Empirical Studies on Software Evolution: Should we (try to) claim Causation?

Massimiliano Di Penta

RCOST - Dept. Of Engineering
University of Sannio, Benevento (Italy)
dipenta@unisannio.it
Where does this guy come from??

- FAQ when people read my badge at conferences
- Sannio is a geographical area in Southern Italy between Naples and Rome
Pictures

Place where crazy people work
(A long time ago in a galaxy far, far away)

Roman’s theatre
(126 d.c.)

Traiano Arch
(114 A.C.)

S. Sophia Church
(VIII cent. A.C.)
13 CS professors (among full, associate, assistant)

4 PhD students, 6 post-docs

Working on topics such as parallel computing, model checking, SOA, software maintenance and evolution, software testing, empirical software engineering
Back to work issues...

- Most of my research is about software evolution
  - Strong empirical content

- Massive usage of data from software repositories

- Sometimes performing controlled experiments involving human subjects
Strong Empirical Nature of Software Evolution Studies

- Models and theories based on empirical observations
- Tools empirical evaluated on past project data

IWPSE 2005
Case studies, 4
Other papers, 16

IWPSE 2007
Case studies, 4
Experiments, 1
Survey, 1
Other papers, 14

IWPSE 2009
Case studies, 6
Experiments, 2
Other papers, 10
Kinds of studies

- **Survey**: Retrospective (post mortem), e.g. about a technology/tool being adopted for a period of time

- **Case study**: monitoring an ongoing (real) project or a study on a post mortem project

- **Experiment**: performed in a laboratory setting, with a high degree of control
  - Objective: manipulate some variables (e.g. method A vs. method B) and control others (e.g. ability, experience, experimental objects)
  - **Quasi-experiments**: you could not really control all variables
From surveys to case studies...

- Evaluating state of the art and practice
- No user involvement
- Tool selection and tailoring

- Evaluating specific aspects of a technology in a controlled environment
- Careful design
- Replication

- Evaluating the whole technology on a realistic project
- Lower level of control than experiments

[Linkerman and Rombach, 97]
Experimental principles [Wohlin et al., 1999]

Theory

- Cause Construct
- Effect Construct

Experiment objective

cause-effect construct

treatment-outcome construct

Experiment operation

Observation

- Treatment
  - Independent variable
- Outcome
  - Dependent variable
Observational Studies

- Observe the evolution of a software system
  - Different points of view
- Possibly, relate different variables
- Even more, try to come out with some principles
Examples

- Laws of software evolution [Lehman and Belady, 1980]
  - ...
  - *As an E-type system evolves its complexity increases unless work is done to maintain or reduce it.*
  - ...

- Software aging [Parnas, 1994]
  - *Like human aging, software aging is inevitable, but like human aging, there are things that we can do to slow down the process and, sometimes, even reverse its effects.*

- Evolution of software entropy [Hassan, ICSE 2009]
CK Metrics for bug prediction
[Gyimothy et al., 2005]

- Classification of faulty classes in Mozilla using Chindamber and Kemerer Metrics
  - WMC (Weighted Method per Class)
  - DIT (Depth of Inheritance Tree)
  - RFC (Response For a Class)
  - NOC (Number of Children)
  - CBO (Coupling Between Object classes)
  - LCOM (Lack of Cohesion on Methods)

- Use of logistic regression model

Bug introducing changes [Kim et al.]

- Kim et al. predicted changes introducing bugs
- Machine learning algorithm (SVM)
- Independent variables: features of the change delta

<table>
<thead>
<tr>
<th>Feature Group</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Added Delta</td>
<td>If, while, for, ==</td>
</tr>
<tr>
<td>Deleted Delta</td>
<td>True, 0, &lt;, ++, int</td>
</tr>
<tr>
<td>Directory/File Name</td>
<td>Src, module, java</td>
</tr>
<tr>
<td>Change Log</td>
<td>Fix, added, new</td>
</tr>
<tr>
<td>New Revision Source Code</td>
<td>If,</td>
</tr>
<tr>
<td>Metadata</td>
<td>Author: hunkim, commit hour: 12</td>
</tr>
<tr>
<td>Complexity Metrics</td>
<td>LOC: 34, Cyclomatic: 10</td>
</tr>
</tbody>
</table>

In summary...

- We often observe two or more variables
- We correlate them
- ...or even build prediction models that actually work pretty well 😊

- So... everything looks pretty nice...
- We got a strong paper... but...
The bad part…

- We know that for sure we are missing something…
  - Do classes change more/exhibit bugs because of certain metrics?
  - Did really the introduction of an additional conditional or pointer increase the fault-proneness?
  - Do antipatterns make systems more change-prone?
  - Are co-occurring terms related? (as from Andi’s keynote)
  - .. Or rather they change because they have to…
Correlation, not causation

- One thing should be very clear
- The best we can do in our study is to obtain statistically significant models
  - Significant and high correlation
  - Significant differences
  - Prediction models with good performances

- We cannot (in any way) precisely identify the cause-effect relationship
Threats to validity discussion

- At the minimum, this is what we (should) do
  - To explicitly define the boundaries of our findings
  - … and to protect the paper against attacks 😊

- Readings on how to properly handle threats to validity
Ambiguity about direction of casual influence

- A causes B, B causes A, or X causes A and B?
- e.g. correlation between complexity and fault-proneness
  - Complexity causes fault-proneness… (A)
  - Could it be that fault-prone code (B) tend to be on average more complex (A)?
  - Or else problem-specific factors (X) make code more complex (A) and fault-prone (B)
Experiment:
Basic UML vs. Stereotypes

Boxplots: Conallen

No significant differences found except for Experiment 2…
Interaction with Ability and Experience

<table>
<thead>
<tr>
<th>Method</th>
<th>Mean of F-Measure</th>
</tr>
</thead>
<tbody>
<tr>
<td>Conallen</td>
<td></td>
</tr>
<tr>
<td>UML</td>
<td></td>
</tr>
<tr>
<td>Low Ability</td>
<td>0.58, 0.60, 0.62, 0.64, 0.66</td>
</tr>
<tr>
<td>High Ability</td>
<td>0.68</td>
</tr>
</tbody>
</table>

Experience

<table>
<thead>
<tr>
<th>Method</th>
<th>Mean of F-Measure</th>
</tr>
</thead>
<tbody>
<tr>
<td>Conallen</td>
<td></td>
</tr>
<tr>
<td>UML</td>
<td></td>
</tr>
<tr>
<td>Undergrad</td>
<td>0.68</td>
</tr>
<tr>
<td>Grad</td>
<td></td>
</tr>
</tbody>
</table>

The comprehension level does not depend only on the kind of diagram, but rather on the interaction with Ability/Experience…
Is that sufficient?

- Discussing threats avoids that people mis-interpret our results.
- Often, what we can say is that “we can claim correlation, while we can’t claim causation”
- Ok fine..
- After that, are our results still useful?
Meaningless models easy to find

R²=0.90

R²=0.70
However...
If we find correlations many times...
We cannot claim for causation…

- We know well that we would never be able to really claim for causation.

- Nevertheless, solid studies that found significant correlations are useful.
  - Especially if multiple studies show consistent results.
  - Replication is therefore important!

- To make them more useful, we should try to find some qualitative explanation of our findings.
Qualitative and Quantitative studies

- **Quantitative:** to get numerical relations among variables
  - Are programmers more productive with Java than with C#?
  - Are defects correlated with Chidamber-Kemerer metrics?

- **Qualitative:** to interpret a phenomenon just observing it in its context
  - Interview developers to know why a given method improves their productivity
  - Mine data from software repositories
Key ingredients

- Combining quantitative with qualitative studies
  - Sources:
    - Mining software repositories
    - Developers/experimental subject questionnaires/interviews
- Consider variables that might have an effect on our main factor/interact with the treatment
  - E.g. ability/experience of developers
- Use better statistical models
  - To capture relations other models cannot capture
    - E.g. temporal relations
- Replicate, replicate, replicate…
Mining Software Repositories

-软件复杂度
-故障预测
-努力估计
-变更传播
-变更模式
-演化可视化

知识推断
分类
关联
聚类
(algoritmi di data mining)
MSR Studies: sources of information

Versioning System Logs

Release Notes

Bug Reports

Source code comments

Security Bulletins

Mailing lists
Combining multiple sources

- CVS/SVN Commit note
  - “fix 367920 setting pop3 messages as junk/not junk ignored when message quarantining turned on sr=mscott”

- Regular expression matching e.g.
  - (b) [ug]{0,2}\setminus{s*}[id]{0,3}\setminus{id|fix|pr|$ $#}$ $[ \setminus{s}$ $#]=]*\setminus{[?([0-9]{4,6})]}\setminus{setminus]}?
Techniques

- **Code browsing**
  - Sometimes the only thing you can do
  - E.g. understand for what features design patterns are used for

- **Pattern matching / text mining**
  - Simple when you know what do you search
  - E.g. identify certain types of commits
  - E.g. find all NPE-related bugs
Mining commit notes

- Identify Refactoring [Ratzinger et al., MSR 2008]

| ArgoUML | Style issues. Privatised stuff.  
|         | Refactoring of the Model component Issue 2696. This part is making the Factory and Helper interfaces accessed from the Model class.  
|         | Replaced deprecated log4j Category with Logger.  
|         | Some cleanups in the use of the Model interfaces. |
| Eclipse-JDT | Refactoring - created new internal package structure  
|           | moved code manipulation & JavaModelUtil to corext  
|           | API cleanup  
|           | Removed unnecessary receivers  
|           | removed unneeded local vars & reduced synthetic accessors |

- Problems:
  - Inconsistent patterns
  - Commit notes can just refer to bug reports...
Issues and Challenges

- Integration of different repositories
  - Relatively easy in some cases, less trivial in other cases
  - E.g. can we trace changes to vulnerability bulletins? (when CVE not referred)
  - RegExp matching, IR techniques inspired from traceability recovery

- Information overload
  - [G. Murphy, ICSM 2007]
  - Indeed we have many sources of information, how we can effectively select the useful data?
Issues and Challenges

- Secret life of bugs
  - Jorge Aranda, Gina Venolia: The secret life of bugs: Going past the errors and omissions in software repositories. ICSE 2009: 298-308
  - Communication in software projects is not always recorded
  - People communicate by phone, chat, etc.
Issues and Challenges

- Information incomplete or incorrect
  - Pattern “Fixed #ID” in commit notes not always followed
  - Not all issues posted in Bugzilla are bugs
    - Including non-Enhancements
      - Giuliano Antoniol, Kamel Ayari, Massimiliano Di Penta, Foutse Khomh, Yann-Gaël Guéhéneuc: Is it a bug or an enhancement?: a text-based approach to classify change requests. CASCON 2008: 23
Human Studies: sources of information

- Survey Questionnaires
- Session Taping
- Interviews
- Screen capturing
- Eye tracking systems
- Event monitoring
Post-Experiment Questionnaire

- Used in controlled experiments after subjects have performed their tasks
- Used to understand:
  - Whether anything went wrong with clarity of objectives material, time available, tasks
  - How much time (approx) subjects spent on particular artifacts
  - Whether they “felt” a particular method easier/better than another
# Example of Post-experiment Questionnaire

<table>
<thead>
<tr>
<th>ID</th>
<th>Question</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q1</td>
<td>I had enough time to perform the lab tasks (1-5).</td>
</tr>
<tr>
<td>Q2</td>
<td>The objectives of the lab were perfectly clear to me (1-5).</td>
</tr>
<tr>
<td>Q3</td>
<td>The questions were clear to me (1-5).</td>
</tr>
<tr>
<td>Q4</td>
<td>I experienced no difficulty in reading the diagrams (1-5).</td>
</tr>
<tr>
<td>Q5</td>
<td>I experienced no difficulty in reading the source code (1-5).</td>
</tr>
<tr>
<td>Q6</td>
<td>How much time (as a percentage) did you spend looking at class diagrams?</td>
</tr>
<tr>
<td></td>
<td>(A. &lt;20%; B. &gt;=20% and &lt;40%; C. &gt;=40% and &lt;60%; D. &gt;=60% and &lt;80%; E. &gt;=80%)</td>
</tr>
<tr>
<td>Q7</td>
<td>How much time (as a percentage) did you spend for source code browsing?</td>
</tr>
<tr>
<td></td>
<td>(A. &lt;20%; B. &gt;=20% and &lt;40%; C. &gt;=40% and &lt;60%; D. &gt;=60% and &lt;80%; E. &gt;=80%)</td>
</tr>
<tr>
<td>Q8</td>
<td>I understood the meaning of Conallen's stereotypes (1-5).</td>
</tr>
<tr>
<td>Q9</td>
<td>I found Conallen’s stereotyped diagrams useful (1-5).</td>
</tr>
</tbody>
</table>

1 = strongly agree, 2 = Agree, 3 = Not certain, 4 = Disagree, 5 = strongly disagree.
Pros and cons of survey questionnaires

- Help to better understand the subjects behavior during an experiment 😊
- Can be used (of course) if you have developers available
  - Controlled experiments, in vivo case studies
  - Not possible for MSR studies
- Risk of bias very high 😞
  - Sometimes subjects tend to be overly positive or negative
  - It remains a purely qualitative feedback
  - **Don’t try to make strong conclusions based only on that**
Interviewing

- Alternative to survey questionnaires
  - Respondent better think about the answer 😊
  - … but they could feel under pressure 😞
  - The questions can be adapted case by case 😊
  - … but the risk is to have a too unstructured set of answers 😞
    - Difficult to make comparisons
Contacting team members

- Questionnaires/surveys cannot be applied to MSR studies
- However it is worth trying to contact project contributors / core project members
  - Instead of just “guessing”
  - They may or may not respond, but it costs nothing…
Eye Tracking: Pros and Cons

- You can really record what subjects looked at during the study 😊
- Might be somewhat expensive 😐
- Very likely, you need to perform the study in sequence, with a few subjects only 😞
- Complex tasks difficult to be tracked 😞
Other monitoring techniques

- Taping the session
- Recording (thinking aloud)
- Intercepting events on the machine

- Problems:
  - Too invasive
  - Data analysis might require a huge amount of work
Combining qualitative and quantitative information

A few examples...
Examples of MSR Studies

- **Design Pattern & code change-proneness**

- **Tracking clone maintenance**

- **Tracking vulnerabilities**
Study 1: Design patterns & code change proneness

- Claimed benefits of design patterns
  - reusability, maintainability, comprehensibility
- Above all, resilience to changes
  - “Each design pattern lets some aspect of system structure vary independently of other aspects, thereby making a system more robust to a particular kind of change” [Gamma et al., 1995]

- We conducted a study to understand
  - What patterns tend to change more frequently (if any)
  - What kinds of changes occurred on patterns
  - Whether patterns make the code more robust to change
Change Frequency in ArgoUML

- Adapter-Command: 0.010
- Composite: 0.008
- Decorator: 0.006
- Factory: 0.004
- Observer: 0.002
- Prototype: 0.000
- Singleton: 0.000
- State-Strategy: 0.000
- Template: 0.000
Change Frequency in Eclipse-JDT

The diagram shows the change frequency for various design patterns in Eclipse-JDT. The x-axis represents different design patterns, including Adapter-Command, Composite, Decorator, Factory, Observer, Prototype, Singleton, State-Strategy, Template, and Observer. The y-axis represents the change frequency, ranging from 0.000 to 0.010.

The pattern with the highest change frequency is the State-Strategy pattern, followed by the Observer pattern. The Adapter-Command pattern has the lowest change frequency.

The diagram highlights the relative change frequency among the different patterns, allowing for a quick visual comparison of their usage in the Eclipse-JDT environment.
We found that some design patterns change more than others.

We found that some patterns underwent some particular kinds of changes.

But...

What could be the real reason for such changes?

Let’s look where the design patterns are used.
Most frequently changed patterns

<table>
<thead>
<tr>
<th>Patterns</th>
<th>J HotDraw</th>
<th>ArgoUML</th>
<th>Eclipse-J DT</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Observer, Composite</td>
<td>Adapter-Command, Decorator, Factory</td>
<td>Visitor</td>
</tr>
<tr>
<td>Used for</td>
<td>Model View Controller of Draws, Handling composite figures</td>
<td>Adapting/ decorating UML objects to different views Execute menu actions</td>
<td>Visiting Java AST</td>
</tr>
<tr>
<td>Purpose of change</td>
<td>Adding new draw elements</td>
<td>Adding new menu actions and presentations</td>
<td>Adding new code analyses</td>
</tr>
</tbody>
</table>
Study 2: Tracking clone maintenance

- **Common wisdom:** clones are harmful
  - One maintains a code fragment, and if the change is not properly propagated on clones...

- Studies in the past investigated on the possibility of clone automatic refactoring
  - Too risky (could introduce bugs)
  - Too much effort required

- However... recent empirical studies showed that...
Clones are not harmful!

- Recent (and past) studies suggested clones are not necessarily harmful
- Developers use cloning as a development practices
  - E.g. code templating
- However, clone evolution should be monitored
  - Tool for manually keeping track of clones during maintenance [Duala-Ekoko and Robillard, 2007]
  - Analyzing clone genealogies [Geiger et al., 2005, Kim et al., 2005]
- **Our study: track clone evolution patterns**
Clone Evolution patterns

**Consistent change**

**Late propagation**

**Late propagation**

**Independent evolution**
Clone evolution patterns

- ArgoUML
  - CO: 55%
  - IE: 34%
  - L2: 4%
  - LP: 3%
  - UN: 0%

- JBoss
  - CO: 52%
  - IE: 40%
  - L2: 2%
  - LP: 3%
  - UN: 5%

- OpenSSH
  - CO: 71%
  - IE: 24%
  - L2: 0%
  - LP: 4%
  - UN: 0%

- PostgreSQL
  - CO: 38%
  - IE: 39%
  - L2: 6%
  - LP: 1%
  - UN: 1%
Need for Qualitative Analysis

- Why some clones are consistently changed and others evolve independently?
- Why there are late propagations?

- Look at what the cloned code is used for
  - Source code browsing
  - Java Doc, release notes, other documentation
- Look at CVS/SVN logs related to cloned code changes
Qualitative analysis results

- **Consistent Change (ArgoUML)**
  - The class `GenAncestorClasses` has been cloned from `GenDescendantClasses`. Both are utility classes used to navigate class hierarchies in UML diagrams.
  - Such classes underwent different refactoring changes, always consistently propagated.

- **Independent Evolution (ArgoUML)**
  - The classes `GeneratorJava` and `GeneratorDisplay` contain some cloned methods.
  - The second class becomes more complex in newer ArgoUML versions to account for enhanced visualization features.
  - Starting from the revision 1.8 of `GeneratorJava` and 1.4 of `GeneratorDisplay`, such classes evolved independently.
Qualitative analysis results

- Late Propagation (PostgreSQL)
  - The modules `parse_oper.c` and `parse_func.c` contain two block size clones.
  - The first underwent to a bug fixing (August, 26 1999)
  - The same bug was discovered six months later on the other clone (February, 20 2000)
  - CVS commit note:
    “...I had previously fixed the identical bug in `oper_select_candidate`, but didn't realize that the same error was repeated over here...”
Study 3: Tracking source code vulnerabilities

- Vulnerable instructions in the source code are crucial problem for maintainers
  - Buffer overflows, SQL injections, cross-site scripting (XSS)

- **Goal:** study the evolution of statically detected vulnerabilities with the purpose of determining their density trend and their permanence in the system.

- **Research questions related to:**
  - Evolution of number of vulnerabilities
  - Vulnerabilities being removed faster than others
  - Vulnerability decay
Evolution of vulnerability density

Samba - Overall

- Splint vulnerabilities tend to have a lower density (thorough analysis)
- Initially, a high number vulnerabilities detected by RATS
  - Pre-release, then vulnerabilities removed by security patches
- No trend detected (ADF test)

Squid - Buffer Overflows

- **Buffer Overflows introduced** at release 2.3 STABLE3
- Then **removed** in the subsequent releases 2.4STABLE7 and 2.5STABLE7 with proper **security patches**
  - As documented in the system history
Vulnerability Decay

- **Buffer Overflows** tend to disappear significantly quicker than most of other vulnerabilities (M-W test)

- File System vulnerabilities the quickest to be fixed
  - Samba domain: sharing files and printers
Evidence of vulnerability removals

- Vulnerability removal tools are often imprecise
  - A lot of false positives
- Sometimes a vulnerability is just removed accidentally
  - We should not really claim in that case that the developer put effort to remove it
- → We inspected commit notes to seek evidence of vulnerability removals
Some examples of commit notes

- Squid:
  - “Fixed buffer overflow bug in whois.cc....”
  - “No longer use scanf() in some places...”
  - “…replaced some strcpy() calls with memset() and strncpy()…”

- Horde:
  - “… close security hole - stream data directly to browser instead of reading it into memory first.”
Some examples of commit notes

- **Samba:**
  - “Don’t use static memory malloc it... Jeremy.”
  - “move to SAFE_FREE(); ‘‘don’t use gets() ! ”
  - “don’t use strcpy ”
  - “‘This is a security audit change of the main source. It removed all occurrences of the following functions: sprintf strcpy strcat The replacements are slprintf safe_strcpy and safe_strcat.... Jeremy.”
Use of Granger Causality to capture Temporal Relations

Gerardo Canfora, Michele Ceccarelli, Luigi Cerulo, Massimiliano Di Penta:


* © Andi Marcus, Sept 20 2010
Change impact analysis
Association rules discovery

Detected couplings: A→C, B→D, D→E
What we may miss here?

Changes occurring in snapshots
What we propose

Multivariate time series analysis

Future changes to an artifacts depend on previous changes on other artifacts

\[
f_2(t) = c_1 + \alpha_1 f_1(t - 1) + \alpha_2 f_1(t - 2) + \cdots + \alpha_p f_1(t - p) +
\]
\[
+ \beta_1 f_2(t - 1) + \beta_2 f_2(t - 2) + \cdots + \beta_p f_2(t - p) + u(t)
\]
\[
f_2(t) = c_1 + \gamma_1 f_1(t - 1) + \gamma_2 f_1(t - 2) + \cdots + \gamma_p f_1(t - p) + e(t)
\]

Correlation between future changes to \( f_2 \) and previous changes to \( f_1 \) tested using the Granger causality test

\[ H_0: f_1 \text{ does not cause } f_2 \]
Association rules vs. Granger

Conjecture: the two techniques are complementary

Association rules: $A \rightarrow C, B \rightarrow D, D \rightarrow E$

Granger causality test: $A \rightarrow \{B,D\}, C \rightarrow \{D,E\}$
### Comparing the two techniques

<table>
<thead>
<tr>
<th>Precision</th>
<th>FreeBSD -i376</th>
<th>Mylyn</th>
<th>Rhino</th>
<th>Squid</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arules</td>
<td>Arules</td>
<td>Arules</td>
<td>Arules</td>
<td>Arules</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Recall</th>
<th>FreeBSD -i376</th>
<th>Mylyn</th>
<th>Rhino</th>
<th>Squid</th>
</tr>
</thead>
<tbody>
<tr>
<td>Granger</td>
<td>No difference</td>
<td>Arules</td>
<td>Granger</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>FMeasure</th>
<th>FreeBSD -i376</th>
<th>Mylyn</th>
<th>Rhino</th>
<th>Squid</th>
</tr>
</thead>
<tbody>
<tr>
<td>Granger</td>
<td>No difference</td>
<td>Arules</td>
<td>Granger</td>
<td></td>
</tr>
</tbody>
</table>
Union, Intersection (Mylyn)

Top Rank

Mean of True Positives

Arules
Granger
Granger and Arules
Granger or Arules

Top Rank

Mean of True Positives
Combining techniques (Mylyn)

Graph showing the mean of F-Measure as a function of Top Rank. The graph compares different combinations of Arules and Granger techniques, such as Arules, Granger, Granger*0.25 + Arules*0.75, and Granger*0.75 + Arules*0.25.
Where is Granger helping out?

FreeBSD-i386

2 months later...

\begin{itemize}
\item \texttt{linux\_dummy.c}
\item \texttt{linux\_ysent.c}
\item \texttt{ibcs2\_xenix\_sysent.c}
\item \texttt{ibcs2\_isc\_sysent.c}
\end{itemize}

Need to regenerate files to account for new system event protocols

“I Implement fdatasync in terms of fsync. The regeneration of proto.h syscall.h and sysent.h was probably forgotten after the last changesyscalls.master.”
Where is Granger helping out?

Samba

"smbd/auth server: Doco we want to use cli_nt_error here soon
smbd/password.c…”

“… added automatic mapping between dos and nt error codes…”

8 August 2001

auth_domain.c
auth_server.c
auth_rhost.c
auth_unix.c
auth_smbpasswd.c
auth.c
reply.c

27 August 2001

errors.c
...

87
Conclusions

- Principles of software evolution often empirically derived
  - Observational studies
  - Building predictive models
  - Correlating phenomena

- Causation cannot be really claimed, in software engineering as in other disciplines

- But we should really try to convince people that these models are meaningful.
What we need to do

- Complementing quantitative with qualitative studies
  - Mining software repositories
  - Developer interviews, survey questionnaires, email contacts
- Better identifying (co)factors affecting our results
- Using better statistical tools
  - Sometimes they are just out of there…
  - Look at what they do in other disciplines (medicine, biology, bioinformatics)
- Replicating studies
Challenges

- Integrating several heterogeneous sources of information
- Mining huge quantities of data
- Relying on (often) unreliable and incomplete sources
- Avoiding to be too intrusive in developers’ task
- Dealing with developers’ subjectiveness
- Ensuring consistent, reliable replications of the studies
Acknowledgements

- Giulio Antoniol
- Lerina Aversano
- Gerardo Canfora
- Michele Ceccarelli
- Luigi Cerulo
- Mariano Ceccato
- Tina Del Grosso
- Yann-Gaël Guéhéneuc
- Foutse Khomh
- Filippo Ricca
- Suresh Thummalapenta
- Paolo Tonella
- Marco Torchiano