Socio-Technical Aspects of Long Term Embedded Systems Maintenance

Talk/Request for Comments

Prof. Dr. Wolfgang Mauerer
Siemens AG, Corporate Research, Munich &
Faculty of Computer Science and Mathematics
Technical University of Applied Sciences Regensburg

wolfgang.mauerer@oth-regensburg.de

Encrypted communication: GPG/PGP-ID 98356E1E, Fingerprint: 5920 9407 AB5C 8B28 3C7B 4F02 F16F 2523 9835 6E1E.
Technical

- Structured programming
- Appropriate languages
- Proper (idiomatic) use of libraries
- ...

Process

- Code reviews (formal/informal)
- Inspections and walkthroughs,
- Use of SW engineering tools (VCS, issue tracking, CI, …)
- ...

Alan Perlis on sugar consumption

When someone says »I want a programming language in which I need only say what I wish done,« give him a lollipop.

- Excellent MPEG decoders in Fortran
- Horrible QM simulations in Java
Conway’s »Law«

»Any organisation that designs a system (defined broadly) will produce a design whose structure is a copy of the organisation’s communication structure.«

Socio-Technical Congruence and Long-Term Maintenance

- Create awareness for LTM issues (i.e., pick right patches)
- Influence »decision makers« (i.e., label important fixes)
- Extract »shared project knowledge« (implicit processes)
Examples from (Commercial) Industry

Microsoft

- Predict bugs from organisational structure
- Successful approach!
- Requires a-priori knowledge of organisation

IBM

- Predict build issues from organisational structure
- Successful approach!
- Requires special data collection infrastructure

Consequences

- Social factors $\rightarrow$ software quality
- Unix reference manual does not cover humans $\rightarrow$ sociology required
Examples from (Commercial) Industry

Microsoft
- Predict bugs from organisational structure
- Successful approach!
- Requires a-priori knowledge of organisation

IBM
- Predict build issues from organisational structure
- Successful approach!
- Requires special data collection infrastructure

Consequences
- Social factors affect software quality
- Unix reference manual does not cover humans; sociology required
- Quantitative, of course
Pharmaceuticals

✓ A-priori understanding (to some extent)
✓ Tests & statistics
Pharmaceuticals

- ✓ A-priori understanding (to some extent)
- ✓ Tests & statistics

Software

- ✗ Comparative experiments
- ✗ Quantify people and behaviour
- ✗ Personal experience limited
# Codeface

## Goals
- *Automatically* determine collaboration structure from development artefacts
- Include temporal *dynamics*

## Approach
- Find relationships between developers
- Infer and verify communities
- Find structural properties of communities

## History
- Initially: Research project at Siemens Corporate Technology
- International academic cooperation
- Open source (mainly GPLv2)
Data source

commit 1bb22891a9609b235f8e43d0315d566f65197ef9  
Author: Mitchell Joblin <joblin@mail.com>  
Date:  April 15 13:22:10 2014 +0200  
Committer: Wolfgang Mauerer <maurerer@mail.com>  
Date:  May 1 23:54:18 2014 +0200  

Abort cluster analysis when matrix off diagonal sum is zero…

Signed-off-by: Mitchell Joblin <joblin@mail.com>  
Reviewed-by: Wolfgang Mauerer maurerer@mail.com

diff --git a/codeface/R/cluster/persons.r b/codeface/R/cluster/persons.r  
@@ -1001,8 +1001,14 @@ performAnalysis <- function(outdir, conf) {
   conf) {
     if (length(colnames(id.subsys)) == 2) {
       id.subsys <- NULL
-    - if(sum(adjMatrix) == 0) {
+    + if(sum(adjMatrix) == 0) {  
  Changed lines
Network construction

- Tagging ("Signed-off-by")
- Committer/author
- Overlapping code contributions
- Feature co-changes
Goal: Partitioning into subgraphs

- Strongly connected internally
- Weakly connected externally

Validation

- Statistical methods
- Sociological verification
Qemu 0.11.0 (Sep 2009) Virtualisation/Machine Emulation
Qemu 0.13.0 (Oct 2010) Virtualisation/Machine Emulation

Community Examples I
Git 1.8.3 (May 2013)

Revision Control System
Quality Estimation

- Meaningful community structures vs. random properties
- Randomise clusters
  - Rewire edges
  - Keep properties (e.g., “amount” of participation)
- $H_0$: Clustering stems from unorganised, random process.
- Reject $\rightarrow$ Decomposition makes sense
- Then: Large-scale sociological verification (surveys)
Quality Estimation

- We did the maths
- We asked people

Alternative Industrial Approach

Ken Schwaber says: A team has seven people (plus or minus two). Full stop!
How’s that relevant?

- Coordination structure vs. non-functional requirements
  - Are there long-term reliable structures?
  - Which persons?
- Handling team dynamics
  - Knowledge loss (i.e., will anyone fix my problem in 7 years?)
  - Knowledge distribution

Can’t beat manual knowledge

- Yes, you may know that for project $\langle x \rangle$
- But how about $\langle Y \rangle, \langle Z \rangle, \langle \zeta \rangle, \langle \Omega \rangle, \ldots$?
Random

- Randomly (iid) distributed edges (connections)
- Erdős-Rényi model: »typical« nodes (developers)
- Hub nodes: extremely rare

Scale free

- No »typical« nodes (developers)
- Hub nodes: frequent
- Large real-world networks (biology, sociologie, internet routers, …)
- Robust against random changes
<table>
<thead>
<tr>
<th>Random</th>
<th>Scale free</th>
</tr>
</thead>
<tbody>
<tr>
<td>▶ Distributed system knowledge ✓</td>
<td>▶ No »typical« nodes (developers)</td>
</tr>
<tr>
<td>▶ Bad scalability ≠ burn out individuals ✗</td>
<td>▶ Hub nodes: frequent</td>
</tr>
<tr>
<td></td>
<td>▶ Large real-world networks (biology, sociologie, internet routers, …)</td>
</tr>
<tr>
<td></td>
<td>▶ Robust against <em>random</em> changes</td>
</tr>
</tbody>
</table>

### Long-Term Implications

<table>
<thead>
<tr>
<th>Easy to find experts on anything</th>
</tr>
</thead>
<tbody>
<tr>
<td>Limited overall complexity</td>
</tr>
<tr>
<td>Bad technical subsystem isolation</td>
</tr>
</tbody>
</table>
Random

- Distributed system knowledge ✓
- Bad scalability ☞ burn out individuals ✗

Scale free

- Maintainer: Architectural (structural) knowledge ✓
- Hub dev hit by proton beam ☞ structural problems ✗

Long-Term Implications

- Easy to find experts on anything
- Limited overall complexity
- Bad technical subsystem isolation

Long-Term Implications

- Support or friction can come from maintainers
- Sudden disruption
<table>
<thead>
<tr>
<th>Hierarchical</th>
<th>Modular</th>
</tr>
</thead>
<tbody>
<tr>
<td>Developers: hierarchical layers</td>
<td>Developers form strongly connected communities</td>
</tr>
<tr>
<td>Command and control</td>
<td>Low coupling, high cohesion</td>
</tr>
</tbody>
</table>

Image source: strategic.mit.edu
<table>
<thead>
<tr>
<th>Hierarchical</th>
<th>Modular</th>
</tr>
</thead>
<tbody>
<tr>
<td>Enforce policies ✓</td>
<td>Developers form strongly connected communities</td>
</tr>
<tr>
<td>Flexibility ✗</td>
<td>Low coupling, high cohesion</td>
</tr>
</tbody>
</table>

**Long-Term Implications**

- Easy to establish favourable processes
- Support or friction can come from upper hierarchy levels
<table>
<thead>
<tr>
<th>Hierarchical</th>
<th>Modular</th>
</tr>
</thead>
<tbody>
<tr>
<td>Enforce policies ✓</td>
<td>Focus on deeply specialised issues ✓</td>
</tr>
<tr>
<td>Flexibility ✗</td>
<td>Friction at boundaries ✗</td>
</tr>
</tbody>
</table>

**Long-Term Implications**

Hierarchical:
- Easy to establish favourable processes
- Support or friction can come from upper hierarchy levels

Modular:
- Maintain specific portions long-term
- Little interference from unrelated code

Image source: strategic.mit.edu
Community Dynamics I

LLVM (typical) vs. Node.js (untypical)

- **LLVM**
  - Developer Count vs. Time across 2003-2015
  - Gini Coefficient vs. Time across 2003-2015
  - Not Scale Free vs. Scale Free

- **Node.js**
  - Developer Count vs. Time across 2011-2015
  - Gini Coefficient vs. Time across 2011-2015
  - Not Scale Free vs. Scale Free

W. Mauerer
ELC San Diego
Mar 17, 2016
Three typical phases

1. High coordination equality, slow growth, hierarchical structure
2. Superlinear growth of developer count, transition to scale freedom
3. Stabilisation of scale freedom — hierarchy (core dev), heterarchy (peripheral devs)

Implications on long-term maintenance

- Establishing structured processes should be done in phase 1 or 3
- Focus LT support efforts differently depending on project phase
- Speeding up transition from 1 to 3
Identifying Key Persons

- Official project structure: maintainers, sub-maintainers, …
- *Unofficial* social order: Who’s effectively in charge?
  - Yes, you may know that for project 〈x〉
  - But how about 〈y〉, 〈z〉, 〈ζ〉, 〈ξ〉, …?
Developer Classification

- **Core**: Developer with connectivity in 80% quantile
- **Peripheral**: Non-Core-Developer with connectivity $> 0$
- **Isolated**: Developer with connectivity $= 0$
- **Absent**: Developer without commits

Markov Chain

- Transition graphs: MaxLike
- Window size: 3 months (quasi stationary)
Chromium

Peripheral

74%

Core

9%
29%
17%

Absent

51%

Isolated

35%

14%

Observations

▶ Strong transfer from core to peripheral
▶ Good integration of isolated developers
▶ High loss rate (isolated to absent)
Chromium

Peripheral

Core

Absent

Isolated

Observations

- Strong transfer from core to peripheral
  - Intermittent efforts? Lack of commitment?
- Good integration of isolated developers
  - Implicit review. Chance to »label«/classify patches regarding back-porting etc.
- High loss rate (isolated to absent)
  - Loss of know-how + responsibilities, bit rot (+ watch out for NSA!).
Observations

- Very stable set of core developers
- Stagnation of peripheral and isolated developers
GCC

Peripheral
87%

Core
82%

Absent
18%

Isolated
29%

Observations

- Very stable set of core developers
  - Established expert code basis; long-term planning possible

- Stagnation of peripheral and isolated developers
  - Potential review bottleneck, uncoordinated structural changes, backport issues
Observations

- Strong transfer from core to peripheral
- High loss of peripheral and isolated developers
- Bad integration of isolated developers
MongoDB

Peripheral

Core

Absent

Isolated

66%

10%

31%

22%

41%

33%

69%

Observations

- Strong transfer from core to peripheral
  ⚠️ Intermittent efforts? Lack of commitment?

- High loss of peripheral and isolated developers
  ⚠️ One time contributions, easily miss important LT fixes!

- Bad integration of isolated developers
  ⚠️ Loss of know-how, bit rot (+ watch out for NSA!). Easily miss important LT fixes!
Questions? Questions!

- **Code**: https://github.com/siemens/codeface
- **Homepage**: https://siemens.github.io/codeface

- WM, M. Jäger: *Open source engineering processes*, IT special issue 55, 2013