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

More information available at
http://ase.csc.ncsu.edu/dmse/

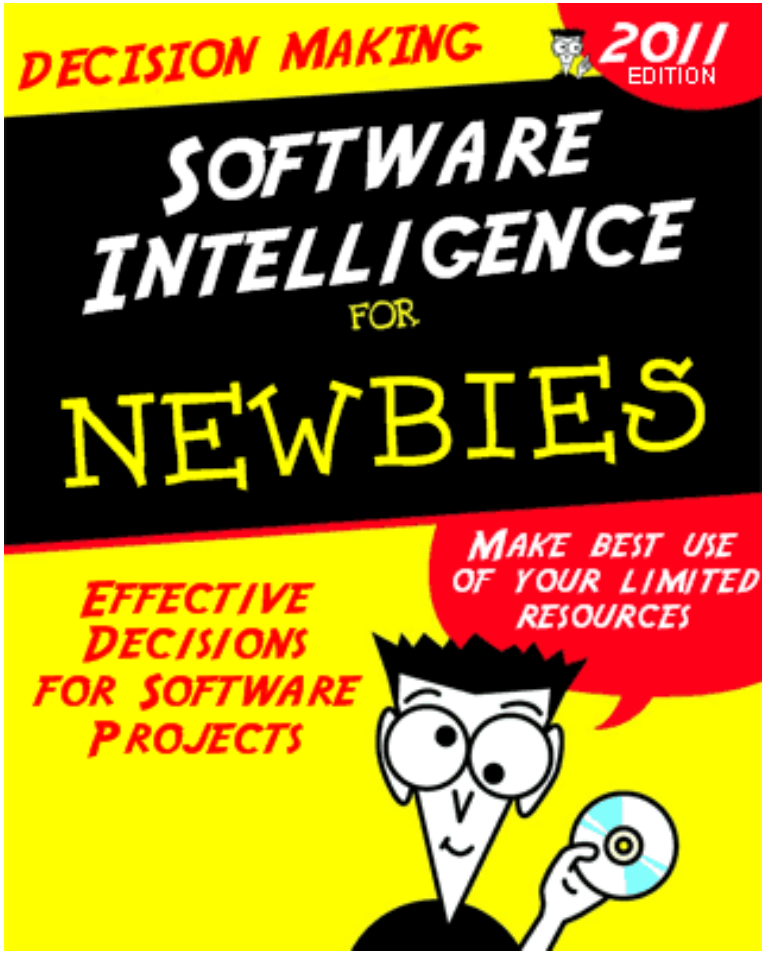
Ahmed E. Hassan

- NSERC/RIM Software Engineering Research Chair Queen's University, Canada
- Leads the SAIL research group at Queen's
- Co-chair for Workshop on Mining Software Repositories (MSR) from 2004-2006
- Chair of the steering committee for MSR

Tao Xie

- Associate Professor at North Carolina State University, USA
- Leads the ASE research group at NCSU
- PC Co-Chair of ICSM 2009, MSR 2011/2012
- Co-organizer of 2007 Dagstuhl Seminar on Mining Programs and Processes

Automated Software Engineering Research Group @NCSU


DECISION MAKING
2011
EDITION
SOFTWARE
INTELLIGENCE
FOR
NEWBIES
MAKE BEST USE
OF YOUR LIMITED
RESOURCES
EFFECTIVE
DECISIONS
FOR SOFTWARE
PROJECTS

References

The Road Ahead for Mining Software Repositories

Ahmed E. Hassan
Software Analysis and Intelligence Lab (SAIL)
School of Computing, Queen's University, Canada
ahmed@cs.queensu.ca

Abstract

Source control repositories, bug repositories, archived communications, deployment logs, and code repositories are examples of software repositories that are commonly available for most software projects. The Mining Software Repositories (MSR) field analyzes and cross-links the rich data available in these repositories to uncover interesting and actionable information about software systems. By transforming these repositories from static record-keeping ones into active repositories, we can guide decision processes in modern software projects. For example, data in source control repositories, traditionally used to archive code, could be linked with data in bug repositories to help practitioners propagate complex changes and to warn them about risky code based on prior changes and bugs. In this paper, we present a brief history of the MSR field and discuss several recent achievements and results of using MSR techniques to support software research and practice. We then discuss the various opportunities and challenges that lie in the road ahead for this important and emerging field.

1 Introduction

Prior experiences and dominant patterns are the driving force for many decision-processes in modern software organizations. Practitioners often rely on their experience, intuition and gut feeling in making important decisions. Managers allocate development and testing resources based on their experience in previous projects and their intuition about the complexity of the new project relative to prior projects. Developers commonly use their experience when adding a new feature or fixing a bug. Testers usually prioritize the testing of features that are known to be error prone based on field and bug reports. Software repositories contain a wealth of valuable information about software projects. Using the information stored in these repositories, practitioners can depend less on their intuition and experience, and depend more on historical and field data.

The Mining Software Repositories (MSR) field analyzes and cross-links the rich data available in software repositories to uncover interesting and actionable information about software systems and projects. Examples of software repositories are:

Historical repositories such as source control repositories, bug repositories, and archived communications record several information about the evolution and progress of a project.

Run-time repositories such as deployment logs contain information about the execution and the usage of an application at a single or multiple deployment sites.

Code repositories such as Sourceforge.net and Google code contain the source code of various applications developed by several developers.

Software repositories are commonly used in practice as record-keeping repositories and are rarely used to support decision making processes. For example, historical repositories are used to track the history of a bug or a feature, but are not commonly used to determine the expected resolution time of an open bug based on the resolution time of previously-closed bugs.

MSR researchers aim to transform these repositories from static record-keeping ones into active repositories which can guide decision processes in modern software projects. Mining these historical, run-time and code repositories, we can uncover useful and important patterns and information. Historical repositories capture important historical dependencies [20] between project artifacts, such as functions, documentation files, or configuration files. Developers can use this information to propagate changes to related artifacts, instead of only using static or dynamic code dependencies which may fail to capture important dependencies. For example, a change to the code which writes data to a file may require changes to the code which reads data from the file, although there exists no traditional dependencies (e.g., data and control flow) between both pieces of

Software Intelligence: The Future of Mining Software Engineering Data

Ahmed E. Hassan
School of Computing
Queen's University
Kingston, ON, Canada
ahmed@cs.queensu.ca

Tao Xie
Department of Computer Science
North Carolina State University
Raleigh, NC, USA
xie@csc.ncsu.edu

ABSTRACT

Mining software engineering data has emerged as a successful research direction over the past decade. In this position paper, we advocate Software Intelligence (SI) as the future of mining software engineering data, within modern software engineering research, practice, and education. We coin the name SI as an inspiration from the Business Intelligence (BI) field, which offers concepts and techniques to improve business decision making by using fact-based support systems. Similarly, SI offers software practitioners (not just developers) up-to-date and pertinent information to support their daily decision-making processes. SI should support decision-making processes throughout the lifetime of a software system not just during its development phase.

The vision of SI has yet to become a reality that would enable software engineering research to have a strong impact on modern software practice. Nevertheless, recent advances in the Mining Software Repositories (MSR) field show great promise and provide strong support for realizing SI in the near future. This position paper summarizes the state of practice and research of SI, and lays out future research directions for mining software engineering data to enable SI.

Categories and Subject Descriptors

D.2.7 [**Software Engineering**]: Distribution, Maintenance, and Enhancement

General Terms

Documentation, Economics, Experimentation, Human Factors, Management, Measurement, Reliability, Verification

Keywords

Software intelligence, mining software engineering data, mining software repositories

FoSER 2010, November 7–8, 2010, Santa Fe, New Mexico, USA.

1. INTRODUCTION

Much of software practice centers around daily decisions and questions (e.g., when to release a software system? which parts of a software system to change? which parts of a software system to test? who is using this feature? and who knows about this feature?). Unfortunately, nowadays many decisions related to a software system are based on intuition and gut feeling. Determining when a software system is ready for release, whether a part of a software system should be re-factored or re-written, or which parts of a software system should be thoroughly tested is a matter of art instead of a well-studied science. These haphazard decision processes lead to wasted resources and increased cost of building and maintaining large complex software systems.

Software practitioners are in dire need of what we propose to call Software Intelligence (SI). While Business Intelligence (BI) [27] offers concepts and techniques to improve business decision making by using fact-based support systems, SI offers software practitioners (not just developers) up-to-date and pertinent information to support their daily decision-making processes. SI provides practitioners with access to specialized fact-supported views of their software system so they can answer critical questions about it. Using SI, owners, maintainers, and developers of software systems can perform long-term and short-term informed strategic planning. Moreover, SI gives companies a better understanding of the true potential and actual limitation of their software assets.

Mining software engineering data has emerged as a research direction over the past decade. This research direction has already achieved substantial success in both research and practice. In this position paper, we advocate Software Intelligence (SI) as the future of mining software engineering data, within modern software engineering research, practice, and education.

The vision of SI has yet to become a reality. Nevertheless, recent advances in the Mining Software Repositories (MSR) field show great promise and provide strong support for realizing SI in the near future, as software engineering research aims to ensure its relevance and impact on modern software practice. This position paper summarizes state of practice and research of SI, and lays out future research directions of mining software engineering data to enable SI.

2. STATE OF PRACTICE

Prior experiences and dominant patterns are the driving force for many decision-making processes in modern software organizations. Software practitioners often rely on their experience, intuition, and gut feeling in making important decisions. Managers allocate development and testing resources based on their experience in previous projects and their intuition about the complexity of the new project relative to prior projects. Developers commonly

Should I test\review my?

A. Ten *most-complex* functions

B. Ten *largest* functions

C. Ten *most-fixed* functions

Business Intelligence

Supporting decision making using facts instead of fortune tellers!

Software Intelligence

Business Intelligence for Software

Look through your software data

Mine through the data!

An international effort to make software repositories actionable

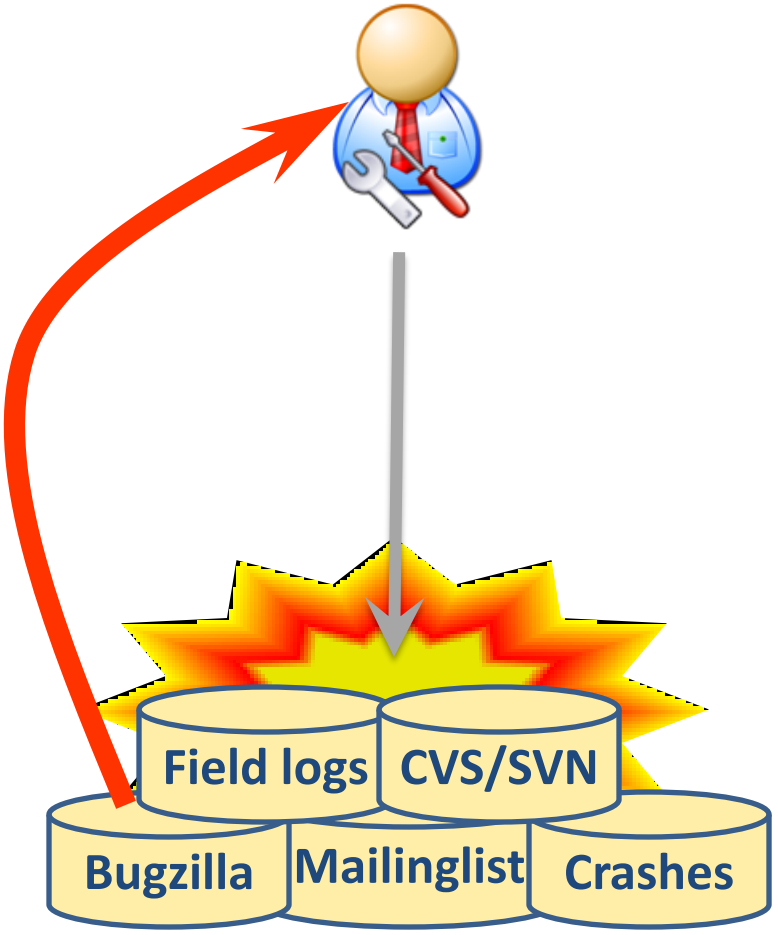
PROMISE 2011

http://msrconf.org

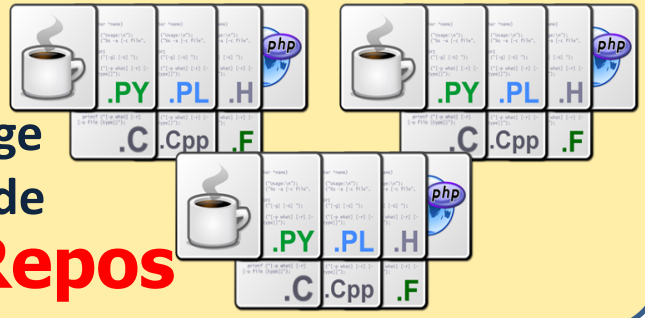
http://promisedata.org

Mining Software Repositories (MSR)

- Transforms static record-keeping repositories to **active** repositories
- Makes repository data ***actionable*** by uncovering hidden **patterns** and **trends**



Sourceforge
GoogleCode
Code Repos



Source Control
CVS/SVN

Bugzilla

Mailing lists

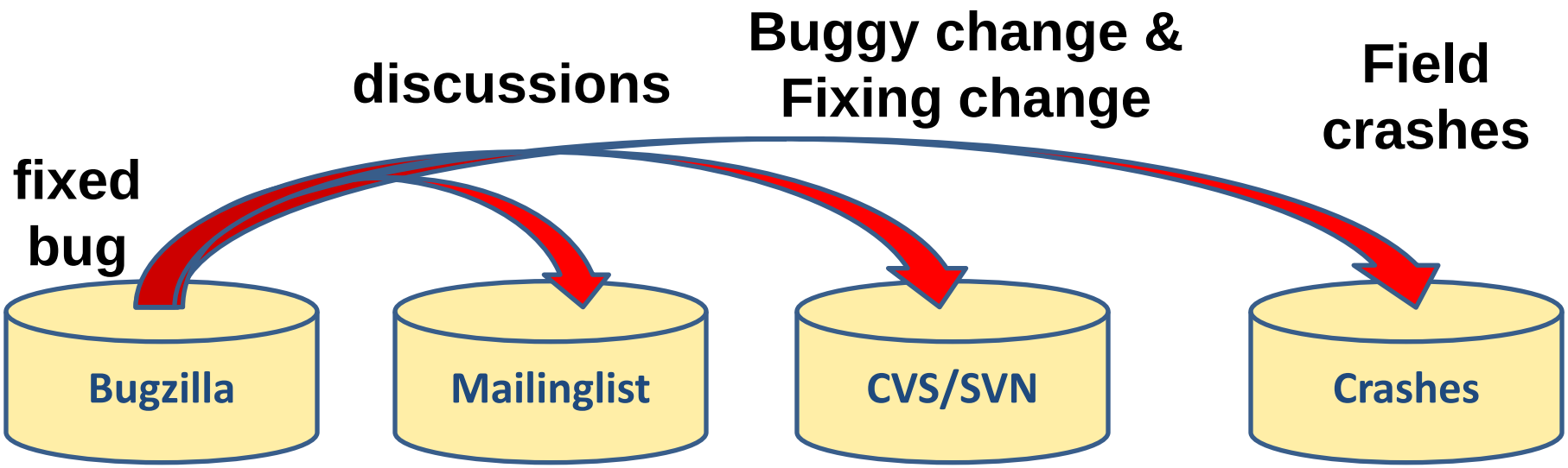
Historical Repositories

Crash Repos

Field Logs

Runtime Repos

MSR researchers analyze and cross-link repositories



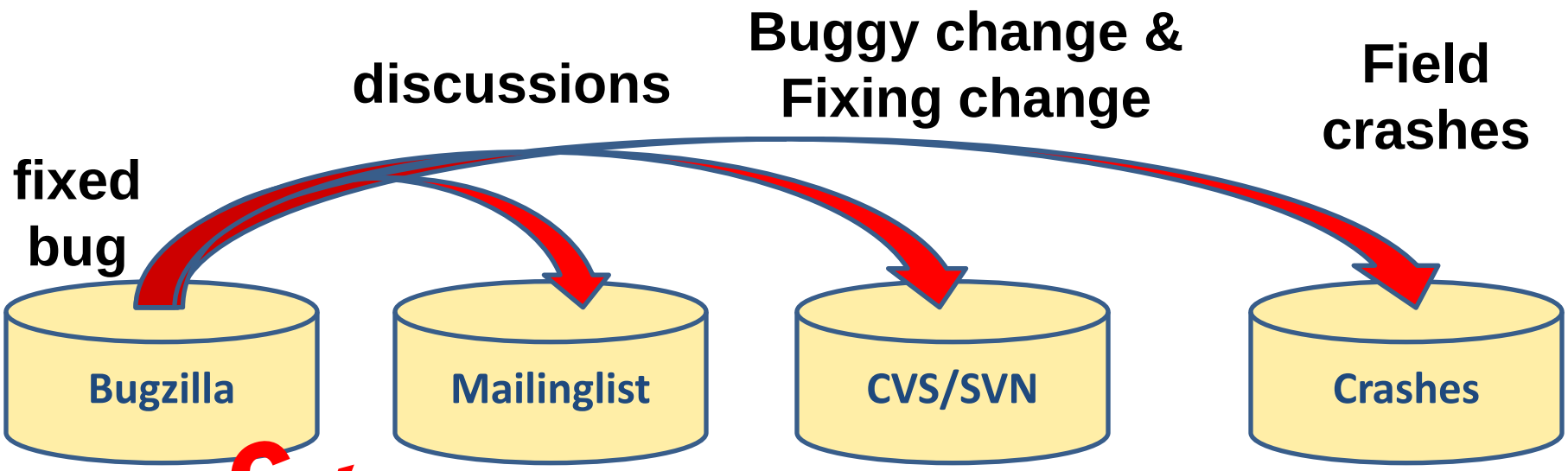
New Bug Report

Estimate fix effort

Mark duplicates

Suggest experts and fix!

MSR researchers analyze and cross-link repositories



3-6x over using only static dependencies (Recall 78%, Precision 64%)

New Change

Suggest APIs

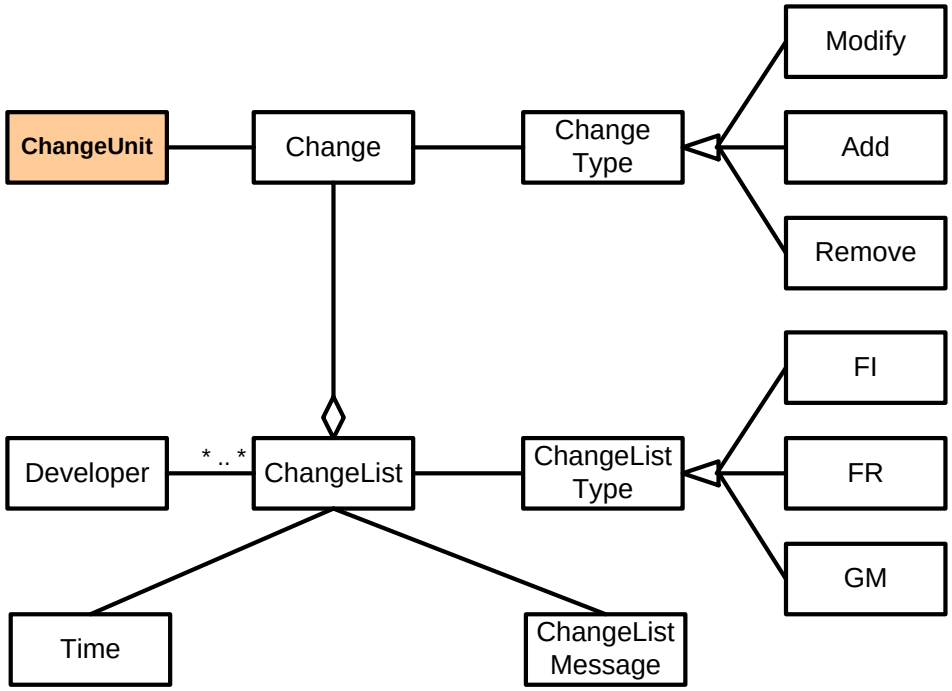
Warn about risky code or bugs

Suggest locations to co-change

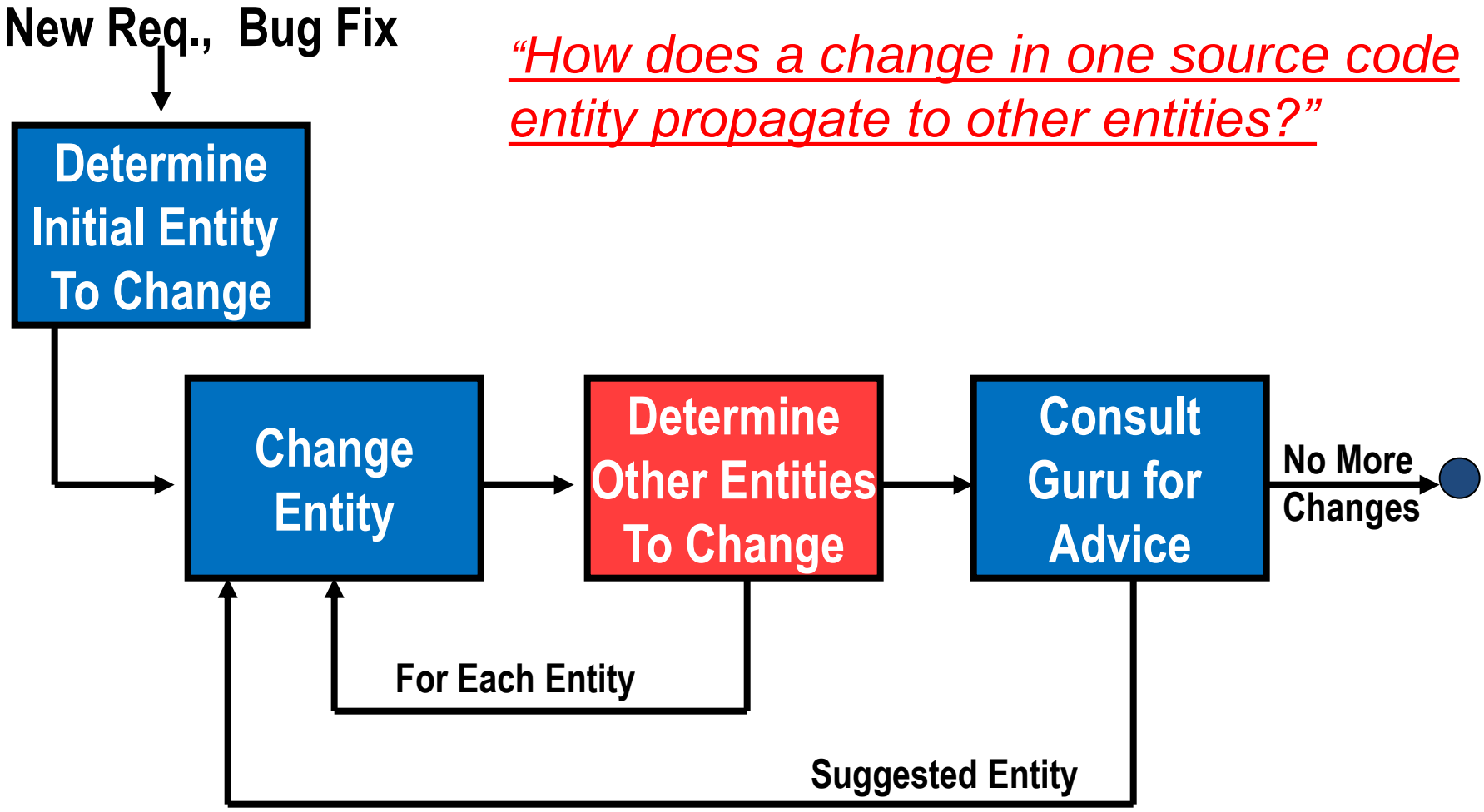
Example Repositories:
Source Control and Bug Repositories

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



"How does a change in one source code entity propagate to other entities?"

Measuring Change Propagation

$$\text{Precision} = \frac{\text{predicted entities which changed}}{\text{predicted entities}}$$

$$\text{Recall} = \frac{\text{predicted entities which changed}}{\text{changed entities}}$$

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

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)
- Better predictor than static dependencies alone

[Zimmermann et al. 05]

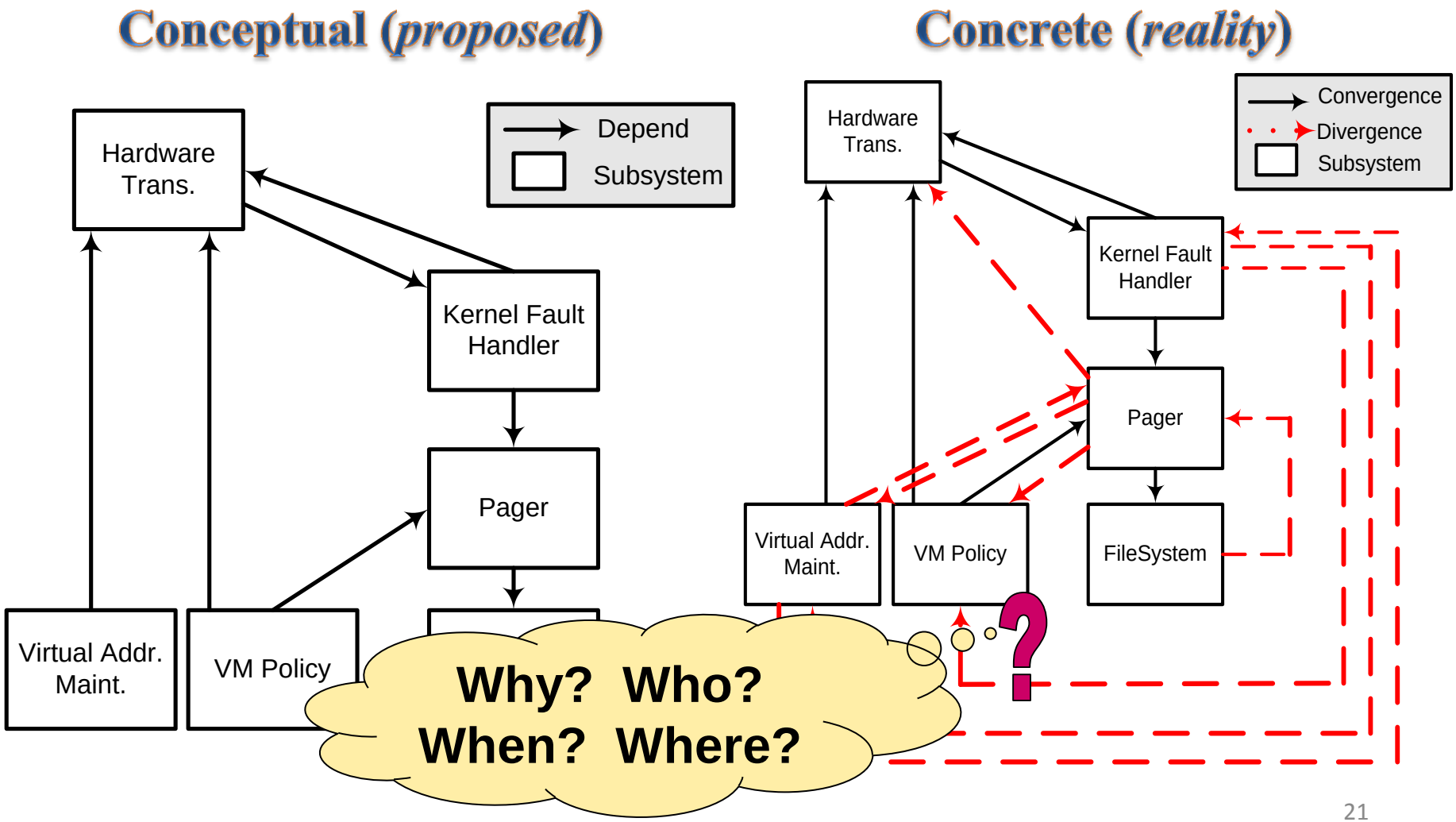
[Hassan&Holt 04]

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]

Supporting software understanding (NETBSD)



Mining supports software understanding (NETBSD)

- Eight unexpected dependencies
- All except two dependencies *existed since day one*:
  - Virtual Address Maintenance → Pager
  - Pager → Hardware Translations

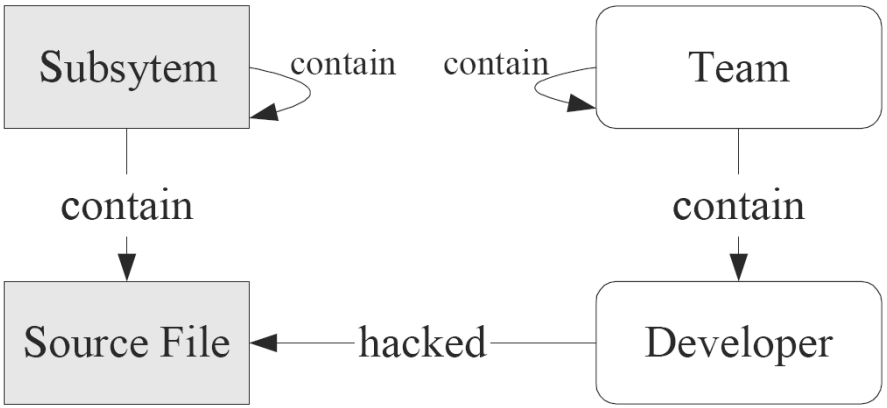
Auto-generated from CVS repository

Which?	vm_map_entry_create (in src/sys/vm/Attic/vm_map.c) *depends on* pager_map (in /src/sys/uvm/uvm_pager.c)
Who?	cgd
When?	1993/04/09 15:54:59 Revision 1.2 of src/sys/vm/Attic/vm_map.c
Why?	from sean eric fagan: it seems to keep the vm system from deadlocking the system when it runs out of swap + physical memory. prevents the system from giving the last page(s) to anything but the referenced "processes" (especially important is the pager process, which should never have to wait for a free page).

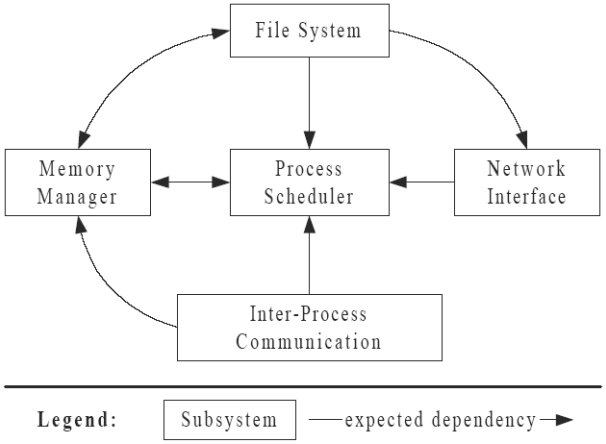
Social Structure Influences Software

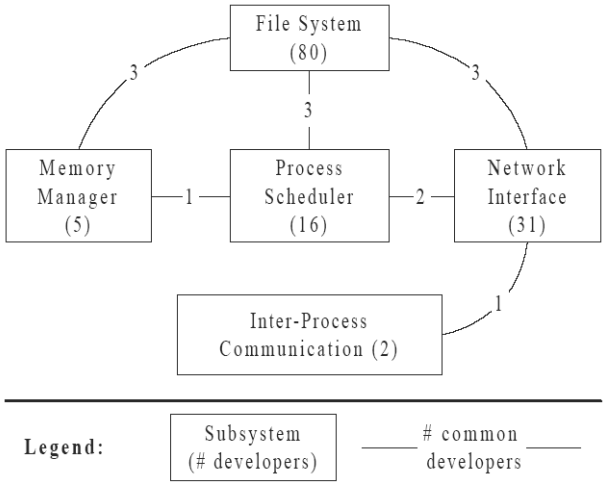
- Conway's Law:

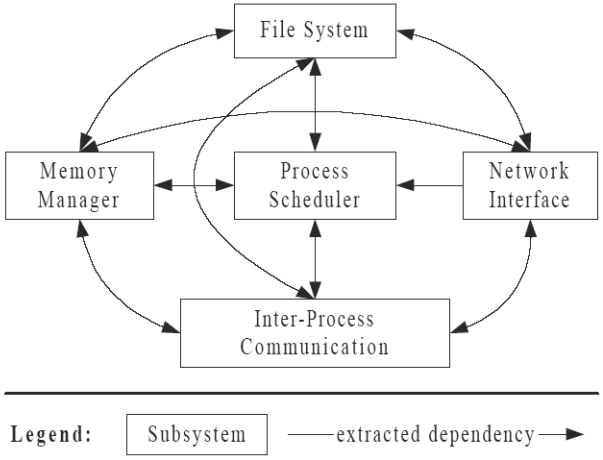
  "The structure of a software system is a direct reflection of the structure of the development team"



Linux: Conceptual, Ownership, Concrete







Conceptual Architecture

Ownership Architecture

Concrete Architecture

Using Imports in Eclipse to Predict Bugs

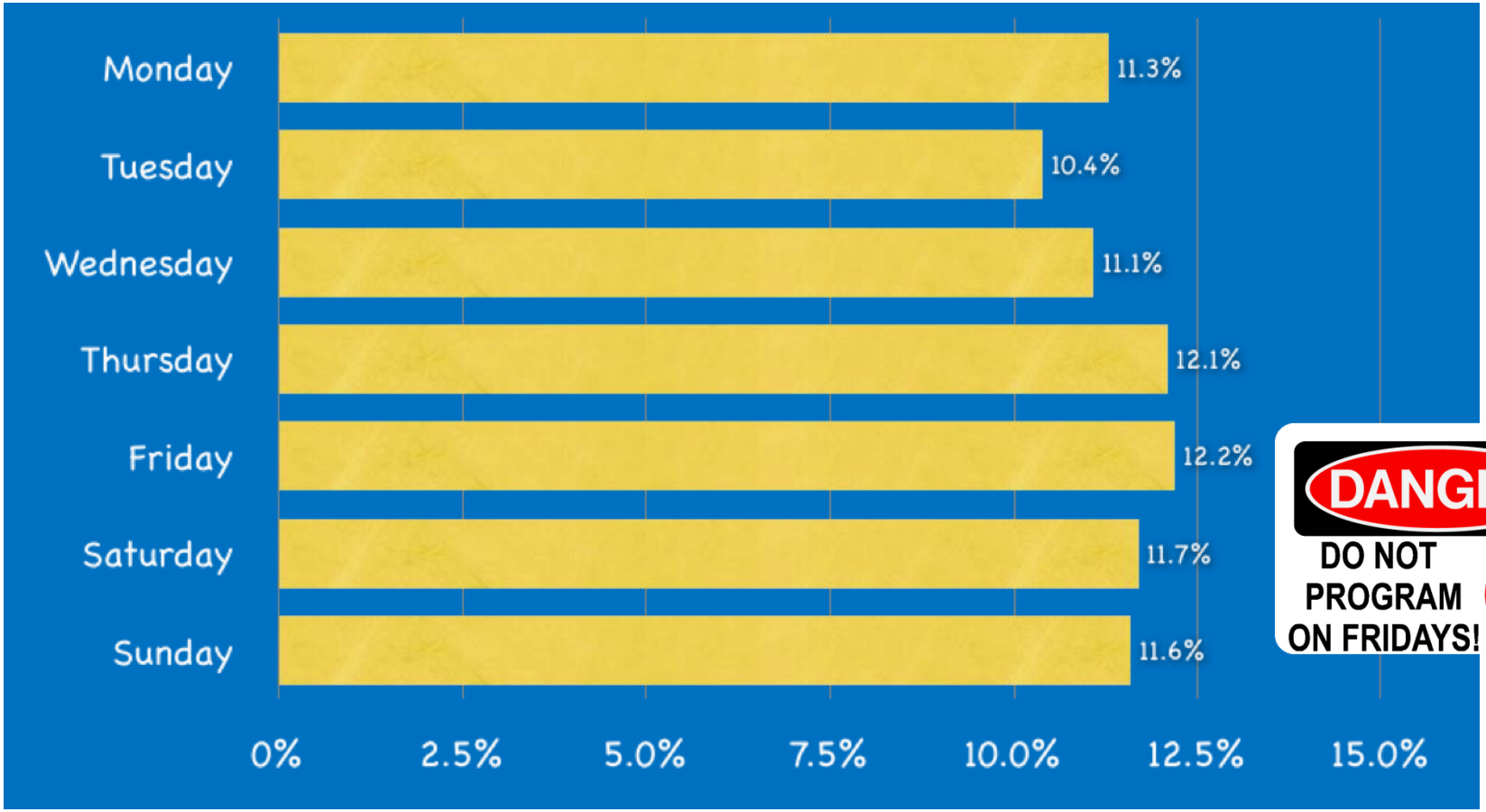
71% of files that import compiler packages, had to be fixed later on.

```
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]

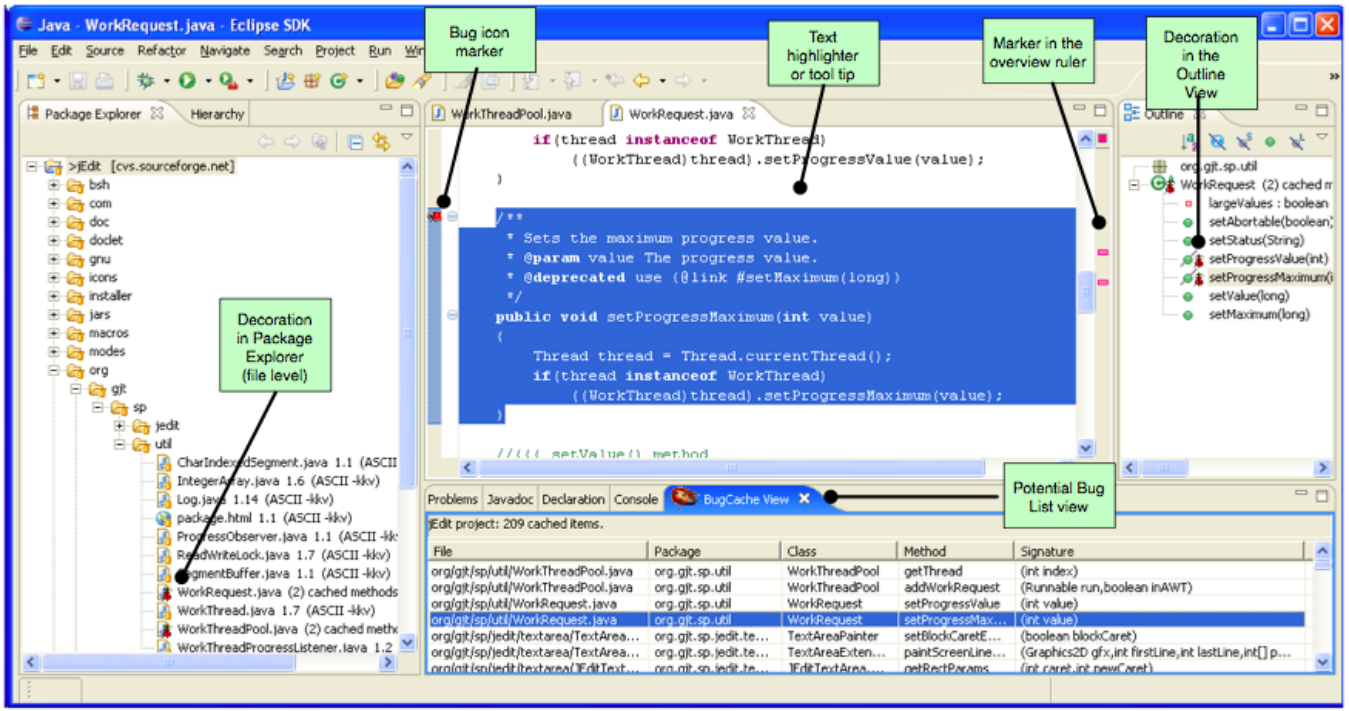
Don’t program on Fridays ;-)



Percentage of bug-introducing changes for eclipse [Zimmermann et al. 05]

Classifying Changes as Buggy or Clean

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



[Kim et al. 06]

Example Repositories:
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

Measure a team's morale around release time?

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

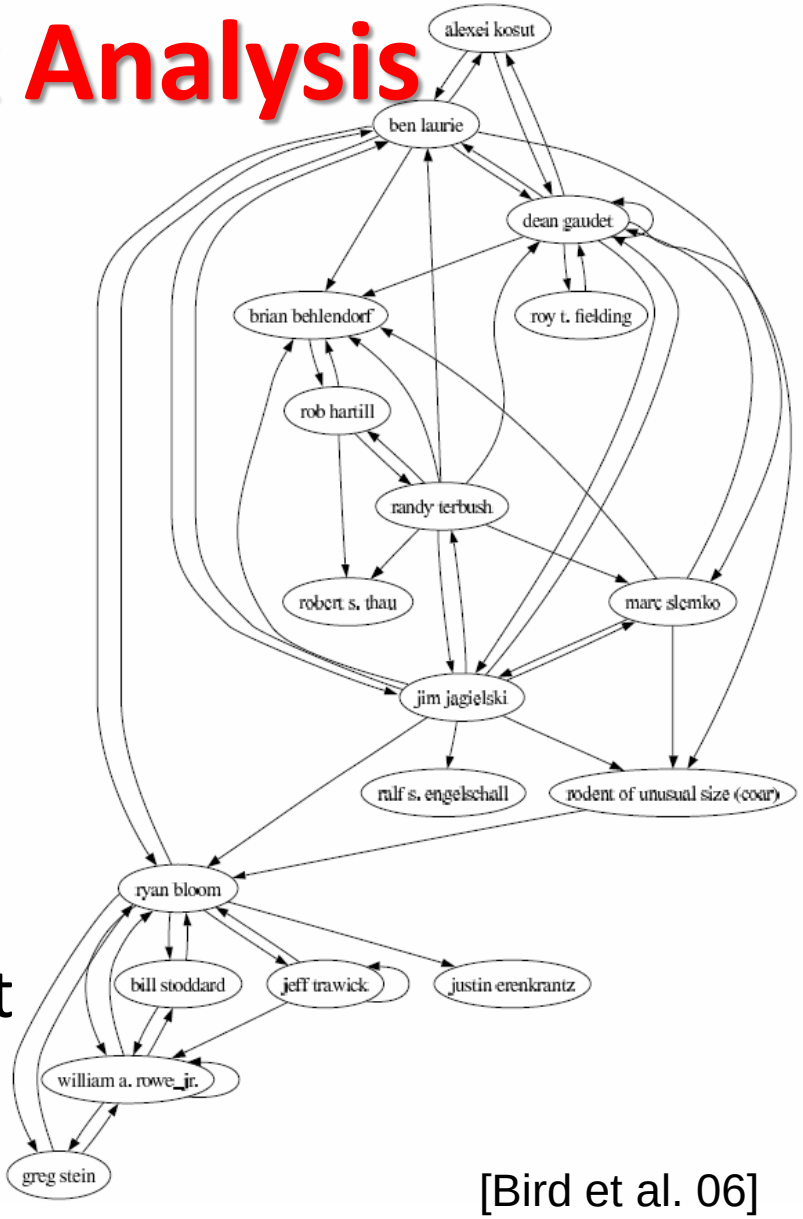
Table 4. Mean differences for Apache 1.3 and 2.0 releases. (* $p > 0.05$, otherwise $p \leq 0.05$)

- 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

[Rigby & Hassan 07]

Social Network Analysis

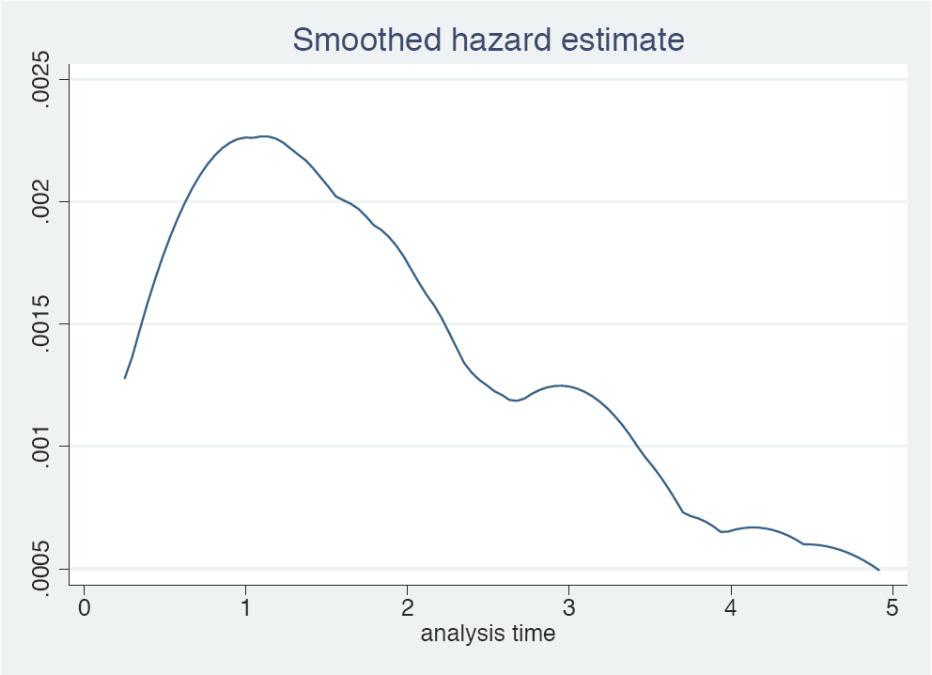
- Mailing list activity:
  - strongly correlates with code change activity
  - moderately correlates with document change activity
- Social network measures (in-degree, out-degree, betweenness) indicate that committers play a more significant role in the mailing list community than non-committers



[Bird et al. 06]

Immigration Rate of Developers

- When will a developer be invited to join a project?
  - Expertise vs. interest



[Bird et al. 07]

Example Repositories:
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 06]
API method signatures	API Jungloids [Mandelin et al. 05]

Mining API Usage Patterns

- How should an API be used correctly?
  - An API may serve multiple functionalities
  - Different styles of API usage
- "I know what type of object I need, but I don't know how to write the code to get the object" [Mandelin et al. 05]
  - Can we synthesize jungloid code fragments automatically?
  - Given a simple query describing the desired code in terms of input and output types, return a code segment
- "I know what method call I need, but I don't know how to write code before and after this method call" [Xie&Pei 06]

Relationships between Code Entities

- Mine framework reuse patterns [Michail 00]
  - Membership relationships
    - A class contains membership functions
  - Reuse relationships
    - Class inheritance/ instantiation
    - Function invocations/overriding
- Mine software plagiarism [Liu et al. 06]
  - Program dependence graphs

[Michail 99/00] http://codeweb.sourceforge.net/ for C++

Detect Copy-Paste Code

- Apply closed sequential pattern mining techniques
- Customizing the techniques
  - A copy-paste segment typically does not have big gaps – use a maximum gap threshold to control
  - Output the instances of patterns (i.e., the copy-pasted code segments) instead of the patterns
  - Use small copy-pasted segments to form larger ones
  - Prune false positives: tiny segments, unmappable segments, overlapping segments, and segments with large gaps

[Li et al. 04]

Find Bugs in Copy-Pasted Segments

- For two copy-pasted segments, are the modifications consistent?
  - Identifier *a* in segment S1 is changed to *b* in segment S2 3 times, but remains unchanged once – *likely a bug*
  - The heuristic may not be correct all the time
- The lower the unchanged rate of an identifier, the more likely there is a bug

[Li et al. 04]

Example Repositories:
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/

Other Profiled Program States

- Goal: detect or locate bugs
- Values of variables at certain code locations [Hangal&Lam 02]
  - Object/static field read/write
  - Method-call arguments
  - Method returns
- Sampled predicates on values of variables [Liblit et al. 03/05][Liu et al. 05]

[Hangal&Lam 02] http://diduce.sourceforge.net/
[Liblit et al. 03/05] http://www.cs.wisc.edu/cbi/
[Liu et al. 05] http://www.ews.uiuc.edu/~chaoliu/sober.htm

Executed Structural Entities

- Goal: locate bugs
- Executed branches/paths, def-use pairs
- Executed function/method calls
  - Group methods invoked on the same object
- Profiling options
  - Execution hit vs. count
  - Execution order (sequences)

[Dallmeier et al. 05] http://www.st.cs.uni-sb.de/ample/
More related tools: http://www.csc.ncsu.edu/faculty/xie/research.htm#related

GUI-Application Stabilizer

- Given a program state ***S*** and an event ***e***, predict whether ***e*** likely results in a bug
  - Positive samples: past bugs
  - Negative samples: “not bug” reports
- A k-NN based approach
  - Consider the k closest cases reported before
  - Compare Σ 1/d for bug cases and not-bug cases, where d is the similarity between the current state and the reported states
  - If the current state is more similar to bugs, predict a bug

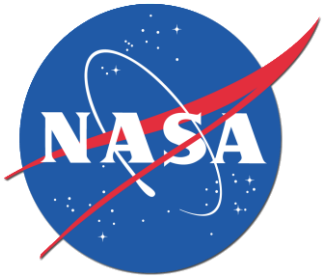
[Michail&Xie 05]

Mining Emerging Patterns in Traces

- A method executed only in failing runs is likely to point to the defect
  - Comparing the coverage of passing and failing program runs helps
- Mining patterns frequent in failing program runs but infrequent in passing program runs
  - Sequential patterns may be used

[Dallmeier et al. 05, Denmat et al. 05]

MSR in Practice



Adoption Challenges for Software Intelligence

Must show value before data quality improves

Correlation vs. Causation

Integration into daily pratice

Mining Software Repositories

- Very active research area in SE:
  - MSR is the most attended ICSE event in last 8 yrs
    - http://msrconf.org

  - Special Issue of IEEE TSE 2005 on MSR:
    - 15 % of all submissions of TSE in 2004
    - Fastest review cycle in TSE history: 8 months
  - Special Issue Empirical Software Engineering 2009, 2011
  - MSR 2012!

Publishing Advice

- Report the statistical significance of your results:
  - Get a statistics book (one for social scientist, not for mathematicians)
- Discuss any limitations of your findings based on the characteristics of the studied repositories:
  - Make sure you manually examine the repositories. Do not fully automate the process!
  - Use random sampling to resolve issues about data noise
- Relevant conferences/workshops:
  - main SE conferences, ICSM, ISSTA, MSR, WODA, PROMISE …