Why We Need Empirical Software Engineering

October 6, 2007

Elaine Weyuker
AT&T Labs – Research
ISSUES TO CONSIDER

- Awareness by practitioners
- Relevant to practitioners
- Useable by practitioners
Researcher’s Perspective

- Are practitioners aware of our research?
- Don’t they ever read anything?
- Do practitioners think it’s relevant?
- Don’t they understand its value?
- Are practitioners using it?
- Why aren’t they smart enough to use it?
- What can we do differently to get our research used?
- How can we knock some sense into their heads?
Practitioner’s Perspective

• Is this research relevant?
• Why don’t they solve a real problem?
• Is it applicable to my project?
• Who can afford to take such a risk?
• Is there adequate tool support?
• They call this a tool?
• Do I have the expertise to use it?
• I need to know WHAT?
• Is it usable for a project of my scale?
• This is only good for toy software.
Why Empirical Studies?  
(Researcher’s Perspective)

It allows us to validate our theories, ideas, techniques, and modify them if appropriate. Empirical studies provide us with our first feedback and help us get feedback from practitioners.
Why Empirical Studies?
(Practitioner’s Perspective)

If practitioners don’t see how the research can be applied to systems that “look like” the sort of systems that they build, they cannot and will not take the risk of trying the research.
GOAL

To determine which files of a large industrial software system with multiple releases are particularly likely to be fault-prone.
APPROACH

Identify properties that are likely to affect fault-proneness, and then build a statistical model to make predictions.
Finding The First Subject System

• Approached development management that we had interacted with previously and had helped in the past.

• Convincing very busy developers and testers with tight deadlines to give us access to their change database, answer questions, provide advice, and trust us not to disturb their data.
Subgoal 1: Determine Whether Faults are Non-uniformly Distributed

We did an initial large case study using 12 successive releases (about 3 years in the field) of an inventory control system containing roughly ½ million LOC. Data was collected during 9 development stages. About ¾ of the files were written in java, with smaller numbers written in shell script, makefiles, xml, html, perl, c, sql, and awk.
Number of Files and Number of Files Containing any Faults
Subgoal 2: Identify which characteristics are associated with the files that contained the highest fault densities.
Considered Structural Characteristics of Files

- Size of file (KLOC)
- Age of file
- New to the current release, and if not, whether it was changed during prior release?
- The number and magnitude of changes made.
- The number of observed faults during earlier releases.
- The number of faults observed during early development stages.
- Programming language used.
Findings

• Size is a very poor predictor of fault density.
• New files and files that had been changed in the last release were significantly more likely to contain faults than files that had not changed.
• Other weaker factors include file age, number of faults in previous release, programming language, and release number.
Preliminary Findings

• Presented them at an in-house software symposium aimed at practitioners.

• Goal was to alert projects of what we were doing to get feedback and interest other projects in being subjects in future empirical studies.
What We Learned

Fault density was the wrong characteristic to target. Instead we should be trying to target files with the largest numbers of faults.
Moving on to Predictions

- Presented findings at research talk.
- Enlisted a statistician to collaborate and help us doing the statistical modeling.
Doing the Actual Prediction

We used the identified characteristics, properly weighted, to build a statistical model (negative binomial regression model), and made predictions for releases 3 through 17.
Percent of Bugs Correctly Predicted by 20% of Files Selected by Model (Average = 83%)
A Second Case Study

• The head of system testing for a service provisioning system who was at the in-house practitioner talk, volunteered their project.
• We applied our model to this system which contained 9 releases (representing two years of field exposure).
Number of Files and Faults by Release (System 2)

![Graph showing number of files and faults by release.](image)
Dealing with Paucity of Data

We aggregated Releases 2-5 (Release B) and 6-9 (Release C) into single releases. Release A (Release 1) contained 24 faults. Release B contained 153 and Release C contained 130 faults.

Using Releases A and B to make predictions for Release C, the 20% of the files selected by the model again included 83% of the faults.
A Project Without Regular Releases

- Automated Voice Response System
- Followed for 27 calendar months
- Month 1 contained 61 files and 16 KLOC
- Month 27 contained 1888 files and 321 KLOC
- Very heterogeneous system - 34 languages used in month 27.
Dealing With the Lack of Regular Releases

• Created Synthetic Releases.
  – 27 “Monthly Releases”
  – 9 “Quarterly Releases”

• Predictions were less accurate – on average the top 20% of the predicted files contained 75% of the faults.
Project 4

• A maintenance support system.
• Developed and maintained by a different company.
• Very mature system - 9 years of field data.
• The top 20% of the files contained 84% of the faults.
# Systems Studied

<table>
<thead>
<tr>
<th>System Type</th>
<th>Period Covered</th>
<th>20% Files</th>
</tr>
</thead>
<tbody>
<tr>
<td>Inventory</td>
<td>4 years</td>
<td>83%</td>
</tr>
<tr>
<td>Provisioning</td>
<td>2 years</td>
<td>83%</td>
</tr>
<tr>
<td>Voice Resp</td>
<td>2.25 years</td>
<td>75%</td>
</tr>
<tr>
<td>Maintenance Support</td>
<td>9 years</td>
<td>84%</td>
</tr>
</tbody>
</table>
Build a Tool

Practitioners Must Have Tools that Automate the Process and Do Not Require Highly Specialized Expertise.
Important Features of the Research

• Develop a theory.
• Get initial interest from practitioners.
• Present preliminary results to practitioners and listen to feedback.
• Perform multiple empirical studies using real large systems to validate results and determine generality.
• Present results to both practitioners and researchers and listen to feedback.
• Collaborate with other researchers with complimentary skills.
• Recognize that the process may be very time-consuming.
• Automate!! No one will use your results if it requires a large amount of time and expertise to use your technology.
ISSUES TO CONSIDER

- Awareness by practitioners
- Relevant to practitioners
- Useable by practitioners