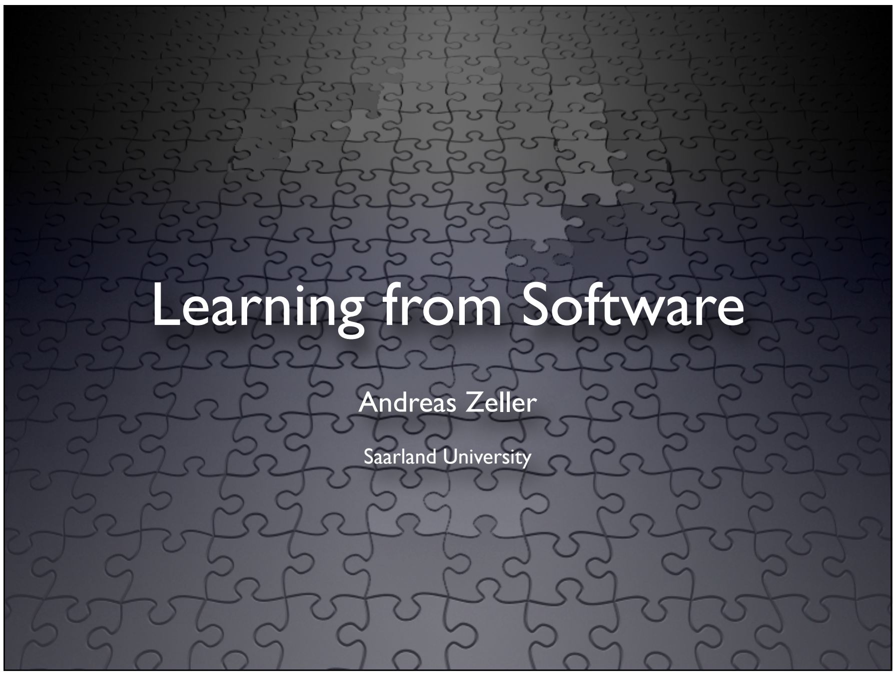


Learning from Software

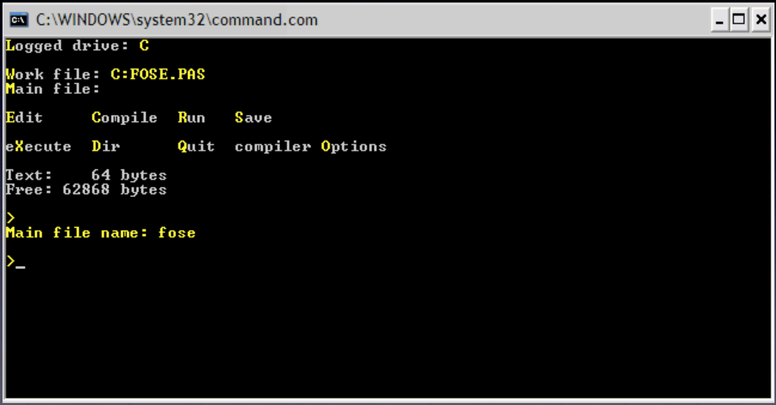
Andreas Zeller

Saarland University

The Future of Programming Environments: Integration, Synergy, and Assistance
Andreas Zeller, Saarland University

Modern programming environments foster the integration of automated, extensible, and reusable tools. New tools can thus leverage the available functionality and collect data from program and process. The synergy of both will allow to automate current empirical approaches. This leads to automated assistance in all development decisions for programmers and managers alike: “For this task, you should collaborate with Joe, because it will likely require risky work on the

Programming Environments



```
C:\WINDOWS\system32\command.com
Logged drive: C

Work file: C:FOSE.PAS
Main file:

Edit      Compile  Run    Save

eXecute   Dir      Quit   compiler Options

Text:     64 bytes
Free: 62868 bytes

>
Main file name: fose

>_
```

Turbo Pascal – just 30K (Eclipse: 118 MB – 4,000x as big)

A Tool Set

Integration – Foto von Werkstatt, Werkzeugkiste

Tools evolve

Tools evolve

Tools integrate

But do these tools work together? Where is the whole more than the sum of its parts?

Tools work together

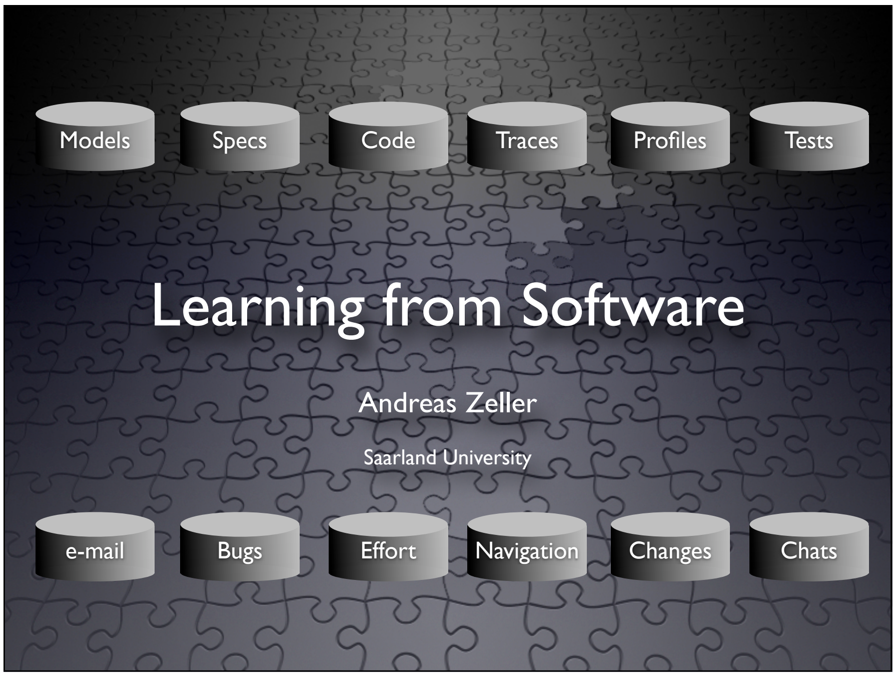
Tools can only work together if they draw on different artefacts

What are we working on in SE – we are constantly producing and analyzing artefacts: code, specs, etc.

Tools work together

Tools can only work together if they draw on different artefacts

What are we working on in SE – we are constantly producing and analyzing artefacts: code, specs, etc.



Learning from Software

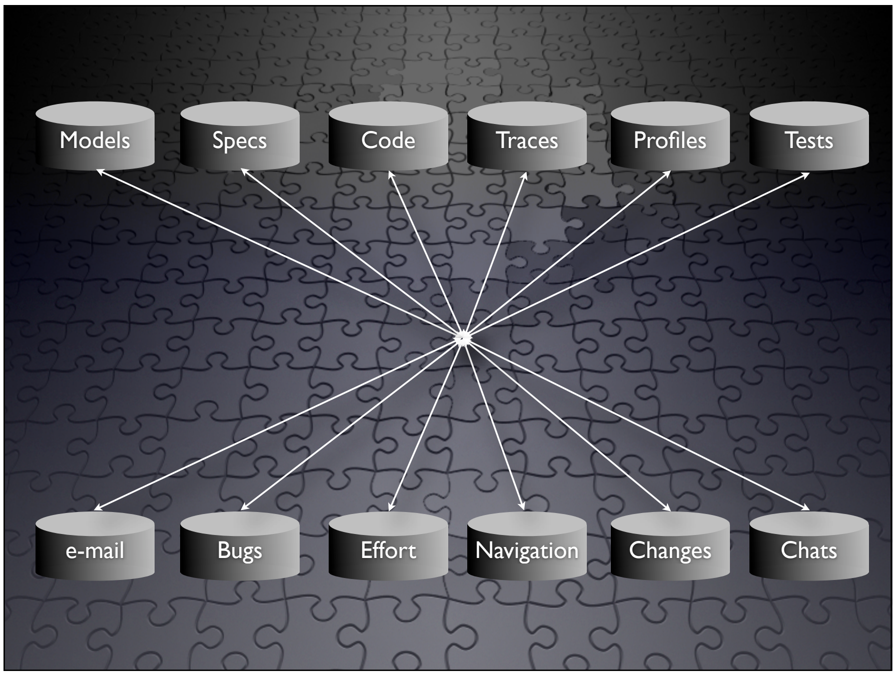
Andreas Zeller

Saarland University

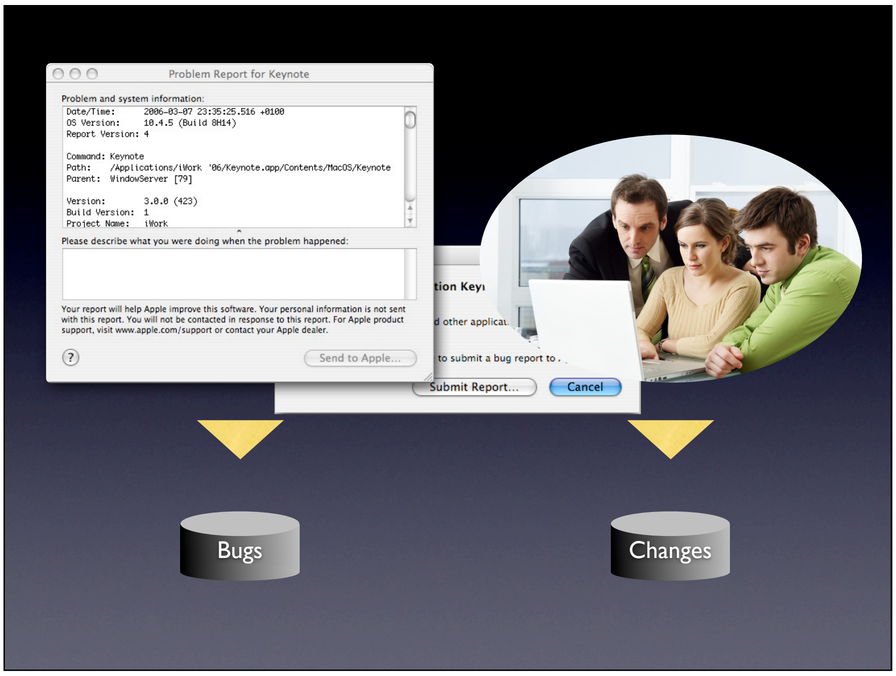
Models
Specs
Code
Traces
Profiles
Tests
e-mail
Bugs
Effort
Navigation
Changes
Chats

Tools can only work together if they draw on different artefacts

What are we working on in SE – we are constantly producing and analyzing artefacts: code, specs, etc.



Combining these sources will allow us to get this “waterfall effect” – that is, being submerged by data; having more data than we could possibly digest.



Problem Report for Keynote

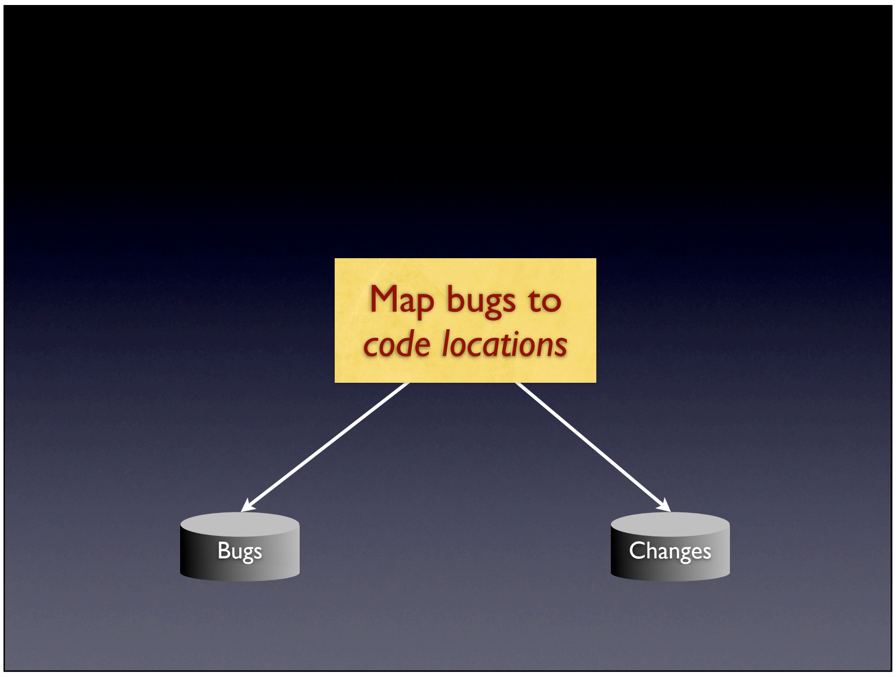
Problem and system information:

```
Date/Time:      2006-03-07 23:35:25.516 +0100
OS Version:     10.4.5 (Build 8H14)
Report Version: 4

Command: Keynote
Path:    /Applications/iWork '06/Keynote.app/Contents/MacOS/Keynote
Parent:  WindowServer [79]

Version:        3.0.0 (423)
Build Version:  1
Project Name:   iWork
```

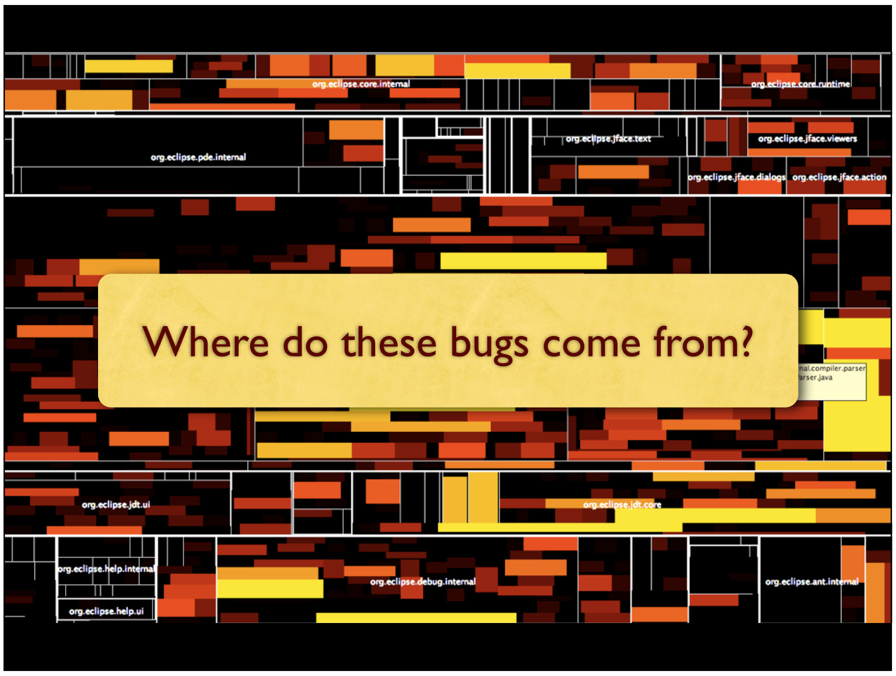
Please describe what you were doing when the problem happened:

Your report will help Apple improve this software. Your personal information is not sent with this report. You will not be contacted in response to this report. For Apple product support, visit www.apple.com/support or contact your Apple dealer.

Such software archives are being used in practice all the time. If you file a bug, for instance, the report is stored in a bug database, and the resulting fix is stored in the version archive.



These databases can then be mined to extract interesting information. From bugs and changes, for instance, we can tell how many bugs were fixed in a particular location.



This is what you get when doing such a mapping for eclipse. Each class is a rectangle in here (the larger the rectangle, the larger its code); the colors tell the defect density – the brighter a rectangle, the more defects were fixed in here. Interesting question: Why are come modules so much more defect-prone than others? This is what has kept us busy for years now.

Is it the Developers?

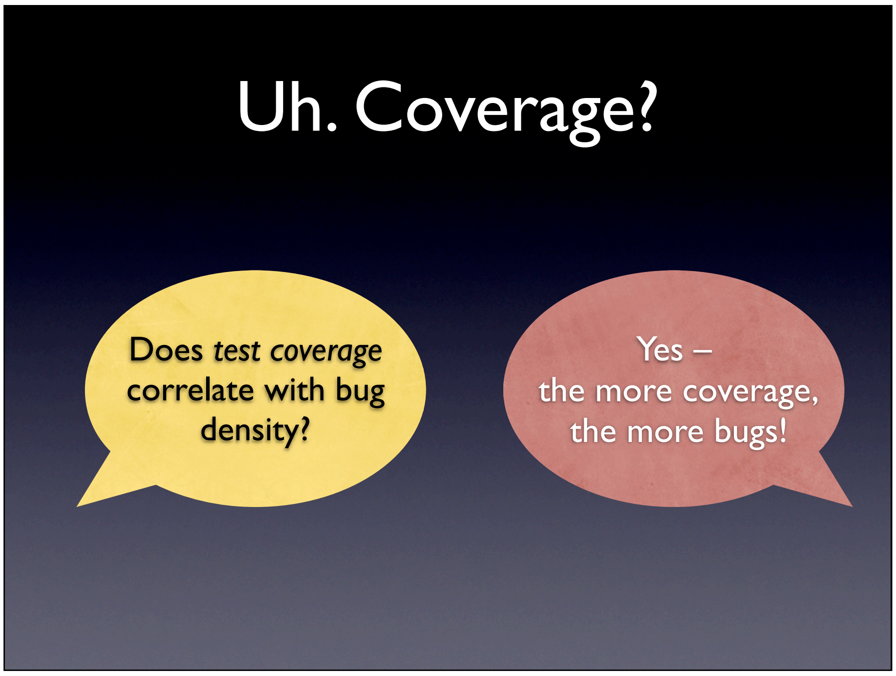
Does experience matter?
Bug density correlates with experience!

Is it History?

I found lots of bugs here. Will there be more?
Yes! (But where did these come from?)

How about metrics?

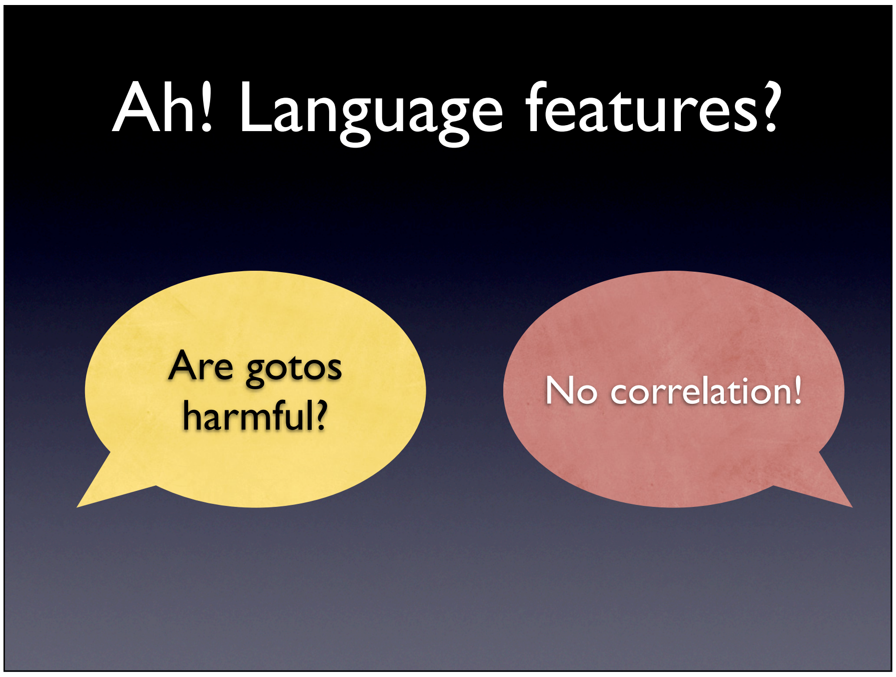
Do *code metrics* correlate with bug density?
Sometimes!



Uh. Coverage?

Does *test coverage* correlate with bug density?

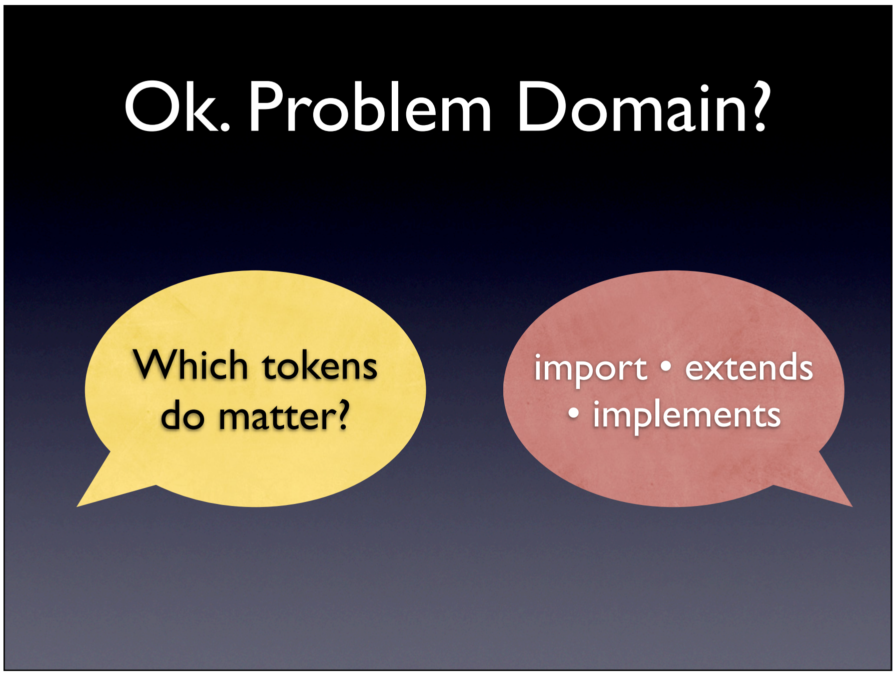
Yes – the more coverage, the more bugs!



Ah! Language features?

Are gotos harmful?

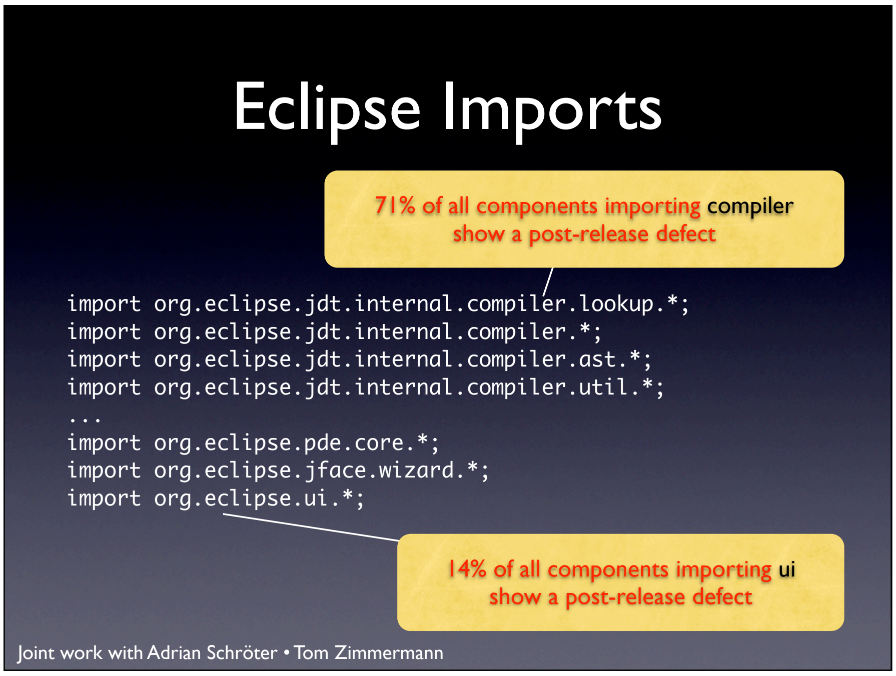
No correlation!



Ok. Problem Domain?

Which tokens do matter?

import • extends • implements



Eclipse Imports

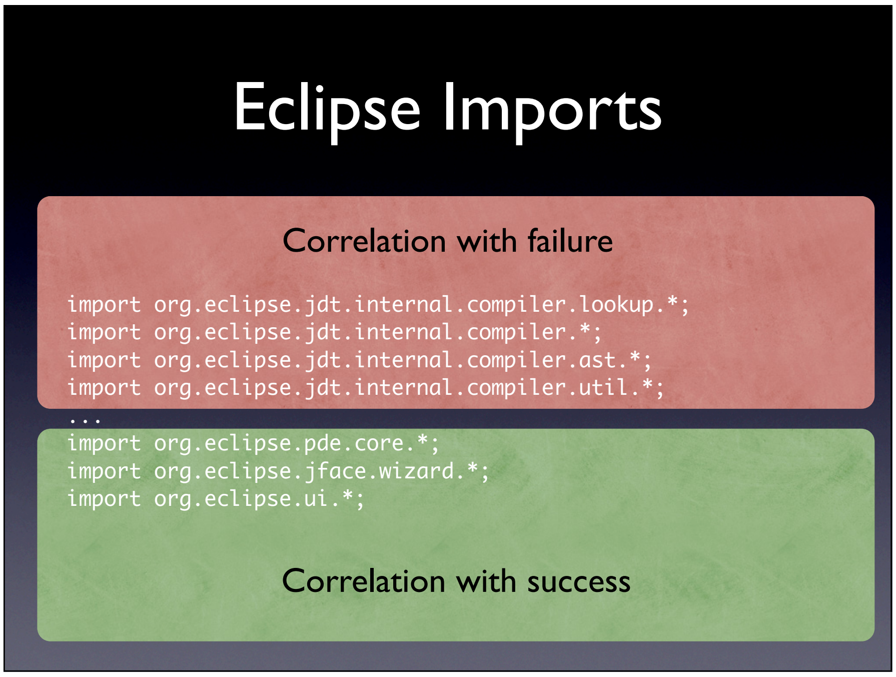
71% of all components importing compiler show a post-release defect

```
import org.eclipse.jdt.internal.compiler.lookup.*;
import org.eclipse.jdt.internal.compiler.*;
import org.eclipse.jdt.internal.compiler.ast.*;
import org.eclipse.jdt.internal.compiler.util.*;
...
import org.eclipse.pde.core.*;
import org.eclipse.jface.wizard.*;
import org.eclipse.ui.*;
```

14% of all components importing ui show a post-release defect

Joint work with Adrian Schröter • Tom Zimmermann

The best hint so far what it is that determines the defect-proneness is the import structure of a module. In other words: “What you eat determines what you are” (i.e. more or less defect-prone).



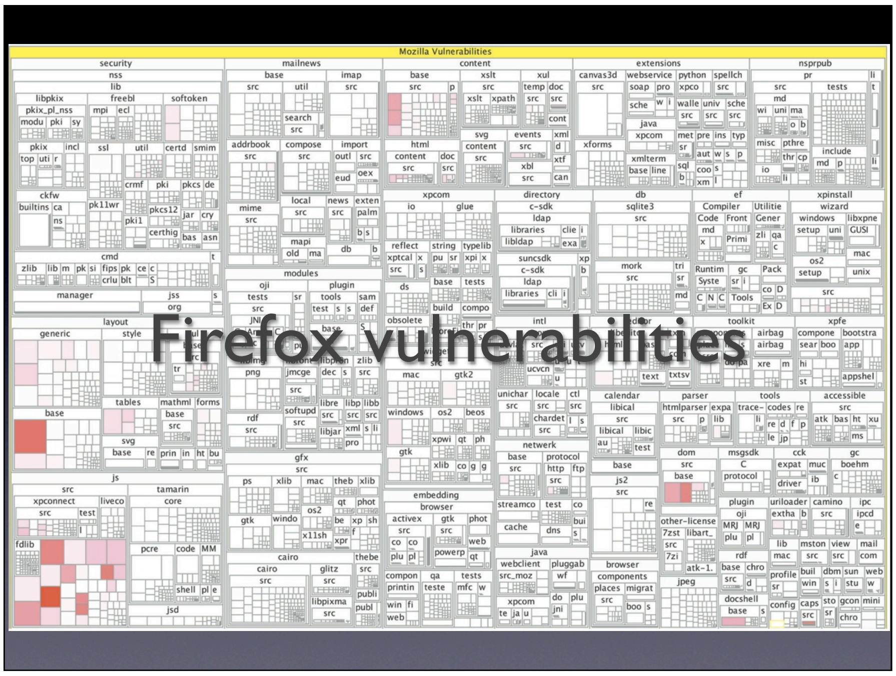
Eclipse Imports

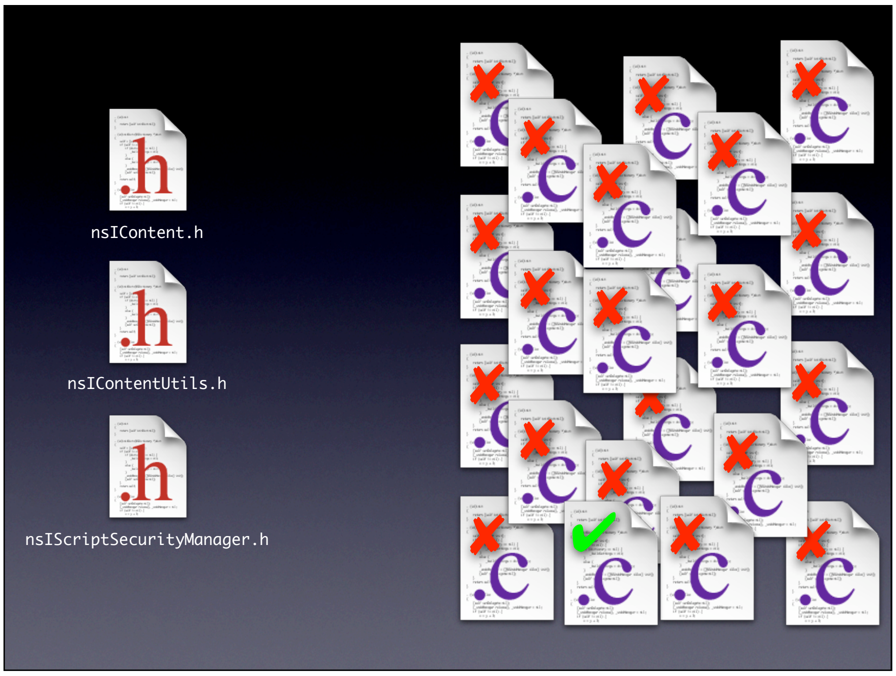
Correlation with failure

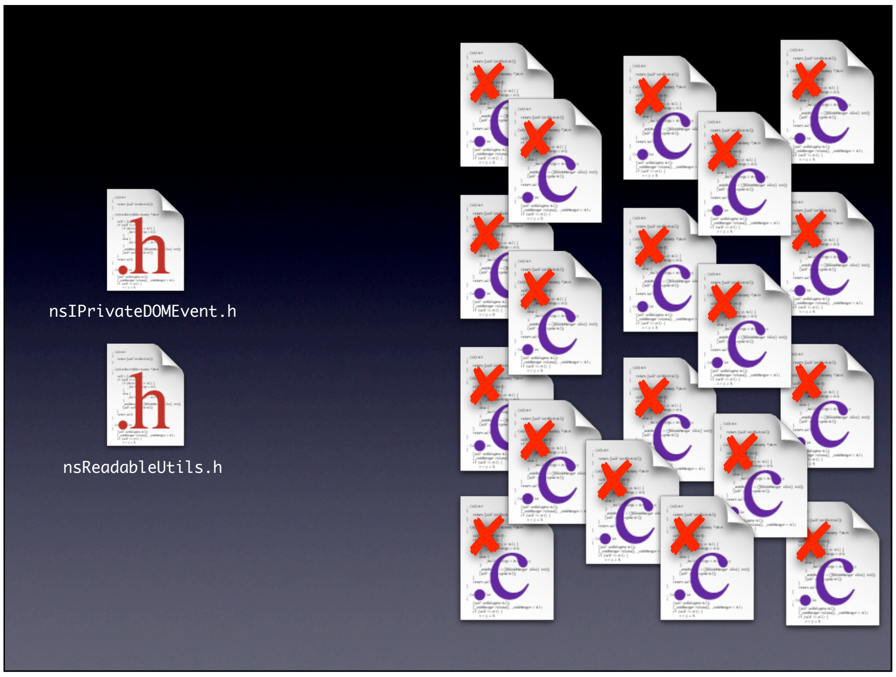
```
import org.eclipse.jdt.internal.compiler.lookup.*;
import org.eclipse.jdt.internal.compiler.*;
import org.eclipse.jdt.internal.compiler.ast.*;
import org.eclipse.jdt.internal.compiler.util.*;
...
import org.eclipse.pde.core.*;
import org.eclipse.jface.wizard.*;
import org.eclipse.ui.*;
```

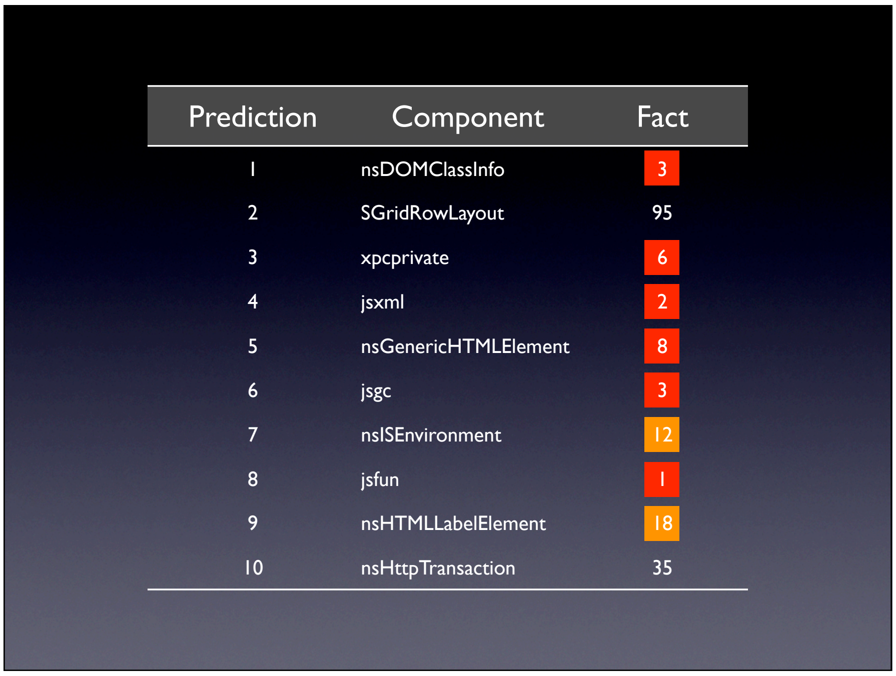
Correlation with success

For instance, if your code is related to compilers, it is much more defect-prone, than, say, code related to user interfaces.

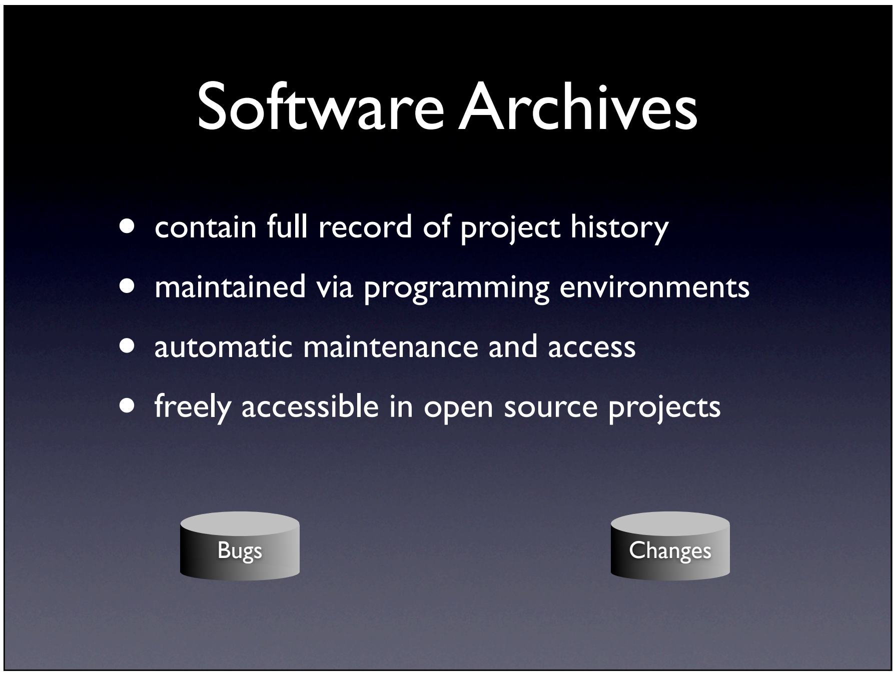








Prediction	Component	Fact
1	nsDOMClassInfo	3
2	SGridRowLayout	95
3	xpcprivate	6
4	jsxml	2
5	nsGenericHTMLElement	8
6	jsgc	3
7	nsISEnvironment	12
8	jsfun	1
9	nsHTMLLabelElement	18
10	nsHttpTransaction	35

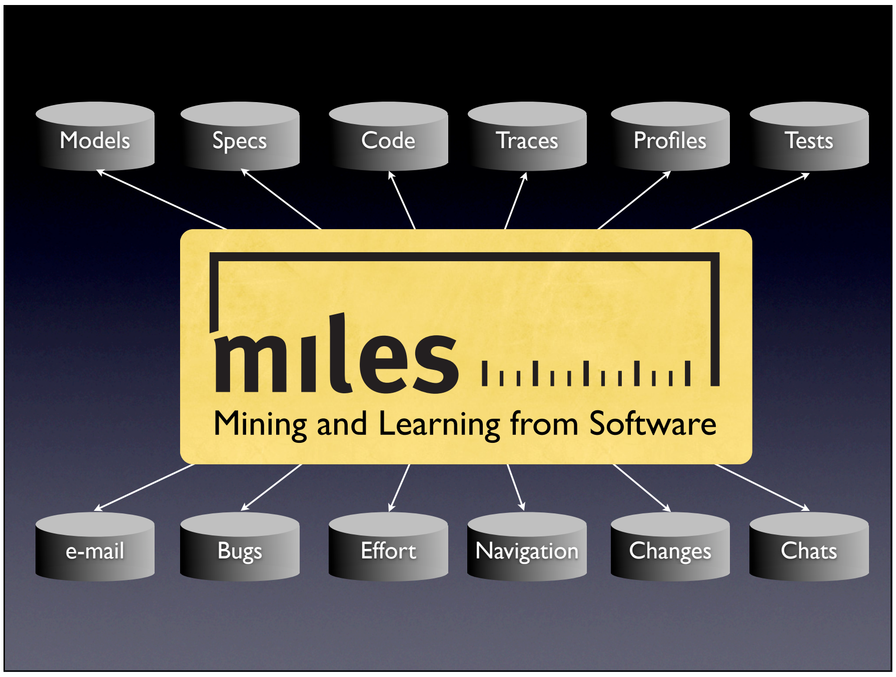


Software Archives

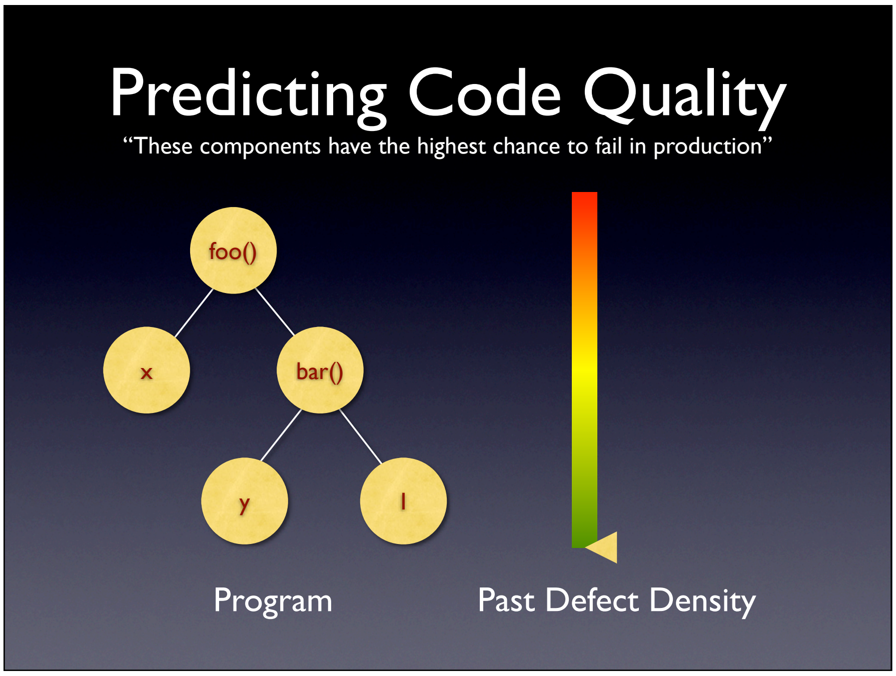
- contain full record of project history
- maintained via programming environments
- automatic maintenance and access
- freely accessible in open source projects

Bugs
Changes

This was just a simple example. So, the most important aspect that software archives give you is automation. They are maintained automatically ("The data comes to you"), and they can be evaluated automatically ("Instantaneous results"). For researchers, there are plenty open source archives available, allowing us to test, compare, and evaluate our tools.



Combining these sources will allow us to get this "waterfall effect" – that is, being submerged by data; having more data than we could possibly digest.

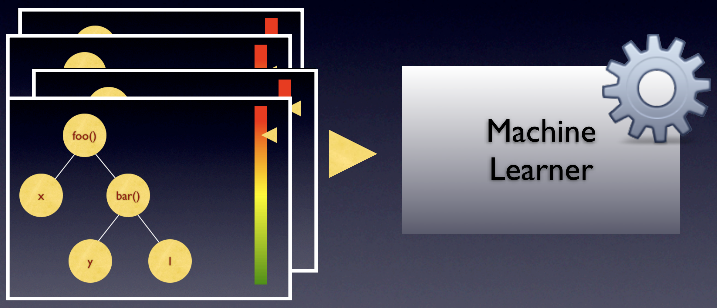


Predicting Code Quality

"These components have the highest chance to fail in production"

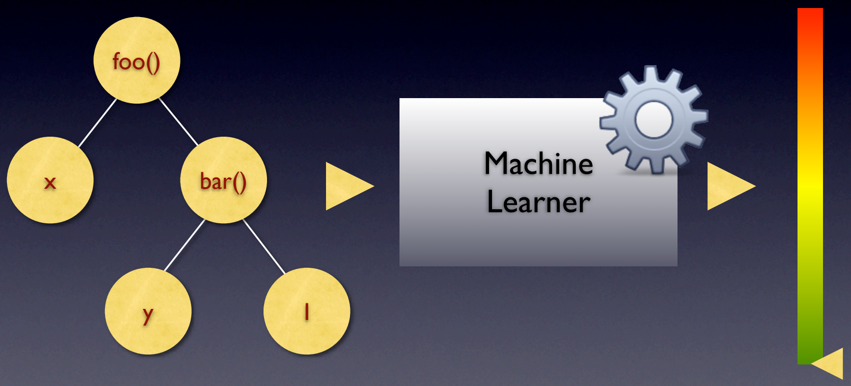
Predicting Code Quality

"These components have the highest chance to fail in production"



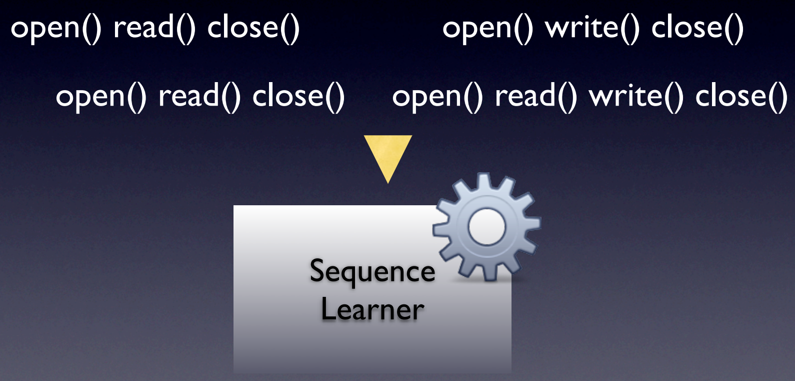
Predicting Code Quality

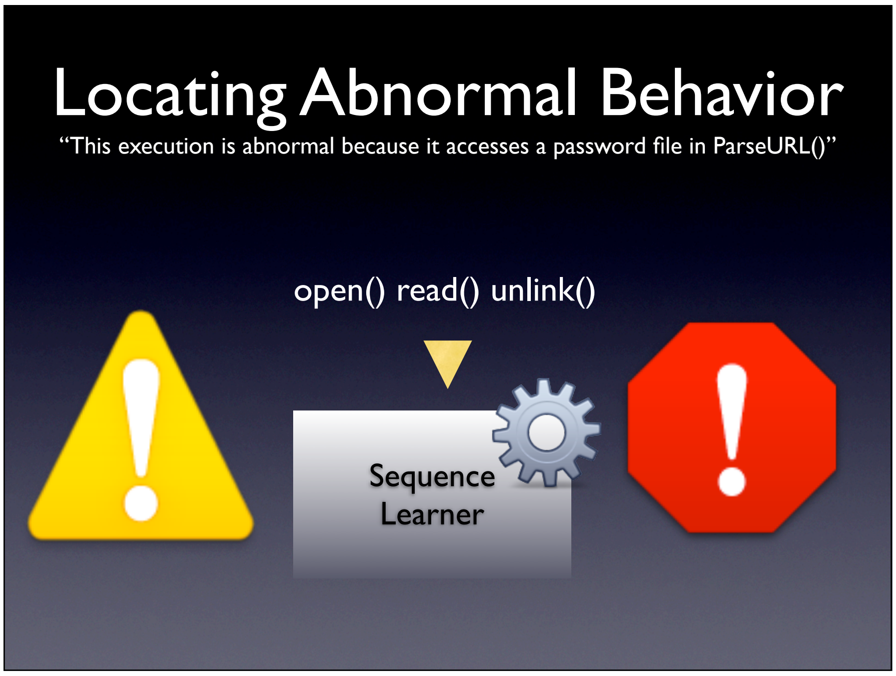
"These components have the highest chance to fail in production"



Locating Abnormal Behavior

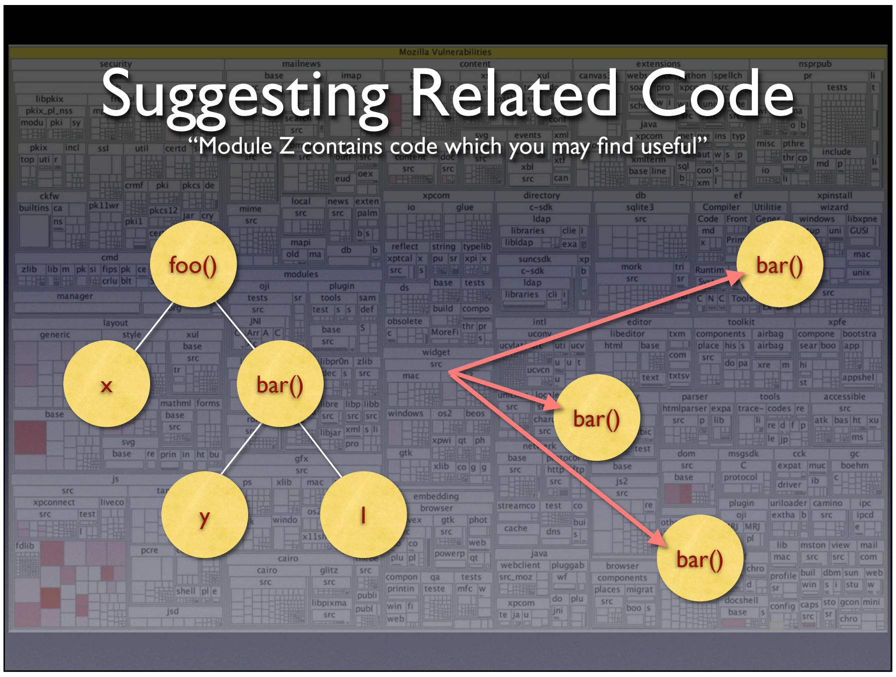
"This execution is abnormal because it accesses a password file in ParseURL()"





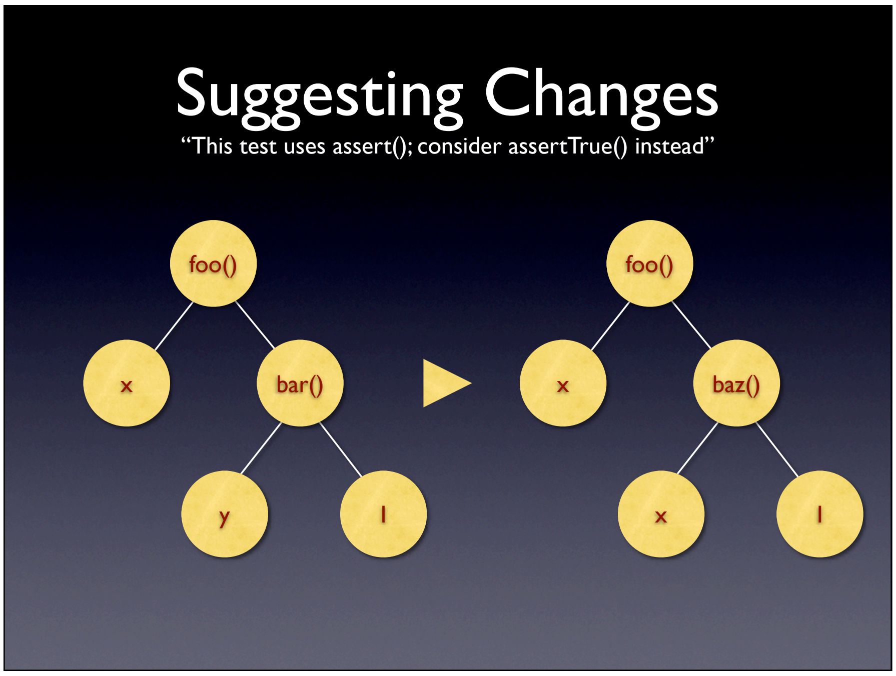
Locating Abnormal Behavior

"This execution is abnormal because it accesses a password file in ParseURL()"



Suggesting Related Code

"Module Z contains code which you may find useful"

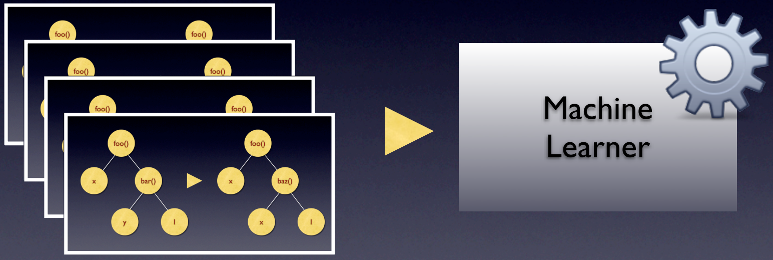


Suggesting Changes

"This test uses assert(); consider assertTrue() instead"

Suggesting Changes

"This test uses assert(); consider assertTrue() instead"



Linking Artifacts

"This workaround is due to our customer's requirement from December 12"

```
public class Purse {
  final int MAX_BALANCE;
  int balance;
  //@ invariant  0 ≤ balance && balance ≤ MAX_BALANCE;

  byte[] pin;
  /*@ invariant pin != null && pin.length == 4 &&
    @           (\forall int i; 0 ≤ i && i < 4;
    @                         0 ≤ byte[i] && byte[i] ≤ 9)
    @*/

  /*@ requires    amount ≥ 0;
    @ assignable balance;
    @ ensures     balance == \old(balance) - amount &&
    @             \result == balance;
    @ signals     (PurseException) balance == \old(balance);
    @*/
  int debit(int amount) throws PurseException { … }
```

Linking Artifacts

"This workaround is due to our customer's requirement from December 12"

```
public class Purse {
  final int MAX_BALANCE;
  int balance;
  //@ invariant  0 ≤ balance &

  byte[] pin;
  /*@ invariant pin != null &&
    @           (\forall int i; 0 ≤ i && i < 4;
    @                         0 ≤ byte[i] && byte[i] ≤ 9)
    @*/

  /*@ requires    amount ≥ 0;
    @ assignable balance;
    @ ensures     balance == \old(balance) - amount &&
    @             \result == balance;
    @ signals     (PurseException) balance == \old(balance);
    @*/
  int debit(int amount) throws PurseException { … }
```

Banking

Purse • balance • PIN • debit…

Linking Artifacts

"This workaround is due to our customer's requirement from December 12"

Banking

Purse • balance • PIN • debit…

When retrieving money from an ATM, the customer inserts his card and enters a PIN (a 4-digit number) and the amount to be retrieved…

Linking Artifacts

"This workaround is due to our customer's requirement from December 12"

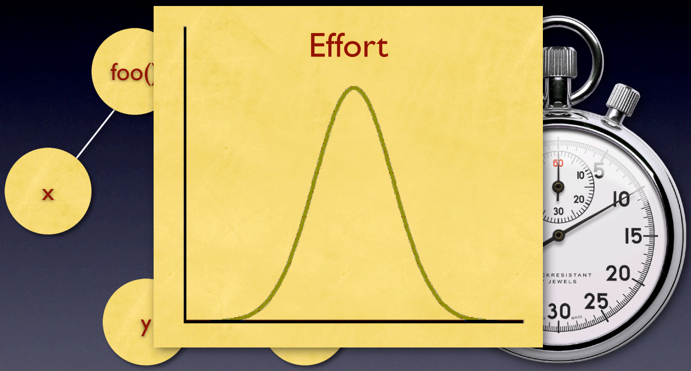
Banking

Purse • balance • PIN • debit…

When retrieving money from an ATM, the customer inserts his card and enters a PIN (a 4-digit number) and the amount to be retrieved…

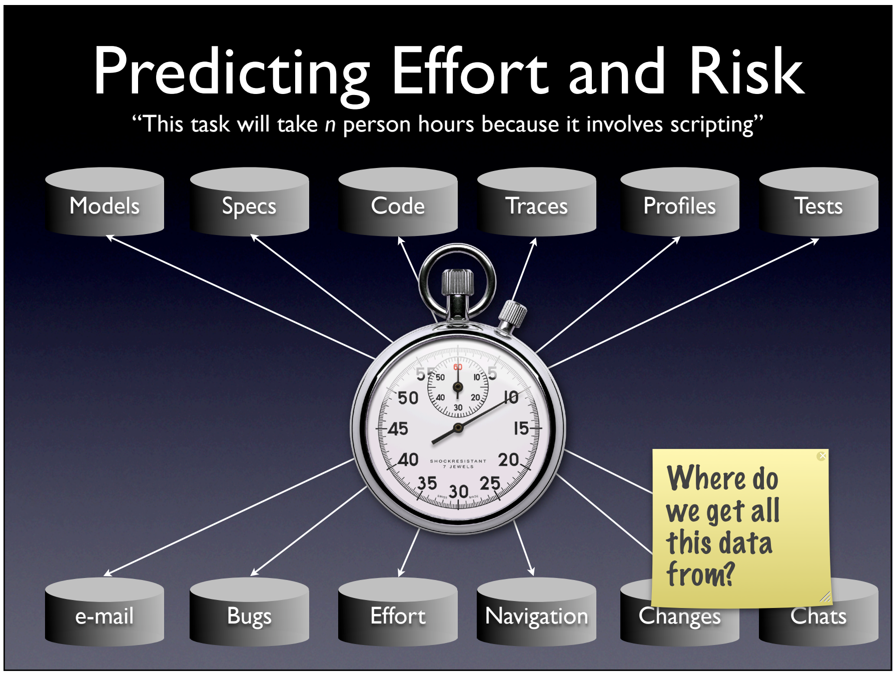
Predicting Effort and Risk

"This task will take *n* person hours because it involves scripting"



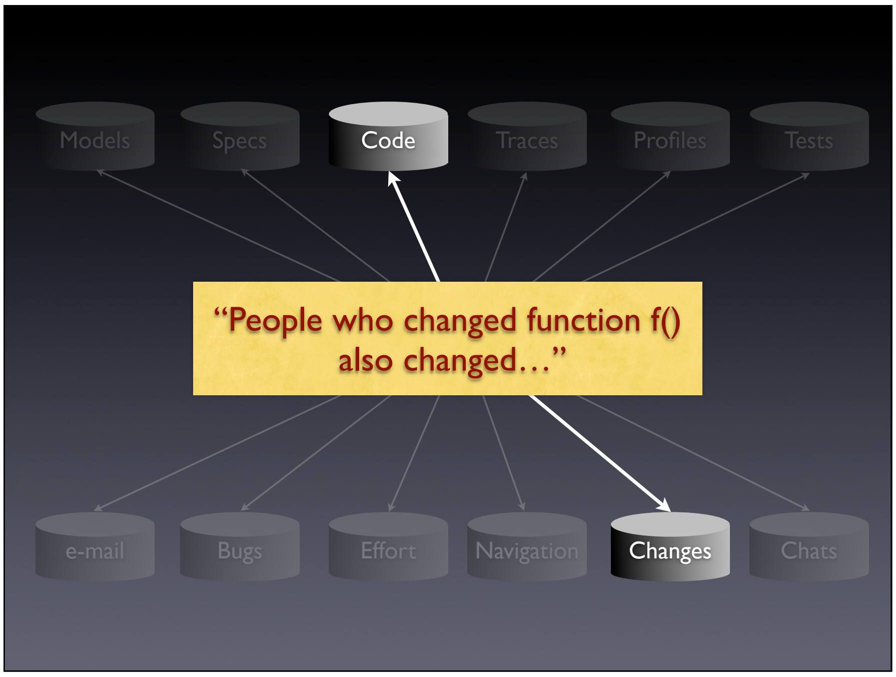
Program

Past Effort



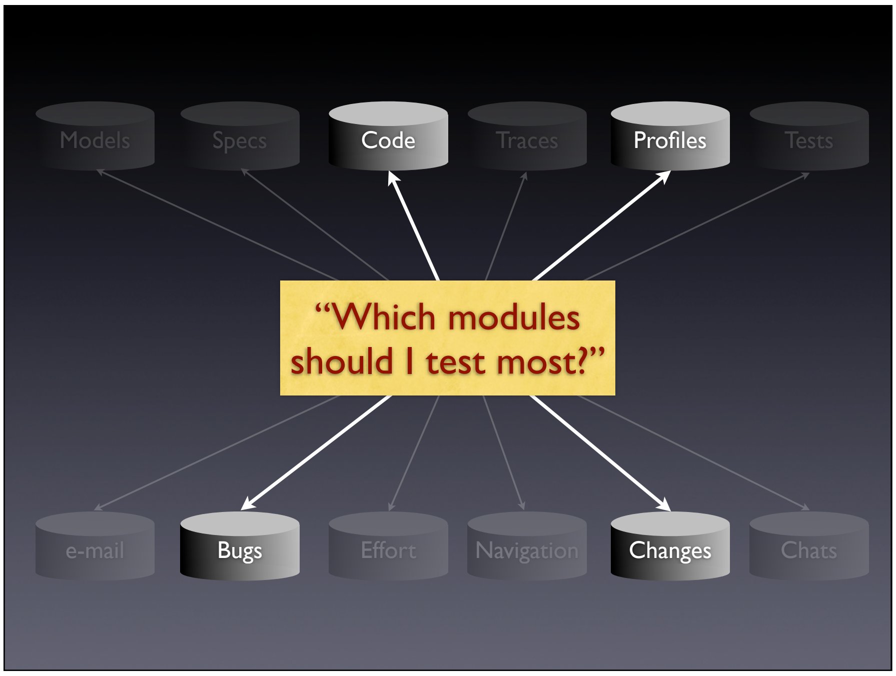
Predicting Effort and Risk

"This task will take *n* person hours because it involves scripting"



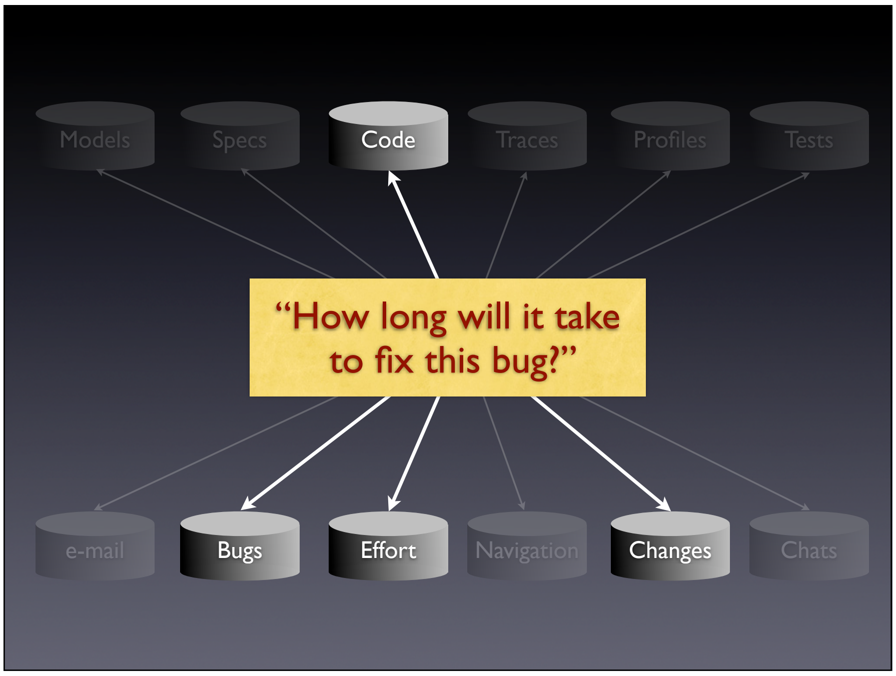
"People who changed function f() also changed…"

This is the oldest example, referring to work by Tom Zimmermann et al. at ICSE 2004 (and the work of Annie Ying et al. at the same time): You change one function – which others should be changed? This is easy to mine drawing on the change history and the code.

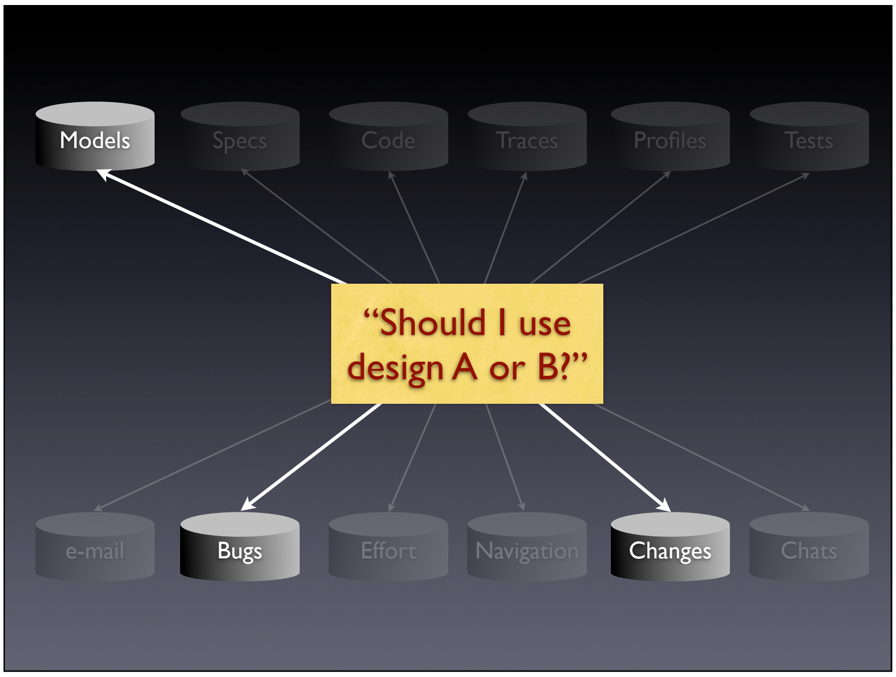


"Which modules should I test most?"

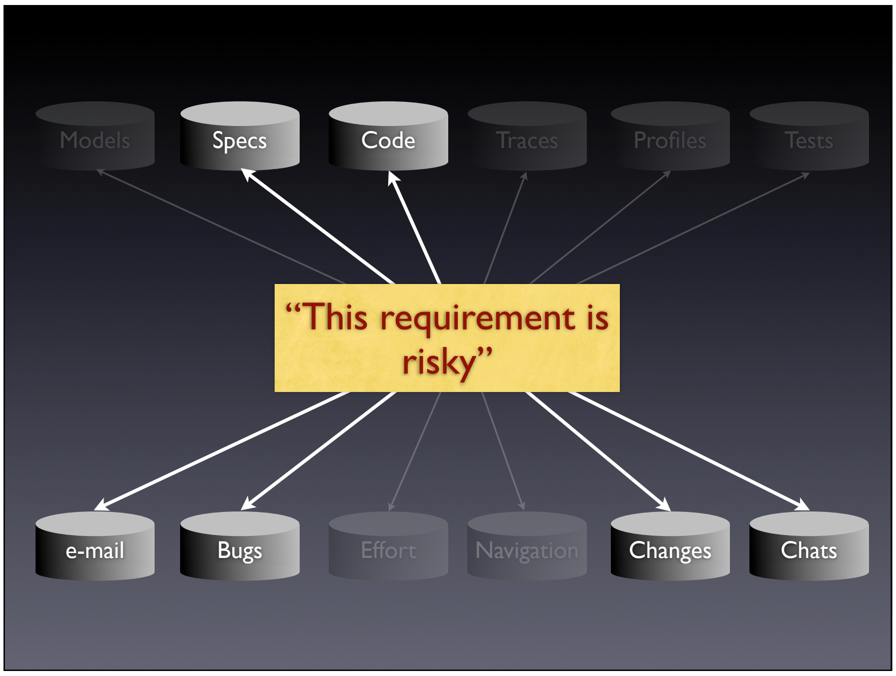
Defect density data as sketched before can be used to decide where to test most – of course, where the most defects are. If one additionally takes profiles (e.g. usage data) into account, one can even allocate test efforts to minimize the predicted potential damage optimally.



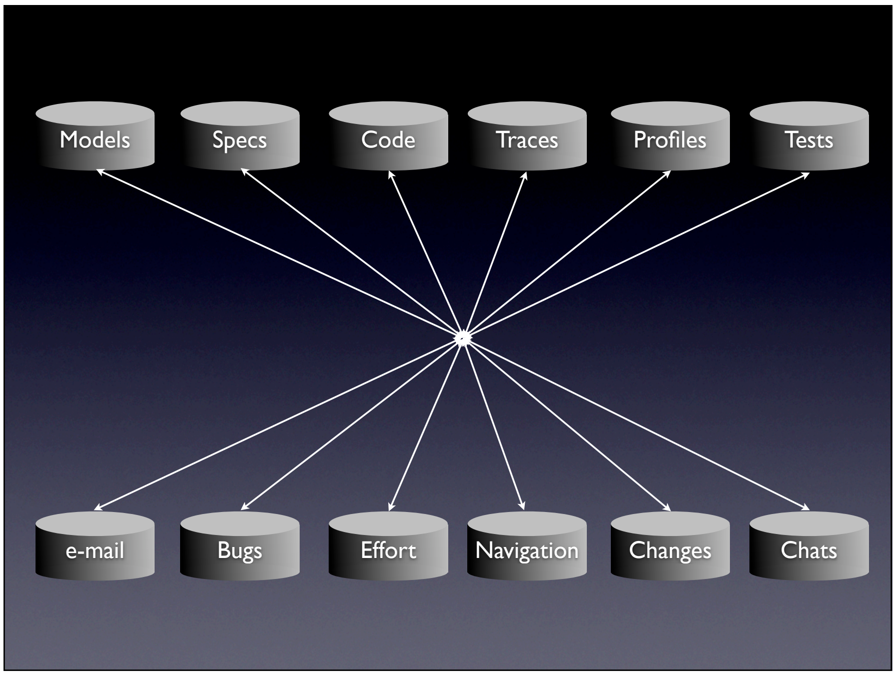
If one has effort data, one can tell how long it takes to fix a bug. Cathrin Weiß has a talk on this topic right after this keynote.



If one knows which program features correlate with which quality, one can use this measure to make all kinds of decisions. Correlating design with failure probability will help making well-founded design decisions. This is not to say that managers can’t do this right now, but having accurate project data available can certainly help assess the risks.



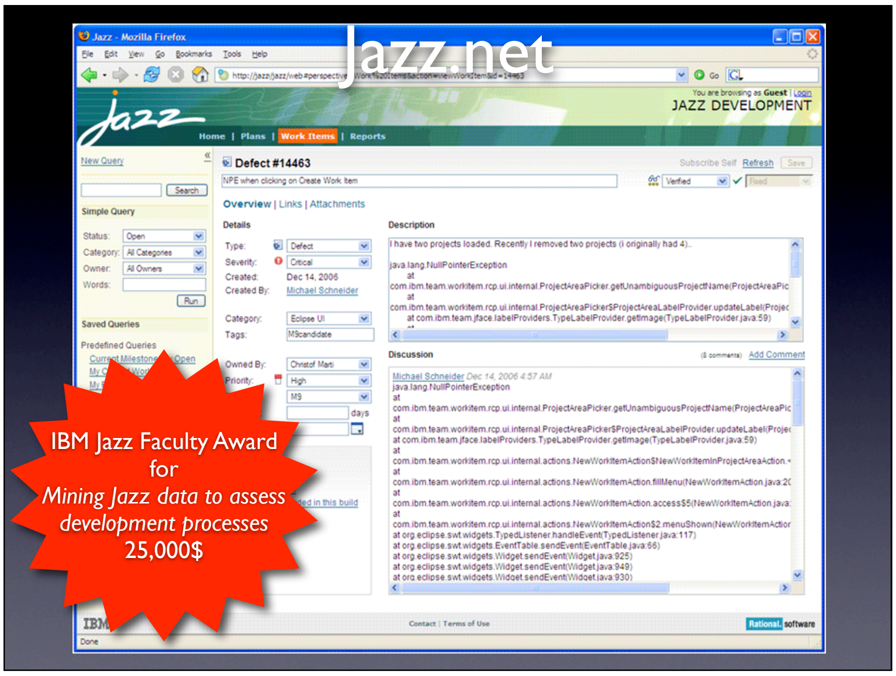
Finally, a glimpse into the future, taking natural language resources into account. The idea is to associate specs with (natural language) topics, and to map these topics to source code. What you then get is an idea of how specific topics (or keywords) influence failure probability, and this will allow you making predictions for specific requirements.



Combining these sources will allow us to get this “waterfall effect” – that is, being submerged by data; having more data than we could possibly digest.

The dirty story about this data is that it is frequently collected manually. In fact, the company phone book is among the most important tools of an empirical software engineering researchers. One would phone one developer after the other, and question them – say, “what was your effort”, or “how often did you test module ‘foo’?”, and tick in the appropriate form. In other words, data is scarce, and as it is being collected from humans after the fact, is prone to errors, and prone to bias.

Jazz.net



Jazz - Mozilla Firefox

File Edit View Go Bookmarks Tools Help

http://jazz/jazz/web#perspective=Work%20Items&action=viewWorkItem&id=14463

You are browsing as **Guest** | Login

JAZZ DEVELOPMENT

Home | Plans | Work Items | Reports

New Query

Search

Simple Query

Status:	Open
Category:	All Categories
Owner:	All Owners
Words:	

Run

Saved Queries

Predefined Queries

Current Milestone ... Open

Defect #14463

NPE when clicking on Create Work Item

Subscribe Self Refresh Save

Verified Fixed

Overview | Links | Attachments

Details

Type:	Defect
Severity:	Critical
Created:	Dec 14, 2006
Created By:	Michael Schneider
Category:	Eclipse UI
Tags:	M9candidate
Owned By:	Christof Marti
Priority:	High
	M9
	days

Description

```
I have two projects loaded. Recently I removed two projects (i originally had 4)..

java.lang.NullPointerException
	at
com.ibm.team.workitem.rcp.ui.internal.ProjectAreaPicker.getUnambiguousProjectName(ProjectAreaPic
	at
com.ibm.team.workitem.rcp.ui.internal.ProjectAreaPicker$ProjectAreaLabelProvider.updateLabel(Projec
	at com.ibm.team.jface.labelProviders.TypeLabelProvider.getImage(TypeLabelProvider.java:59)
```

Discussion (8 comments) Add Comment

```
Michael Schneider Dec 14, 2006 4:57 AM
java.lang.NullPointerException
at
com.ibm.team.workitem.rcp.ui.internal.ProjectAreaPicker.getUnambiguousProjectName(ProjectAreaPic
at
com.ibm.team.workitem.rcp.ui.internal.ProjectAreaPicker$ProjectAreaLabelProvider.updateLabel(Projec
at com.ibm.team.jface.labelProviders.TypeLabelProvider.getImage(TypeLabelProvider.java:59)
at
com.ibm.team.workitem.rcp.ui.internal.actions.NewWorkItemAction$NewWorkItemInProjectAreaAction.<
at
com.ibm.team.workitem.rcp.ui.internal.actions.NewWorkItemAction.fillMenu(NewWorkItemAction.java:20
at
com.ibm.team.workitem.rcp.ui.internal.actions.NewWorkItemAction.access$5(NewWorkItemAction.java:
at
com.ibm.team.workitem.rcp.ui.internal.actions.NewWorkItemAction$2.menuShown(NewWorkItemActior
at org.eclipse.swt.widgets.TypedListener.handleEvent(TypedListener.java:117)
at org.eclipse.swt.widgets.EventTable.sendEvent(EventTable.java:66)
at org.eclipse.swt.widgets.Widget.sendEvent(Widget.java:925)
at org.eclipse.swt.widgets.Widget.sendEvent(Widget.java:949)
at org.eclipse.swt.widgets.Widget.sendEvent(Widget.java:930)
```

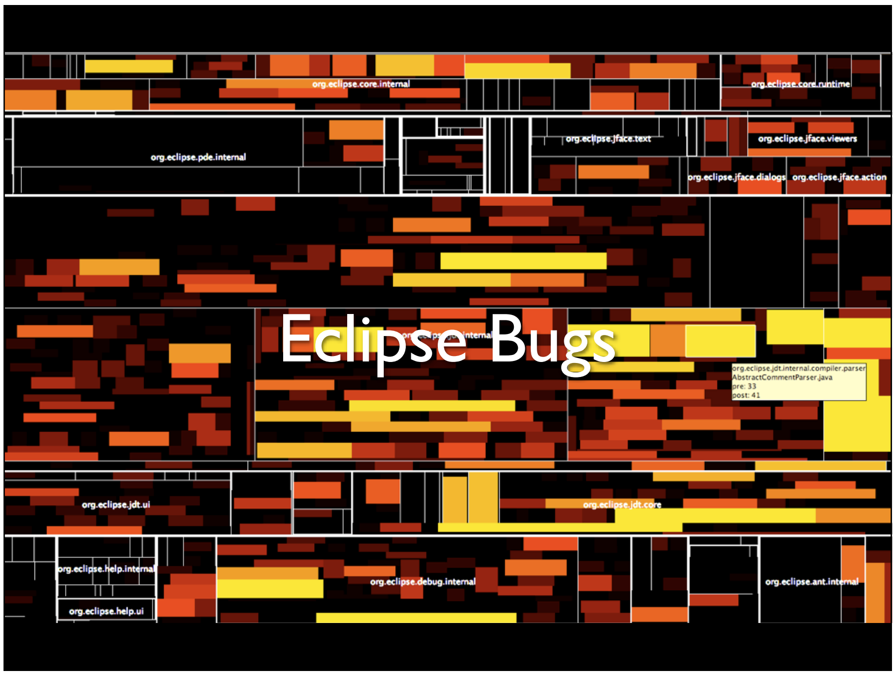
IBM Contact | Terms of Use Rational. software

Done

IBM Jazz Faculty Award
for
Mining Jazz data to assess development processes
25,000$

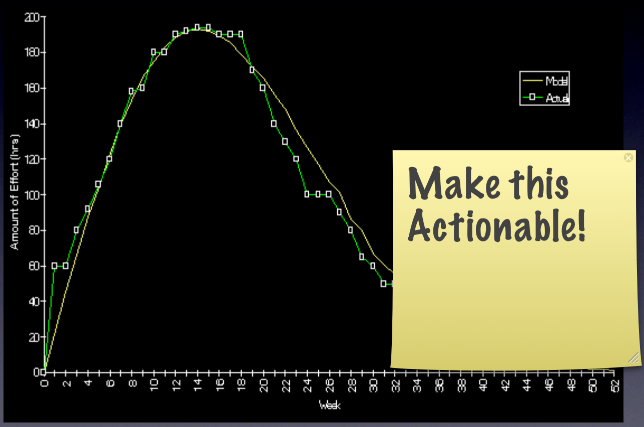
Combining these sources will allow us to get this “waterfall effect” – that is, being submerged by data; having more data than we could possibly digest.

Eclipse Bugs



This is what you get when doing such a mapping for eclipse. Each class is a rectangle in here (the larger the rectangle, the larger its code); the colors tell the defect density – the brighter a rectangle, the more defects were fixed in here. Interesting question: Why are come modules so much more defect-prone than others? This is what has kept us busy for years now.

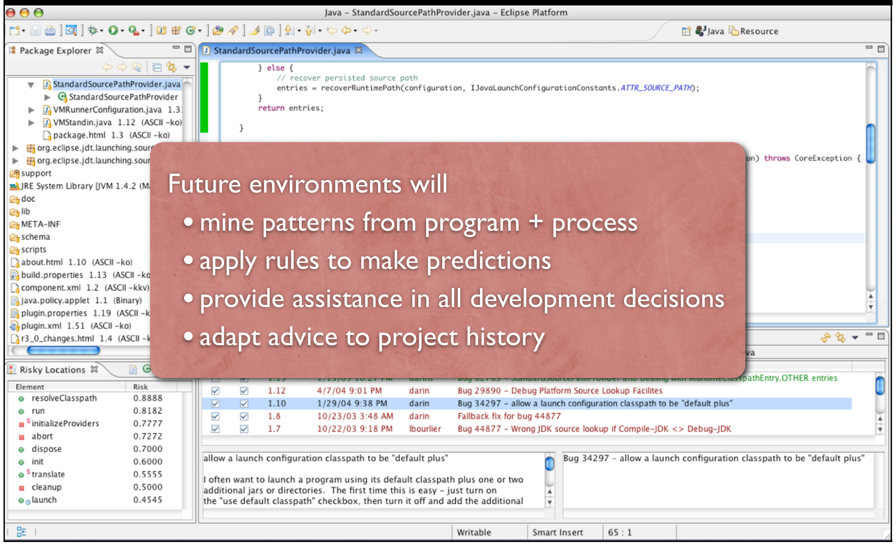
Studies



Rosenberg, L. and Hyatt, L. "Developing An Effective Metrics Program"
European Space Agency Software Assurance Symposium, Netherlands, March, 1996

Let's now talk about results. What should our tools do? Should they come up with nice reports, and curves like this one?

Assistance



Java – StandardSourcePathProvider.java – Eclipse Platform

Package Explorer

- StandardSourcePathProvider.java
  - StandardSourcePathProvider
- VMRunnerConfiguration.java 1.3
- VMStandin.java 1.12 (ASCII -ko)
- package.html 1.3 (ASCII -ko)
- org.eclipse.jdt.launching.sour
- org.eclipse.jdt.launching.sour
- support
- JRE System Library [JVM 1.4.2 (M
- doc
- lib
- META-INF
- schema
- scripts
- about.html 1.10 (ASCII -ko)
- build.properties 1.13 (ASCII -ko
- component.xml 1.2 (ASCII -kkv)
- java.policy.applet 1.1 (Binary)
- plugin.properties 1.19 (ASCII -k
- plugin.xml 1.51 (ASCII -ko)
- r3_0_changes.html 1.4 (ASCII -k

StandardSourcePathProvider.java

```
		} else {
			// recover persisted source path
			entries = recoverRuntimePath(configuration, IJavaLaunchConfigurationConstants.ATTR_SOURCE_PATH);
		}
		return entries;

	}
```

on) throws CoreException {

Future environments will
- mine patterns from program + process
- apply rules to make predictions
- provide assistance in all development decisions
- adapt advice to project history

Risky Locations

Element	Risk
resolveClasspath	0.8888
run	0.8182
initializeProviders	0.7777
abort	0.7272
dispose	0.7000
init	0.6000
translate	0.5555
cleanup	0.5000
launch	0.4545

```
☑ ☑ 1.12  4/7/04 9:01 PM     darin      Bug 29890 – Debug Platform Source Lookup Facilites
☑ ☑ 1.10  1/29/04 9:38 PM    darin      Bug 34297 – allow a launch configuration classpath to be "default plus"
☑ ☑ 1.8   10/23/03 3:48 AM   darin      Fallback fix for bug 44877
☑ ☑ 1.7   10/22/03 9:18 PM   lbourlier  Bug 44877 – Wrong JDK source lookup if Compile-JDK <> Debug-JDK
```

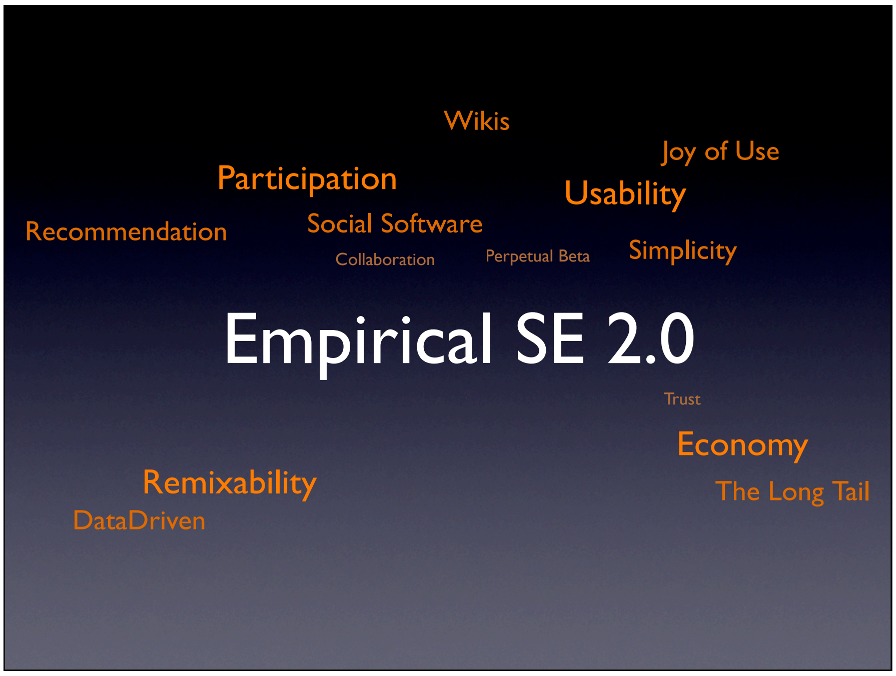
allow a launch configuration classpath to be "default plus"

I often want to launch a program using its default classpath plus one or two additional jars or directories. The first time this is easy – just turn on the "use default classpath" checkbox, then turn it off and add the additional

Bug 34297 – allow a launch configuration classpath to be "default plus"

Writable | Smart Insert | 65 : 1

Programming environments also are the tools that allow us to collect, maintain, and integrate all this project data. This is where the waterfall becomes imminent. In pair programming, you have a navigator peering over your shoulder, giving you advice whether what you are doing is good or bad. We want the environment peer over your shoulder – as an automated "developer's buddy". Whatever we do must stand the test of the developers – if they accept it, it will be good enough.



Empirical SE 2.0

Wikis

Joy of Use

Participation

Usability

Recommendation

Social Software

Collaboration

Perpetual Beta

Simplicity

Trust

Economy

Remixability

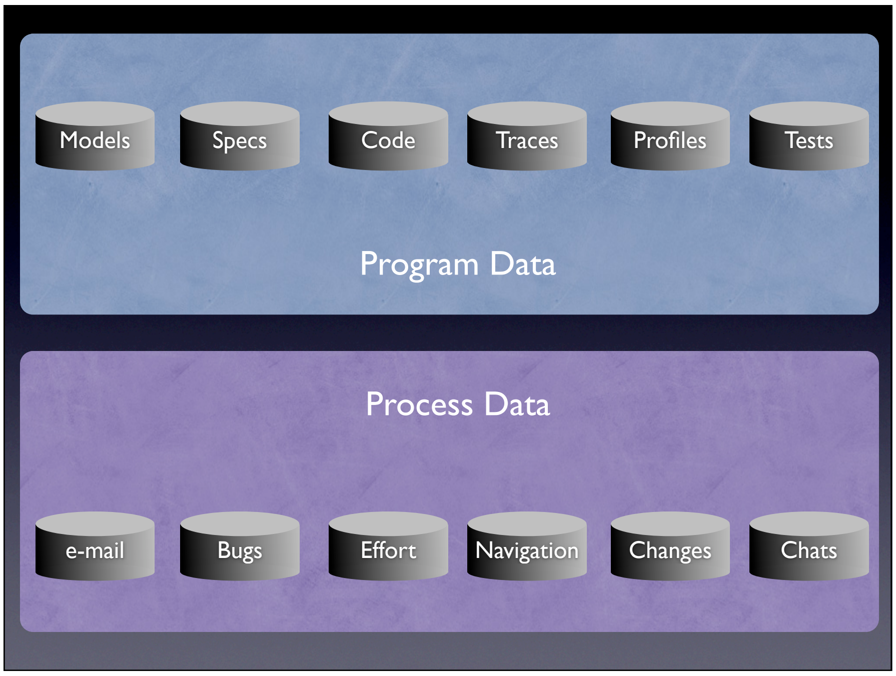
The Long Tail

DataDriven

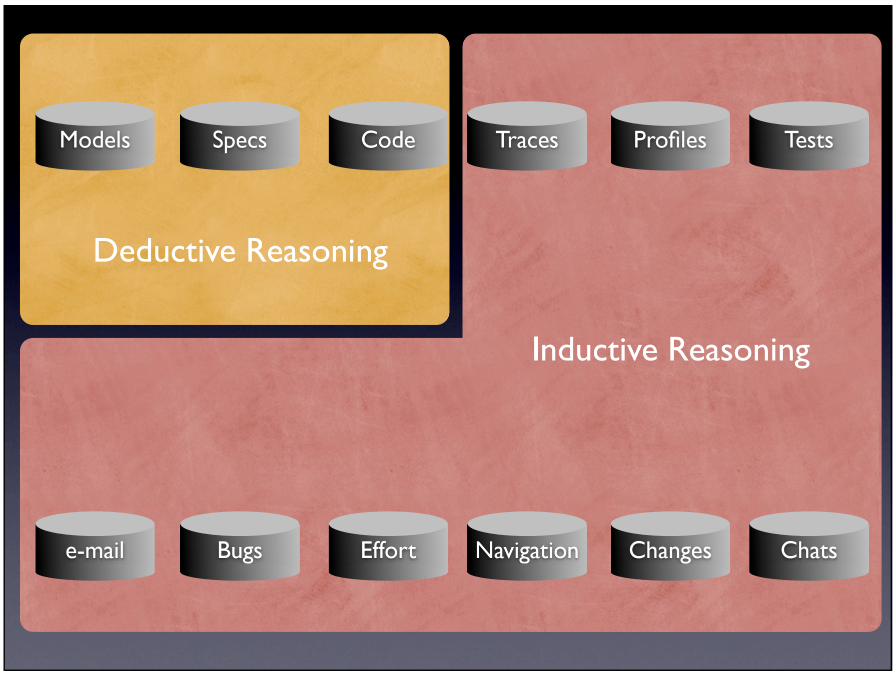
…and thus realizing the concept of Empirical Software Engineering 2.0. You will find traces of all these concepts in my talk – from participation over usability and remixability to, hopefully, economic consequences.

Challenges

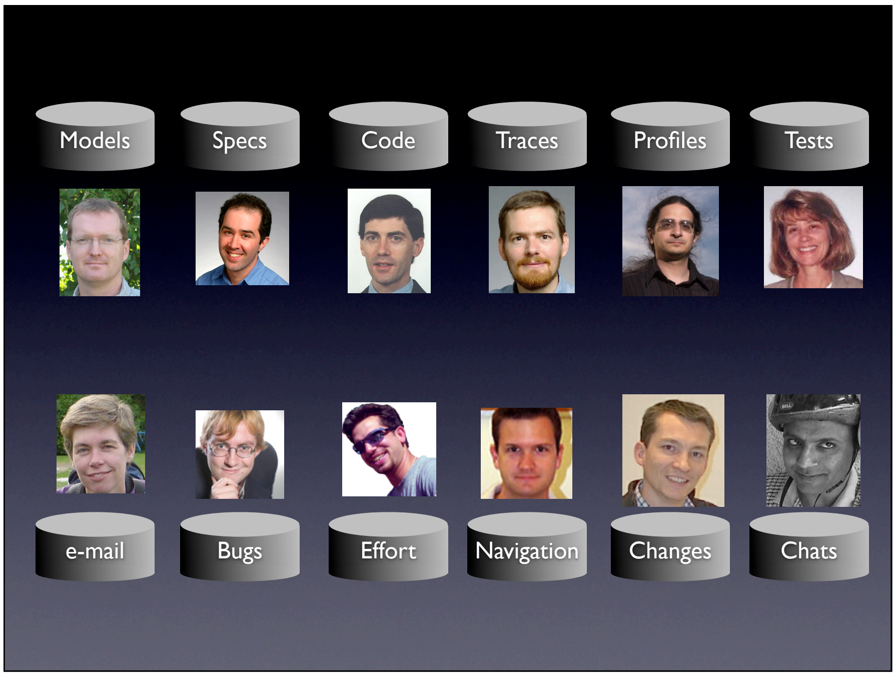
In order to get there, we have plenty of challenges to overcome.



To start with, half of the data is related to programs, the other half to processes. People analyzing programs are not necessarily process experts, and vice versa.

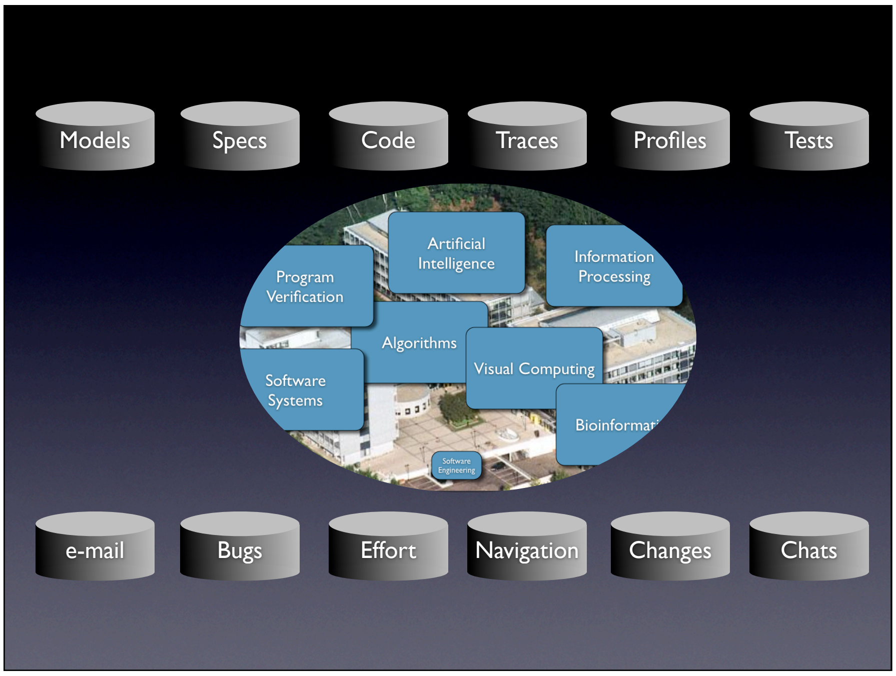


Also, we have huge differences in terms of methods. For code and models, we use deductive reasoning, predicting what can happen in the concrete by analyzing the abstraction. In the other areas, it is the other way round: From collected data, we build abstractions that capture patterns and rules. These two methods are hard to bring together.

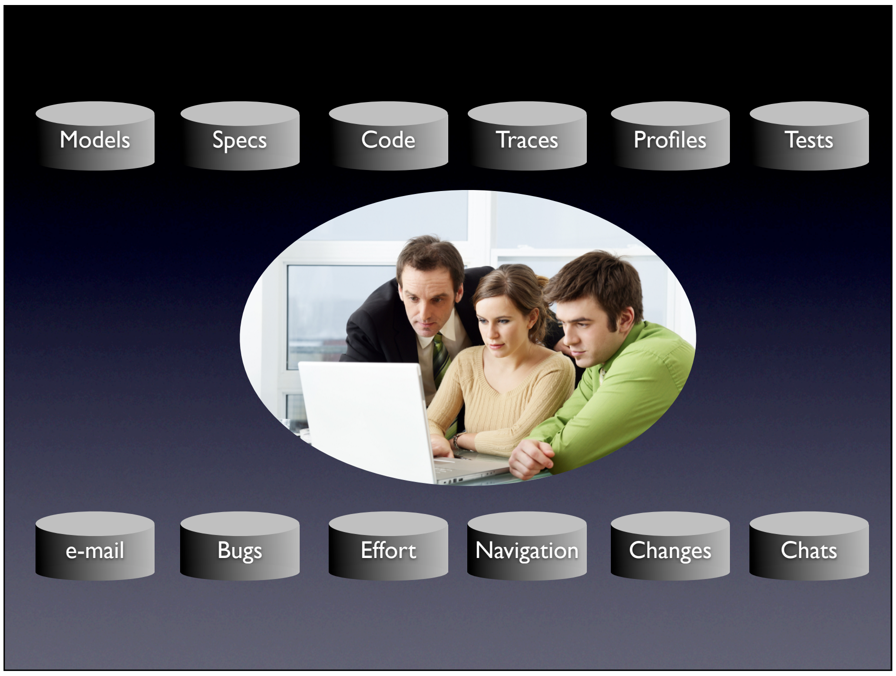


In the past, all of this data has been processed by individual researchers. Each of these faces stands for an entire community, sometimes encompassing thousands of researchers.

Matt Dwyer - Daniel Jackson - Tom Reps - Mike Ernst - Ben Liblit - Mary Jean Harrold - Gail Murphy - Tom Zimmermann - Cathrin Weiß - Rob DeLine - Harald Gall - Prem Devanbu



And to bring the data together, we need to bring together the researchers. What better place could there be than ICSE or this workshop for this purpose?



And to bring the data together, we need to bring together the researchers. What better place could there be than ICSE or this workshop for this purpose?

Combining these sources will allow us to get this "waterfall effect" – that is, being submerged by data; having more data than we could possibly digest.