code
  - copied/pasted code means copied/pasted bugs

# Pretty Much Done ... Approach



- ◆ Java Code is Analysed, then Domain Rules are run
- ◆ Examples of Domain-specific Rule Sets:  
<http://pmd.sourceforge.net/rules/index.html>
- ◆ Rules can be written in
  - XPath (an XML Query Language) or
  - Java

# Pretty Much Done ... Practice



- ◆ Integration:
  - Eclipse Plugin
  - Ant Task
  - ... Many more
  
- ◆ Examples:
  - <http://pmd.sourceforge.net/scoreboard.html>
  - [http://pmd.sourceforge.net/reports/easystruts\\_easystruts-plugin.html](http://pmd.sourceforge.net/reports/easystruts_easystruts-plugin.html)
  
- ◆ Demo

# FindBugs™ - Find Bugs in Java Programs



- ◆ Designed and maintained by U. of Maryland
- ◆ Analyses data flow and decision trees
- ◆ Hunts for Multithreading and Data Access Bugs
  
- ◆ Professional Tool. Users:
  - Sun (for JDK7)
  - Google (where Java is used)
  - Eclipse org (e.g. Eclipse 3.5)

# FindBugs™ - 369 Types / 9 Categories



- ◆ **Bad practice**
- ◆ **Correctness**
- ◆ Experimental
- ◆ Internationalization
- ◆ Malicious code vulnerability
- ◆ **Multithreaded correctness**
- ◆ **Performance**
- ◆ Security
- ◆ **Dodgy**

**Bold = Large Number of Types**

# FindBugs™ - Some Real-life Statistics



Application	Correctness bugs		Bad Practice	Dodgy	KNCSS
	Null Pointer	Other			
<b>Sun JDK 1.7.0-b12</b>	68	180	954	654	597
<b>eclipse- SDK-3.3M7- solaris-gtk</b>	146	259	1,079	643	1,447
<b>netbeans-6_0-m8</b>	189	305	3,010	1,112	1,022
<b>glassfish-v2-b43</b>	146	154	964	1,222	2,176
<b>jboss-4.0.5</b>	30	57	263	214	178

KNCSS = Thousands of lines of non-commenting source statements  
NP-Bugs = Null-Pointer Bugs

Source: Findbugs Website (<http://findbugs.sourceforge.net>)

## For Extra Credit ...

- ◆ In marking the final report I will allocate 50% of my excellence marks to workgroups...
  - That have calculated metrics of their projects using the three tools shown in this lecture
  - Have automated the step of gathering the metrics for at least two tools
  - Show significant numeric improvement against the original measurements **after** implementation