A Brief History of Software

From Bell Labs to MSR

Thomas Ball

Research in Software Engineering, MSR
A Brief History of Software
From Bell Labs to “MSR”

Thomas Ball

Research in Software Engineering, MSR
A Brief History of Me

• 1965, born NYC
• 1969-70, Thailand
• 1977-1978
  – chorus, Star Wars, Space Invaders
• 1979, Basic
• 1980-82, Apple II games, w. pal Eric Varsanyi
  – Falcons, ~10K units
  – Warp Destroyer, ~1k
• 1983-86, Cornell
  – grad compilers + PL theory
  – synthesizer generator
• 1987-93, U. Wisconsin
  – profiling w. Jim Larus
  – slicing w. Susan Horwitz
• 1993-95, AT&T Bell Labs
  – Software Production Research Dept.
  – software visualization
  – code decay project
• 1996-99, Lucent Technologies, Bell Labs
  – web languages
  – testing/dynamic analysis
• 1999-, Microsoft Research
  – SLAM, Static Driver Verifier
  – SRR group (now 10 strong)
  – Part of Research in SE
From Bell Labs to "MSR"

(A Personal Account)

Mining
Version Control
Data

@ Bell Labs

Mining Software Repositories

@ MSR
Version Control in 5ESS c. 1984

Diagram:
- Description
- Time Date
- Developer
- delta
- MR
- File, Module
- #lines add., del.
#include <stdio.h>
main() {
    printf("Hello,world\n");
}

#include <stdio.h>
main() {
    printf("Hello,world\n");
    exit(0);
}

#include <stdio.h>
main() {
    printf("Hello,world\n");
}

Delta History

#include <stdio.h>
main() {
    printf("Hello,world\n");
}

#include <stdio.h>
main() {
    printf("Hello,world\n");
    exit(0);
}

#include <stdio.h>
main() {
    printf("Hello,world\n");
}

#include <stdio.h>
main() {
    printf("Hello,world\n");
}

Version Sensitive Editing: Change History as a Programming Tool


© Springer-Verlag Berlin Heidelberg 1998
MSR, Take #1 : Task → Tool
Task: Find the Bug

```c
String FindSource(String base, String dir) {
    DIR * dirp = opendir(dir);
    for (int i = 0; i < NS; ++i) { // Loop over suffix list
        String tmp = base + suffix[i]; // Target name to find
        for (dirent *de = readdir(dirp); de != NULL; de = readdir(dirp))
            if (tmp == de->d_name) { // We found it, stop looking
                return tmp;
            }
        rewinddir(dirp);
    }
    closedir(dirp);
    return ""; // No match was found
}
```

http://ix.cs.uoregon.edu/~datkins/ve.html
String FindSource(String base, String dir) {
    DIR * dirp = opendir(dir);
    String result; // The filename, if found
    for (int i = 0; i < NS; ++i) { // Loop over suffix list
        String tmp = base + suffix[i]; // Target name to find
        for (dirent *de = readdir(dirp); de != NULL; de = readdir(dirp))
            if (tmp == de->d_name) { // We found it, stop looking
                result = tmp;
                break;
            }
    }
    return tmp;
}
rewinddir(dirp);
closedir(dirp);
return result; // Return the found name (may be null)
return ""; // No match was found

Deleted by MR 595 by vz, 97/11/15, approved [Stop source search at 1st match]
MR 467 by dla, 97/09/21, integrated [Find source using list of suffixes]
"findsource.c", line 15 of 23
MSR, Take #2: Gestalt

Software Visualization in the Large
Ball, Eick

Code Age
Code Ownership
Preprocessor Scopes
If Your Version Control System Could Talk...
If Your Version Control System Could Talk...

Class Inheritance

Ball, Porter, Siy  ICSE '97 Workshop
If Your Version Control System Could Talk...

Class Inheritance

Modified Together

Ball, Porter, Siy

ICSE '97 Workshop
MSR, Take #3: Significance
-or-
Software Engineering meets Statistics
Does Code Decay? Assessing the Evidence from Change Management Data

Stephen G. Eick, Todd L. Graves, Alan F. Karr, J. S. Marron, Audris Mockus

Abstract

A central feature of the evolution of large software systems is that change — which is necessary to add new functionality, accommodate new hardware and repair faults — becomes increasingly difficult over time. In this paper we approach this phenomenon, which we term code decay, scientifically and statistically. We define code decay, and propose a number of measurements (code decay indices) on software, and on the organizations that produce it, that serve as symptoms, risk factors and predictors of decay. Using an unusually rich data set (the fifteen-plus year change history of the millions of lines of software for a telephone switching system), we find mixed but on the whole persuasive statistical evidence of code decay, which is corroborated by developers of the code. Suggestive indications that perfective maintenance can retard code decay are also discussed.
Files Touched Per Change

Highlighted smooths

Audris Mockus

works in Software Technology Research Department of Avaya Labs Research, AVAYA located at Rm 2D-30, 233 Mt. Airy Road, Basking Ridge, NJ 07920. ph: +1 908 696 5608, fax: +1 908 696 5402 audris@research.avayalabs.com. His bibliography and curriculum vitae.

He is interested in inventing ways to describe and understand large, dynamic, and complex systems, such as large software systems. His goal is to find ways to summarize system evolution data, to gain new insights, and to design new analysis methods and tools that would significantly improve the understanding and control of such systems.

Results from analysis of software changes include the ability to determine why the software is changed, how difficult each change is, evaluate the impact of a software tool or process, and predict risk that a change will break existing features. It also allows quantification of key features of a development process including Open Source development process. Details are in the Apache server case study and the scripts used to process Apache and Mozilla change data.
Effort Estimation

Inferring Change Effort from Configuration Management Databases  Todd L. Graves and Audris Mockus
5th IEEE International Software Metrics Symposium (METRICS 1998),
Using Version Control Data to Evaluate the Impact of Software Tools

David Atkins, Thomas Ball, Todd Graves, and Audris Mockus

ICSE 1999
if (!PreCheckRoute(route))
    return FAIL;
    dest = GetDest(route);
    if (dest.port == 0 || dest.module == 0) {
        return(RouteLocal(route));
    }
    DoRoute(route);

MR 12467 by dla, 97/9/21, assigned [Local routing]
Versioning: 5A inside 4A
"route.c" [modified] line 67 of 241
Hypothesis

"When making a change involving version lines, developers are more effective when using VE than when using a standard text editor."
$E(\text{effort}) = \alpha \times \beta \times \text{SIZE}^\gamma \times \Theta_{\text{DEV TYPE}} \times \Theta_{\text{TOOL}}$

$\text{TOOL} \in \{\text{VE, HAND, NONE}\}$

\[
\frac{\Theta_{\text{HAND}}}{\Theta_{\text{VE}}} = 1.36
\]
MSR

Version Control
MSR

Build config/output
Component DB
Version Control
MSR

- Code metrics
- Dependencies
- Static defects
- Build
- Config/Output
- Component DB
- Version Control
Nachi Nagappan

NEW: Please consider submitting to the Second DEFECTS workshop co-located with ISSTA 2009 in Chicago!

Nachiappan Nagappan works in the Empirical Software Engineering and Measurement Research Group (ESM) at Microsoft Research. His research interests are in the field of software measurement focusing on software metrics, software reliability, failure analysis/prediction and empirical software engineering processes. Prior to his current position he earned his PhD from North Carolina State University in 2005.

My work primarily focuses on large-scale commercial Microsoft systems. Currently my research interest focuses on the application of software measurement and statistical modeling to software. My work is primarily targeted at making early estimates of software quality to predict post-release failures. We are currently focused on the next generation Windows operating system (Vista) leveraging the rich history of in-process and product metrics available from prior versions of the Windows operating system. I also work on measuring test effectiveness and in defect analysis. Of late I have also begun working on social network analysis of developer contributions. If you have any questions please feel free to send me email.

Email: |com|microsoft|nachin
Office: (425) 7XX-XXXX
Fax: (425) 936-7329
One Microsoft Way
Redmond, WA 98052

Links

Empirical Software Engineering and Measurement (ESM)

Publications

Professional activities

Personal
MSR, Take #4: Prediction
Failure-prediction
Failure-prediction
Failure-prediction

Building an Empirical body of knowledge
Applications

Resource allocation
- Focus QA on components more likely to fail

Decision making
- bug triage
- risk of fix
- test prioritization
Software Reliability

- Field failures (per binary)
- Collected post-release
- Statistical model building

failure prob. = f (metrics)
Using XP SP1 to predict Server 2003

- Windows XP-SP1 release
- 6 months to collect post-release failure
- Collect metrics

Time
Using XP SP1 to predict Server 2003

- 6 months to collect post-release failure

Collect metrics

Predicted failure-proneness

Windows XP-SP1 release

Windows Server 2003 release

Windows XP-SP1

Windows Server 2003

Time
Metric Evaluation at Microsoft

- Which metrics can predict failures?
  - Churn [ICSE 05a]
  - Code complexity [ICSE 06]
  - Dependencies [ESEM 07]
  - Static analysis tool defects [ICSE 05b]
  - Coverage [Tech Report]
  - Assertion density [ISSRE 06a]
  - People [ICSE 08, FSE 08]
  - Combining factors [ISSRE 06b, APSEC 06]
In Collaboration With

- Andy Begel, Brendan Murphy, Tom Zimmerman, MSR
- Vic Basili, UMD
- Prem Devanbu, UC Davis
- Harald Gall, Univ. of Zurich
- Audris Mockus, Avaya
- Laurie Williams, NCSU
- Andreas Zeller, Saarland Univ
Measuring Success

Predicted failure-prone

Precision = \frac{B}{A}

Observed failure-prone

Recall = \frac{B}{C}
## Classification Results for Windows Vista

<table>
<thead>
<tr>
<th>Model</th>
<th>Precision</th>
<th>Recall</th>
</tr>
</thead>
<tbody>
<tr>
<td>Organization Structure</td>
<td>86.2%</td>
<td>84.0%</td>
</tr>
<tr>
<td>Churn</td>
<td>78.6%</td>
<td>79.9%</td>
</tr>
<tr>
<td>Complexity</td>
<td>79.3%</td>
<td>66.0%</td>
</tr>
<tr>
<td>Dependencies</td>
<td>74.4%</td>
<td>69.9%</td>
</tr>
<tr>
<td>Coverage</td>
<td>83.8%</td>
<td>54.4%</td>
</tr>
<tr>
<td>Pre-Release Bugs</td>
<td>73.8%</td>
<td>62.9%</td>
</tr>
</tbody>
</table>
The Big Picture

- Software reliability
  - post-mortem analysis of field failures
  - statistical analysis and model building to evaluate product/process metrics + efficacy at predicting failures
  - Example: Using XP SP1 to predict Server 2003

- Effecting change
  - tools/methodology incorporate statistical models
  - inline evaluation of the process
  - evaluate and observe actions to quantify feedback
  - Example: Change Risk Analysis and Estimation (CRANE)
MSR, Take #1: Task → Tool
MSR, Take #5: Task $\rightarrow$ Tool

Support decision making with statistically valid evidence
From Bell Labs to "MSR"
From Bell Labs to "MSR"

The Version Editor

```c
String FindSource(String base, String dir) {
    DIR * dirc = opendir(dir);
    String result; // The filename, if found
    for (int i = 0; i < NS; ++i) { // Loop over suffix list
        String tmp = base + suffix[i]; // Target name to find
        dirent *de = readdir(dirc); de = NULL; de = readdir(dirc))
        if (tmp == de->d_name) { // We found it, stop looking
            result = tmp;
            break;
            return tmp;
        }
        rewinddir(dirc);
    }
    closedir(dirc);
    return result; // Return the found name (may be null)
    return ""; // No match was found
}
```

Deleted by MR 595 by vz, 97/11/15, approved [Stop source search at 1st match]
MR 467 by dia, 97/09/21, integrated [Find source using list of suffixes]
"findsource.c", line 15 of 23

we view (recent additions bold, deletions underlined)

Atkins et al.
Bell Labs 1986
From Bell Labs to "MSR"

The Version Editor

Deep Intellisense

String FindSource(String base, String dir) {
    DIR ^ dirp = opendir(dir);
    String result;    // The filename, if found
    for (int i = 0; i < NS; ++i) { // Loop over suffix list
        String tmp = base + suffix[i]; // Target name to find
        for (dirent *de = readdir(dirp); de != NULL; de = readdir(dirp))
            if (tmp == de->d_name) { // We found it, stop looking
                result = tmp;
                break;
                return tmp;
            }
    }
    rewinddir(dirp);
    closedir(dirp);
    return result;    // Return the found name (may be null)
    return "";    // No match was found
}

Deleted by MR 595 by vz, 97/11/15, approved [Stop source search at 1st match]
MR 467 by dia, 97/09/21, integrated [Find source using list of suffixes]
"findsource.c", line 15 of 23

ve view (recent additions bold, deletions underlined)

Atkins et al.
Bell Labs 1986

Holmes, Begel
MSR 2008
What Has Changed?

- Variety + volume of data
- Ease of access
  - open source
  - storage, network, DB
- Compute power
  - mining over giga/tera-bytes

Insight Into the Invisible!

• **Do Crosscutting Concerns Cause Defects?** (ICSE: TSE/TOSEM Session), Marc Eaddy, Thomas Zimmermann, Kaitlin D. Sherwood, Vibhav Garg, Gail C. Murphy, Nachiappan Nagappan, Alfred V. Aho. TSE/TOSEM Session.


• **HOLMES: Effective Statistical Debugging via Efficient Path Profiling** (ICSE: Research papers), Trishul Chilimbi, Ben Liblit, Krishna Mehra, Aditya Nori, Kapil Vaswani.

• **The Secret Life of Bugs: Going Past the Errors and Omissions in Software Repositories** (ICSE: Research Paper), Jorge Aranda, Gina Venolia. Every bug has a story behind it.

• **Improving Bug Tracking Systems** (ICSE: NIER Track), Thomas Zimmermann, Rahul Premraj, Jonathan Sillito, Silvia Breu.


• **Coordination in Large-Scale Software Teams** (CHASE Workshop). Andrew Begel, Nachiappan Nagappan, Christopher Poile, and Lucas Layman.