

Back to Software Traceability

Many traditional Software Traceability approaches rely on processes that “enforce” traceability

- Through rigid processes - manual/automated linking of software artifacts at creation time or as part of quality assurance.
- Through the use of document driven approaches

Question:

But how can we deal with traceability in agile development contexts, where documentation is of lesser importance or not available at all (e.g., agile processes, open source development)?



Mining Software Repositories

- The following lecture notes are a subset of the notes available at:

<http://ase.csc.ncsu.edu/dmse/>

Mining Software Engineering Data

Ahmed E. Hassan

Queen's University

www.cs.queensu.ca/~ahmed

ahmed@cs.queensu.ca

Tao Xie

North Carolina State University

www.csc.ncsu.edu/faculty/xie

xie@csc.ncsu.edu

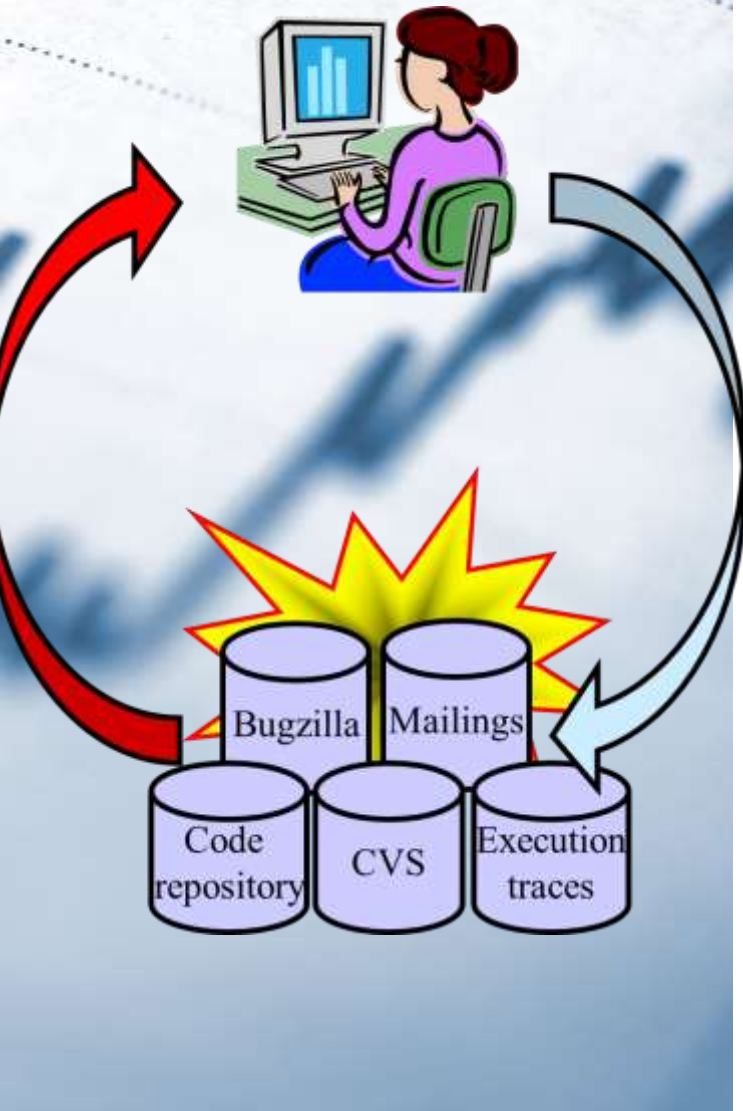
Some slides are adapted from tutorial slides co-prepared by Jian Pei
from Simon Fraser University, Canada

An up-to-date version of this tutorial is available at
<http://ase.csc.ncsu.edu/dmse/dmse-icse08-tutorial.pdf>

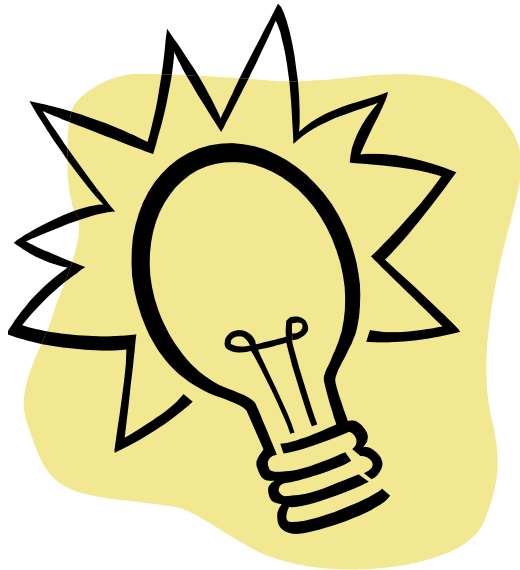
Mining SE Data

- **MAIN GOAL**

- Transform static record-keeping SE data to **active** data
- Make SE data actionable by uncovering hidden **patterns** and **trends**

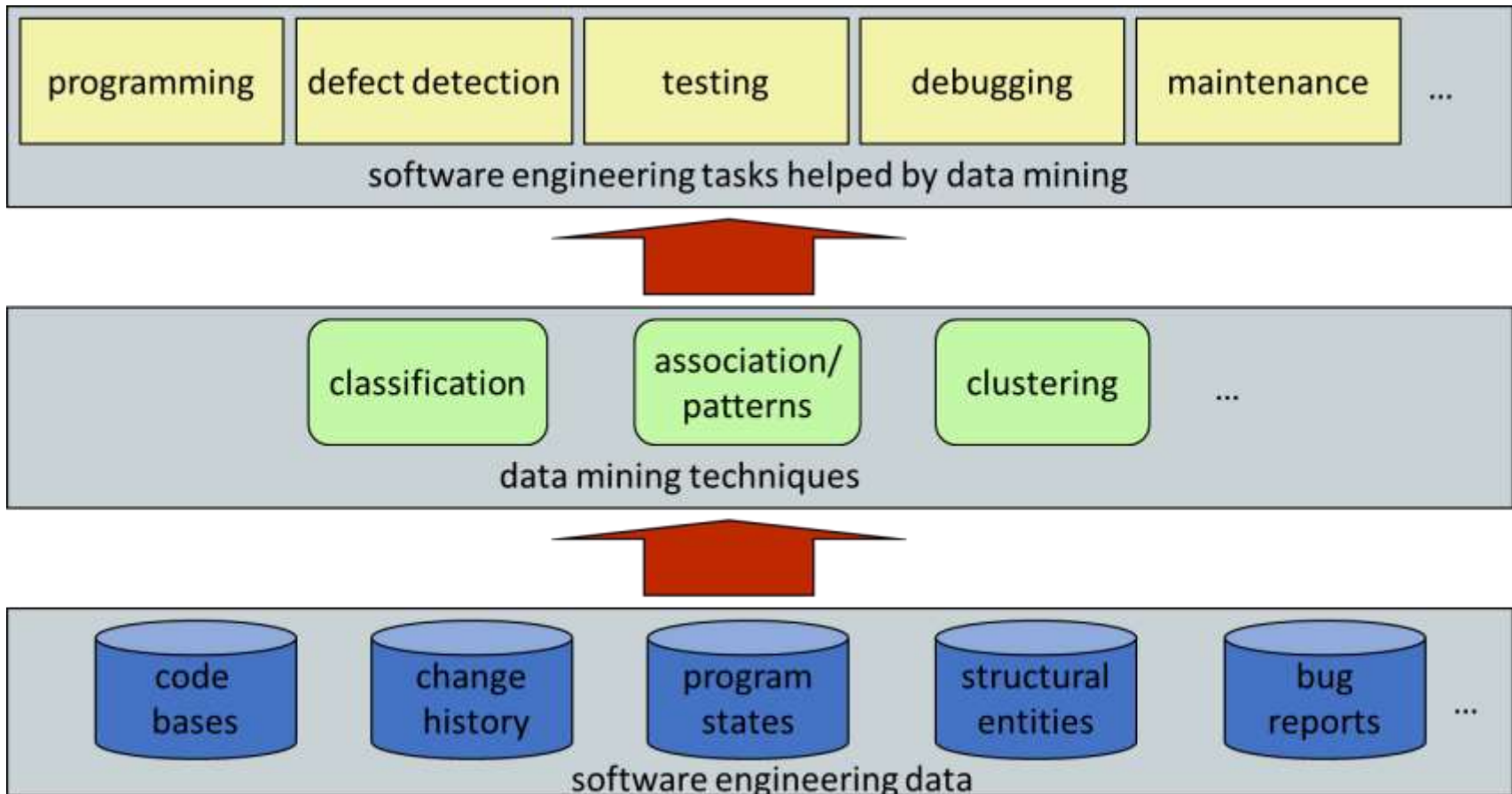


Mining SE Data



- SE data can be used to:
 - Gain empirically-based understanding of software development
 - Predict, plan, and understand various aspects of a project
 - Support future development and project management activities

Overview of Mining SE Data



Types of SE Data

- **Historical data**

- Version or source control: cvs, subversion, perforce
- Bug systems: bugzilla, GNATS, JIRA
- Mailing lists: mbox

- **Multi-run and multi-site data**

- Execution traces
- Deployment logs

- **Source code data**

- Source code repositories: sourceforge.net, google code





Historical Data

“History is a guide to navigation in perilous times. History is who we are and why we are the way we are.”

-

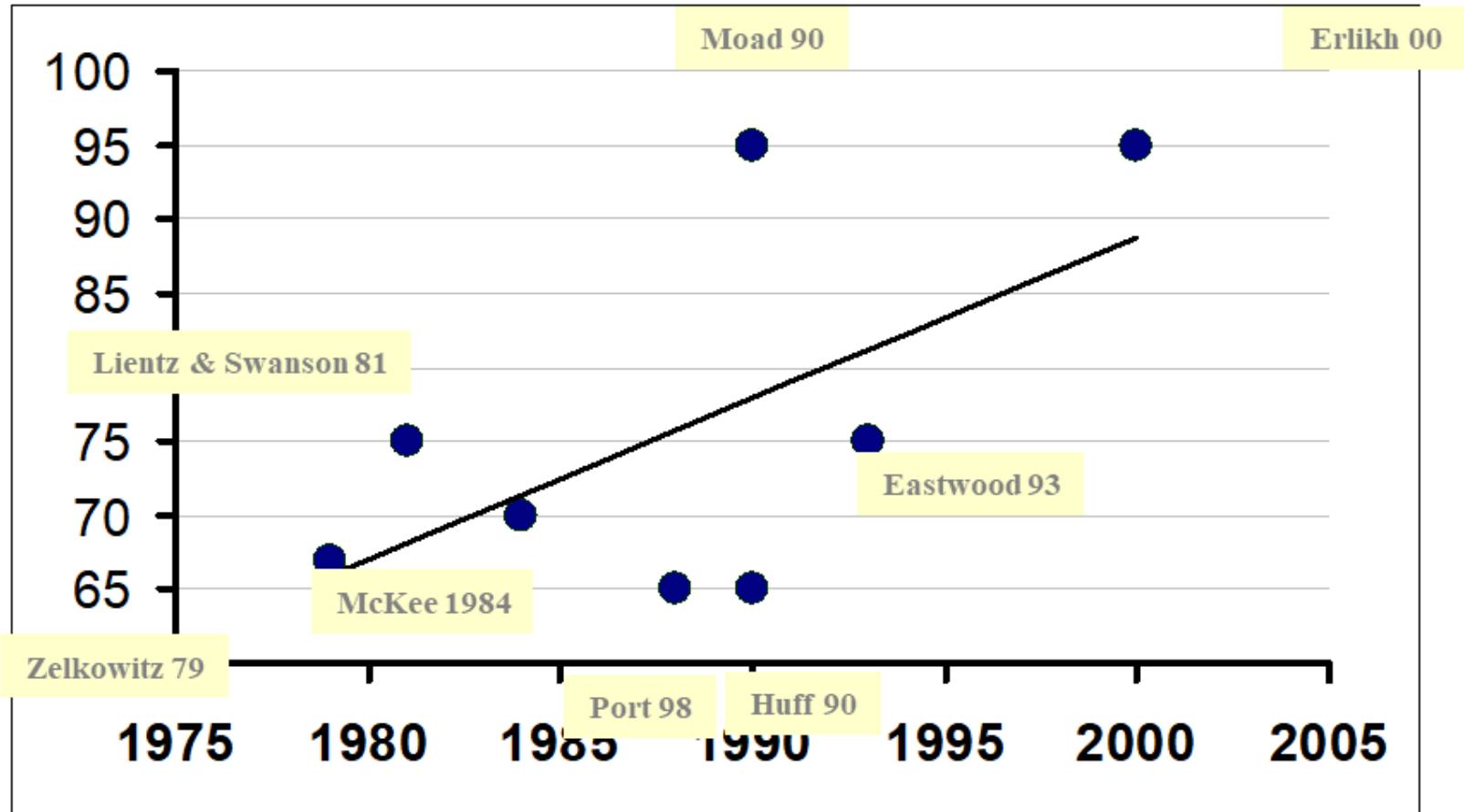
David C. McCullough

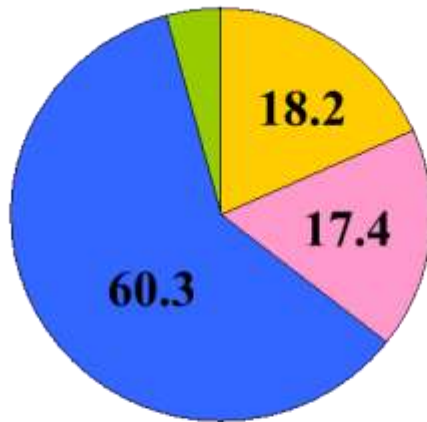
Historical Data

- Track the evolution of a software project:
 - *source control systems* store changes to the code
 - *defect tracking systems* follow the resolution of defects
 - *archived project communications* record rationale for decisions throughout the life of a project
- Used primarily for record-keeping activities:
 - checking the status of a bug
 - retrieving old code

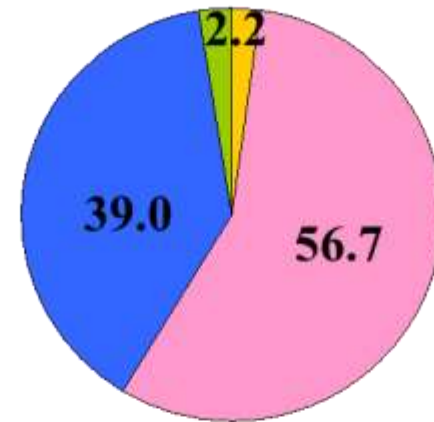


Percentage of Project Costs Devoted to Maintenance





Lientz, Swanson, Tomhkins [1978]
Nosek, Palvia [1990]
MIS Survey



Schach, Jin, Yu, Heller, Offutt [2003]
Mining ChangeLogs
(Linux, GCC, RTP)

Survey of Software Maintenance Activities

- **Perfective:** add new functionality
- **Corrective:** fix faults
- **Adaptive:** new file formats, refactoring

Source Control Repositories

Source Control Repositories

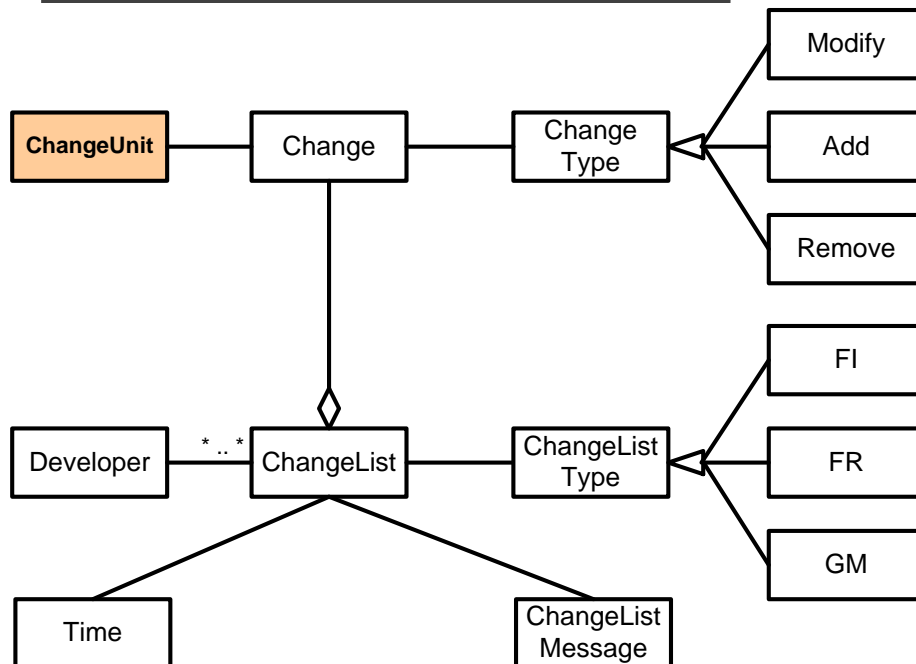
A source control system tracks changes to ChangeUnits

Example of ChangeUnits:

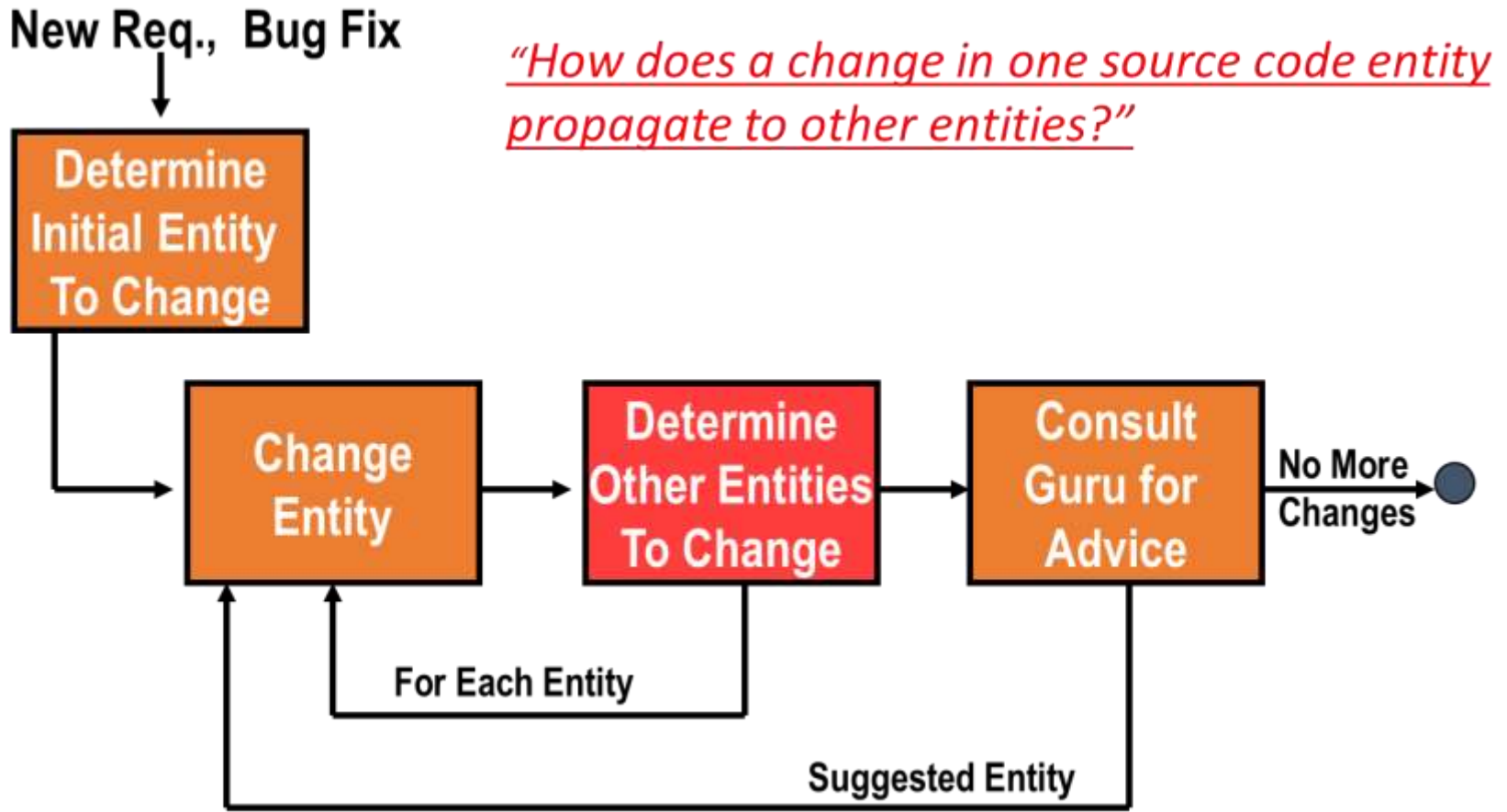
- File (most common)
- Function
- Dependency (e.g., Call)

Each ChangeUnit:

- Records the developer, change time, change message, co-changing Units



Change Propagation



Guiding Change Propagation

Mine association rules from change history

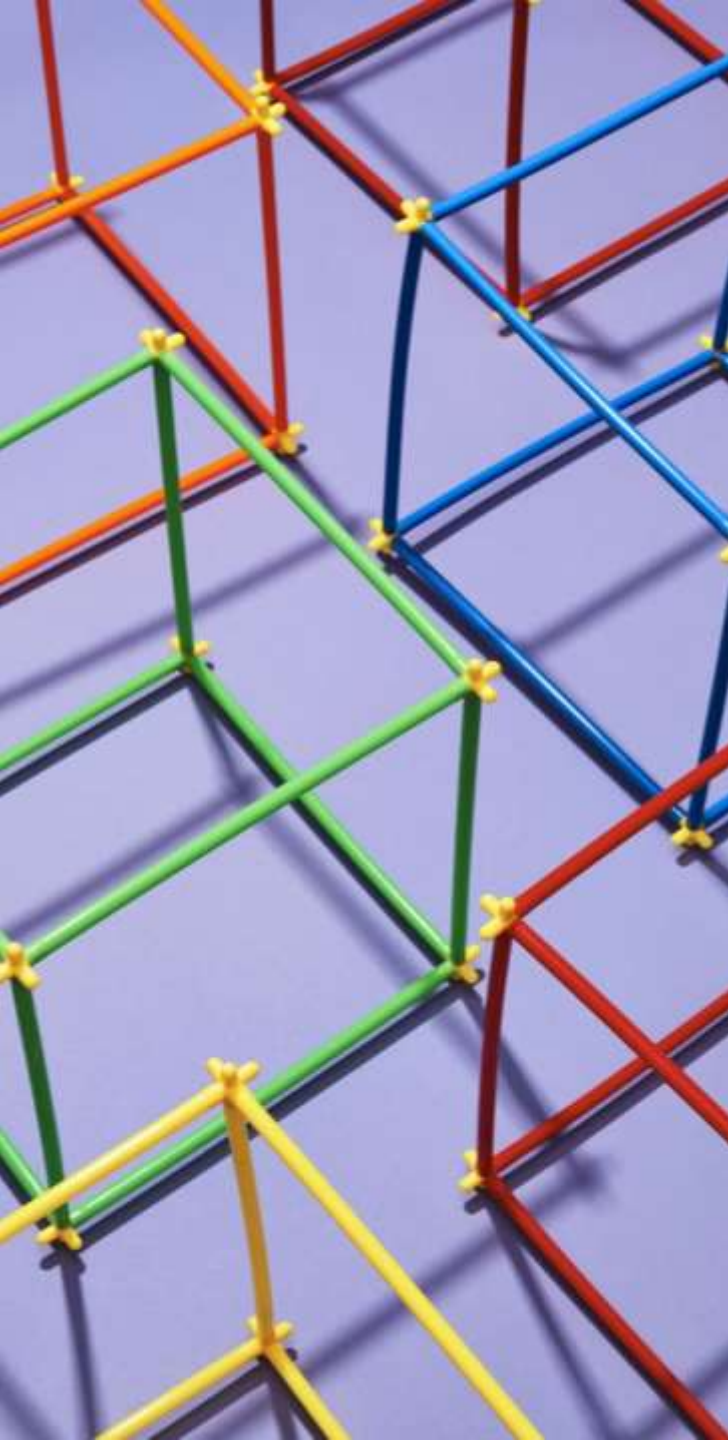
Use rules to help propagate changes:

- Recall as high as 44%
- Precision around 30%

High precision and recall reached in < 1mth

Prediction accuracy improves prior to a release (i.e., during maintenance phase)

[Zimmermann et al. 05]

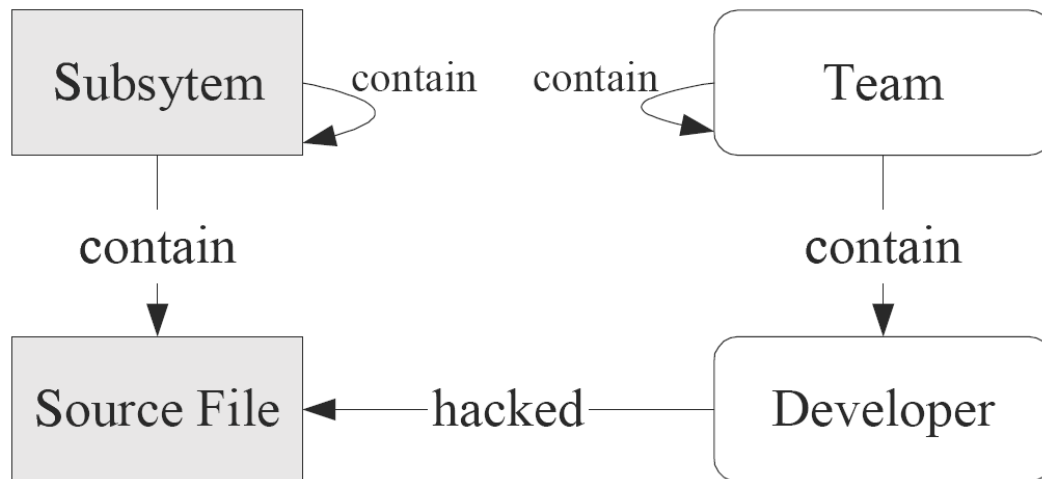


Code Sticky Notes

- **Traditional dependency graphs and program understanding models usually do not use historical information**
- **Static dependencies capture only a static view of a system – not enough detail!**
- **Development history can help understand the current structure (architecture) of a software system**

[Hassan & Holt 04]

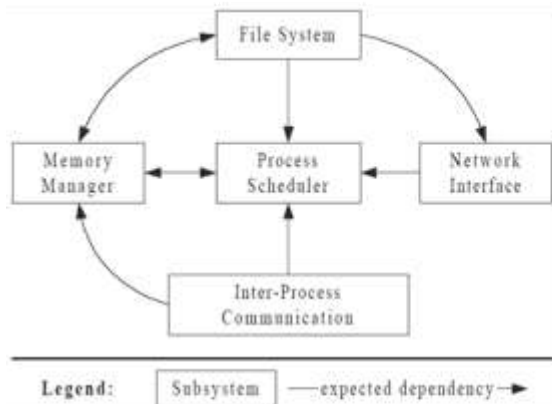
Studying Conway's Law



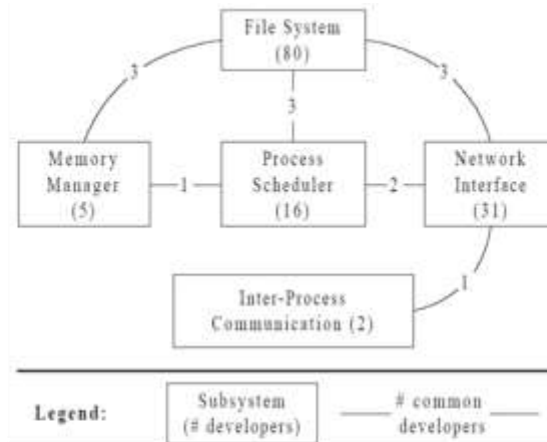
- Conway's Law:
"The structure of a software system is a direct reflection of the structure of the development team"

[Bowman et al. 99]

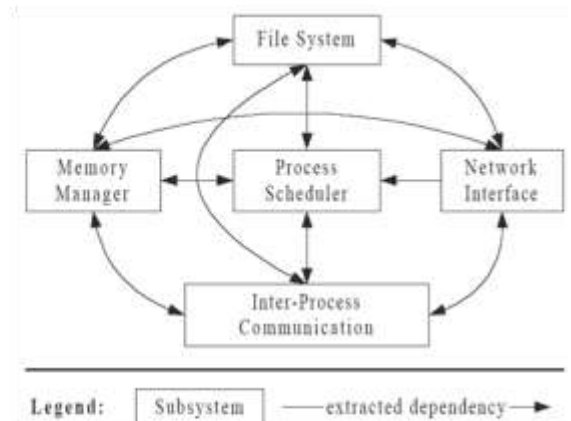
Linux: Conceptual, Ownership, Concrete



**Conceptual
Architecture**

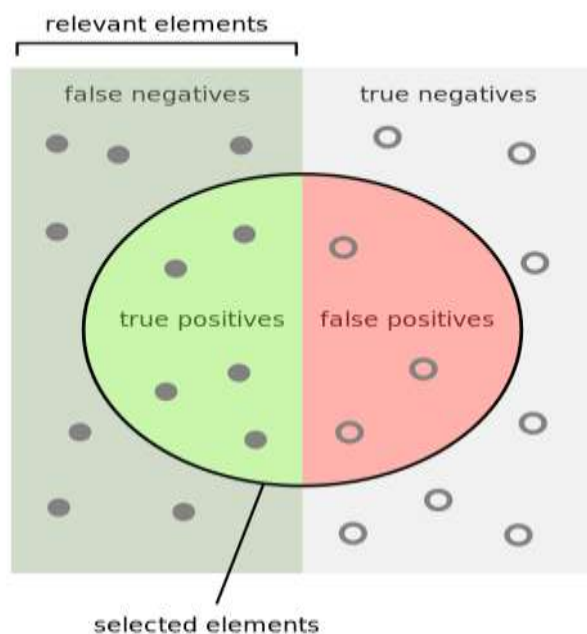


**Ownership
Architecture**



**Concrete
Architecture**

Measuring Quality of Mining approach



How many selected items are relevant?

$$\text{Precision} = \frac{\text{true positives}}{\text{true positives} + \text{false positives}}$$

How many relevant items are selected?

$$\text{Recall} = \frac{\text{true positives}}{\text{true positives} + \text{false negatives}}$$

$$\text{Recall} = \frac{tp}{tp + fn}$$

$$\text{Precision} = \frac{tp}{tp + fp}$$

- We want:
 - High Precision to avoid wasting time
 - High Recall to avoid bugs

Source Control and Bug Repositories

Predicting Bugs

Studies have shown that most complexity metrics correlate well with LOC!

- Graves et al. 2000 on commercial systems
- Herraiz et al. 2007 on open source systems

Noteworthy findings:

- Previous bugs are good **predictors** of future bugs
- The **more a file changes**, the **more likely** it will have bugs in it
- Recent changes affect more the bug potential of a file over older changes (*weighted time damp models*)
- Number of developers is of little help in predicting bugs
- Hard to **generalize bug predictors** across projects unless in similar domains [Nagappan, Ball et al. 2006]

Using Imports in Eclipse to Predict Bugs

71% of files that import compiler packages,
had to be fixed later on.

```
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 files that import ui packages, had
to be fixed later on.

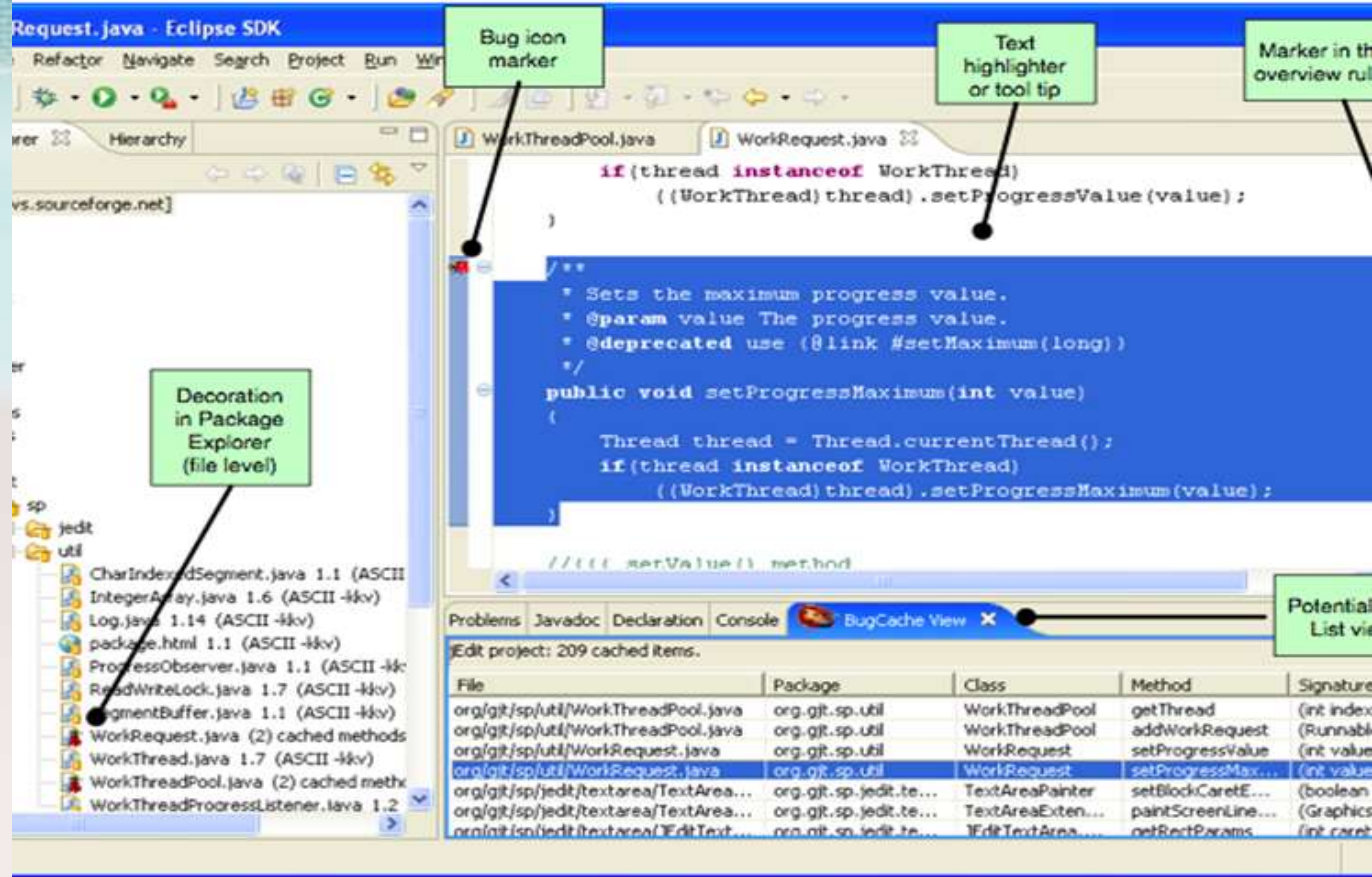
[Schröter et al. 06]

Don't program on Fridays ;-)




Percentage of bug-introducing changes for eclipse

[Zimmermann et al. 05]



- Given a change can we warn a developer that there is a bug in it?
 - Recall/Precision in 50-60% range



A decorative graphic on the right side of the slide consists of several overlapping hexagons. The largest hexagon at the top left contains a close-up, slightly blurred image of various colored paper scraps (yellow, orange, green, blue) and a thin green stick. To its right and below are several smaller, semi-transparent white hexagons. At the bottom center, another hexagon shows a close-up of a purple paper scrap on a yellow background.

Project Communication – Mailing lists

Project Communication (Mailinglists)

Most open source projects communicate through mailing lists or IRC channels

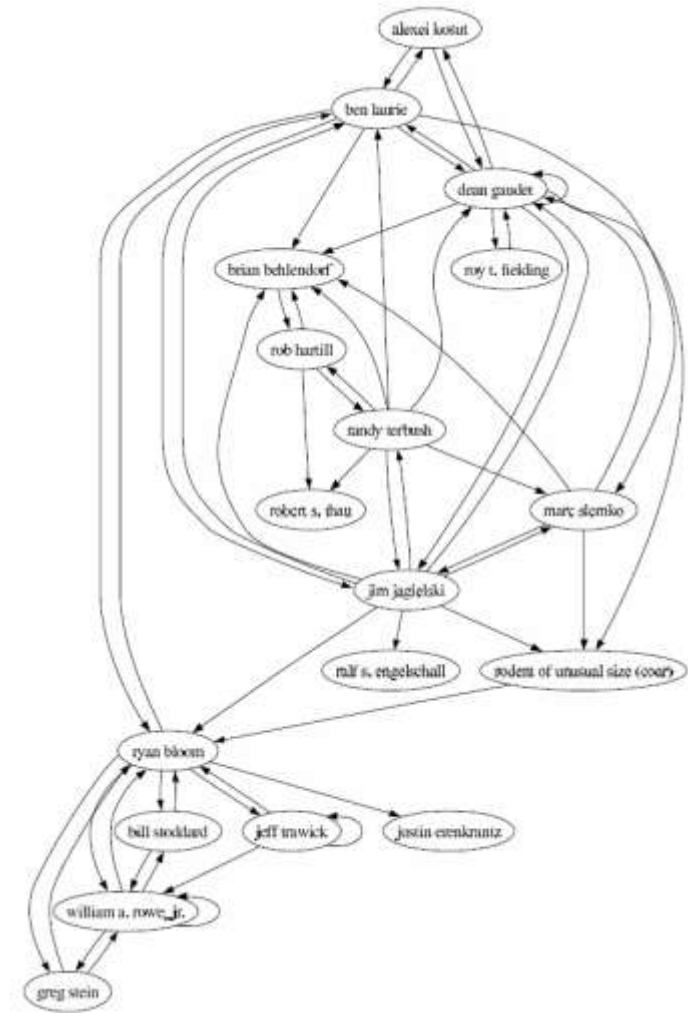
Rich source of information about the inner workings of large projects

Discussions cover topics such as future plans, design decisions, project policies, code or patch reviews

Social network analysis could be performed on discussion threads

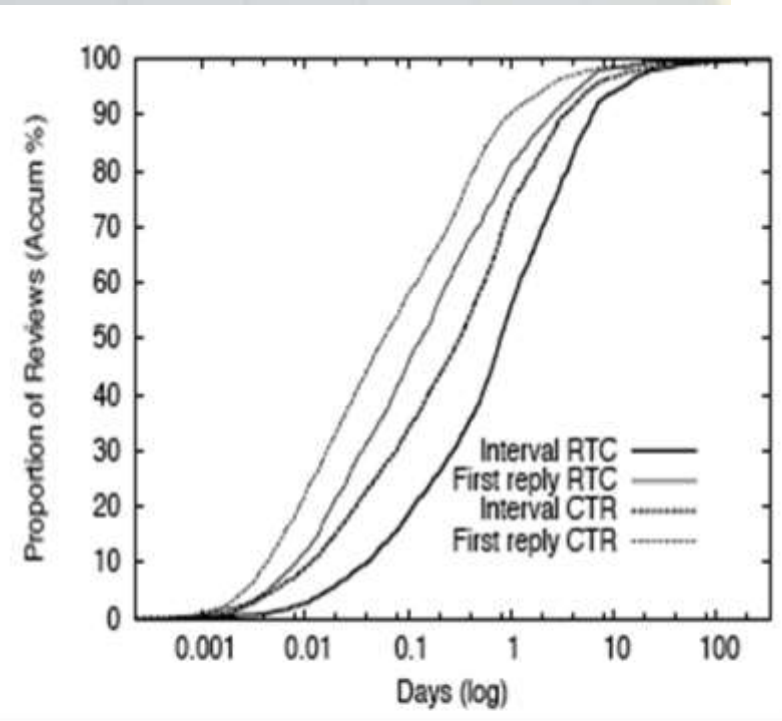
Social Network Analysis

- Mailing list activity:
 - strongly correlates with code change activity
 - moderately correlates with document change activity
- Social network measures (in-degree, out-degree, betweenness) indicate that committers play a more significant role in the mailing list community than non-committers



[Bird et al. 06]

The Patch Review Process



- Two review styles
 - RTC: Review-then-commit
 - CTR: Commit-then-review
- 80% patches reviewed within 3.5 days and 50% reviewed in <19 hrs

Measure a team's morale around release time?

- Study the content of messages before and after a release
- Use dimensions from a psychometric text analysis tool:
 - After Apache 1.3 release there was a drop in optimism
 - After Apache 2.0 release there was an increase in sociability

Dimension	-1.3	-2.0
Optimism	-0.37	*
Tentative	-1.3	*
References to Time	1.1	*
Future tense verbs	-0.7	*
Social Processes	*	0.74
Inclusive	*	-0.64

Table 4. Mean differences for Apache 1.3 and 2.0 releases. (* $p > 0.05$, otherwise $p \leq 0.05$)

Program
Source
Code

Code Entities

Source data	Mined info
Variable names and function names	Software categories [Kawaguchi et al. 04]
Statement seq in a basic block	Copy-paste code [Li et al. 04]
Set of functions, variables, and data types within a C function	Programming rules [Li&Zhou 05]
Sequence of methods within a Java method	API usages [Xie&Pei 05]
API method signatures	API Jungloids [Mandelin et al. 05]

Program Execution Traces

Method-Entry/Exit States

- Goal: mine specifications (pre/post conditions) or object behavior (object transition diagrams)
- State of an object
 - Values of transitively reachable fields
- Method-entry state
 - Receiver-object state, method argument values
- Method-exit state
 - Receiver-object state, updated method argument values, method return value

[Ernst et al. 02] <http://pag.csail.mit.edu/daikon/>

[Xie&Notkin 04/05][Dallmeier et al. 06] <http://www.st.cs.uni-sb.de/models/>