Socio-Technical Aspects of Long Term Embedded Systems Maintenance

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.
Where is Regensburg?
Where is Regensburg?
Where is Regensburg?
Curriculum Vitae

- Technical University of Applied Sciences Regensburg
- Siemens AG, Corporate Research and Technologies
- Max Planck Institute for the Science of Light
- University of Erlangen-Nuremberg

Projects
### 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

That’s not physics!
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 Large Scale SW Development

▶ Nuclear physics is a team effort
▶ …and so is SW development
▶ Optimal cooperation structure?
▶ OSS: Matching of communities?
Examples from Non-Physics 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 influence software quality
- Lattice QCD does not describe human behaviour, sociology required
Examples from Non-Physics 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 influence software quality
- Lattice QCD does not describe human behaviour; 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
**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 <mauerer@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 mauerer@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) {
      
       Changed lines
Network construction

- Tagging (“Signed-off-by”)
- Committer/author
- Overlapping code contributions
- Feature co-changes
Software and Communities II
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 1.5.0 (May 2013) Virtualisation/Machine Emulation
Community Examples II

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

- 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)

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
- Handling team dynamics
### Network categorisation I

#### Random
- Randomly (iid) distributed edges (connections)
- Erdős-Rényi model: "typical" nodes (developers)
- Hub nodes: extremely rare

![Random network diagram]

#### Scale free
- No "typical" nodes (developers)
- Hub nodes: frequent
- Large real-world networks (biology, sociologie, internet routers, ...)
- Robust against *random* changes

![Scale free network diagram]
Network categorisation I

Random

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

Scale free

- No »typical« nodes (developers)
- Hub nodes: frequent
- Large real-world networks (biology, sociologie, internet routers, …)
- Robust against random changes
Random

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

Scale free

- Maintainer: Architectural (structural) knowledge ✓
- Hub dev hit by proton beam ☞ structural problems ❌
Hierarchical
- Developers: hierarchical layers
- Command and control

Modular
- Developers form strongly connected communities
- Low coupling, high cohesion
<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>

![Network Categorisation II](strategic.mit.edu)
<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>

Image source: strategic.mit.edu
LLVM (typical)

Node.js (untypical)
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)
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

Core

Absent

Isolated

Observations

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

Observations

- Strong transfer from core to peripheral
  - Intermittent efforts? Lack of commitment?
- Good integration of isolated developers
  - Implicit review
- High loss rate (isolated to absent)
  - Loss of know-how, bit rot (+ watch out for NSA!)
GCC

87% Peripheral
5% Core
18% Absent
6% Isolated

82% Core
29% Absent
16% Isolated
56% Peripheral

Observations

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

W. Mauerer
ELC San Diego
Mar 17, 2016
Observations

- Very stable set of core developers
  - Established expert codebasis
- Stagnation of peripheral and isolated developers
  - Potential review bottleneck, innovation stagnation
Observations

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

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

- High loss of peripheral and isolated developers
  
  ❁ One time contributions

- Bad integration of isolated developers
  
  ❁ Loss of know-how, bit rot (+ watch out for NSA!)
Questions? Questions!

- **Code**: [https://github.com/siemens/codeface](https://github.com/siemens/codeface)
- **Homepage**: [https://siemens.github.io/codeface](https://siemens.github.io/codeface)

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