We need more coverage, stat!
Classroom experience with the Software ICU

Philip Johnson, Shaoxuan Zhang
University of Hawaii

Presentation by Sandro Heinzelmann
Software Engineering Seminar 2010
April 27, 2010
Teaching software measurement - Motivation

• Not easy
• Tradeoff between too much work and too little insight
• Personal Software Process (PSP) / Team Software Process (TSP) versus simple literature review
• → Find a balance by using automation tools
Hackystat

- Opensource project initiated by Philip Johnson
- Collection of services
- Enables subtle, unobtrusive data collection in various development tools (Eclipse, Ant,...)
- Notion of sensors integrated in applications
  - Keep track of work, send data to Hackystat SensorBase
- Layer of analysis modules
- Webinterface to display data

code.google.com/p/hackystat/
Hackystat in the past

- Continuously improved over time
- Used in case studies in 2003 and 2006 with varying success
- Hard to install, confusion about various measurements and interpretation

- New approach with a medical metaphor
  Software Intensive Care Unit (ICU)
Software health metaphor

- Terminology of „health“
- Not new - „runtime health“ of life-critical hardware-software systems (NASA)
- Here focus is on health during development
- Notion of vital signs and their normal ranges
  - Normal or improving → healthy
  - Interpreted as a whole
Software health metaphor

• High-level characteristics of a healthy project
  – High efficiency, high effectiveness, high quality
• „Healthy programmer behavior“
  – Work consistently, contribute equally, consistent committing, no last minute rushes, ...
Vital signs

- Coverage
- Complexity
- Coupling
- Churn
- Builds

- Commits
- Unit tests
- Size
- Dev time

→ Research hypotheses
Vital sign interpretation

- Normal ranges and coloring defined by current value as well as trends
- Thresholds and methods can be configured

**Coverage**  
high or increasing

**Dev time**  
$\geq 50\%$ of the members commit, commits on $\geq 50\%$ of the days in the project interval

**Size**  
No interpretation (color white)
ICU display

- Current value as well as trend lines

code.google.com/p/hackystat/
Drill-downs

- Detailed, per-member view of vital signs
Research questions

• What are the strengths and weaknesses of the medical ICU metaphor for teaching software measurement in a classroom setting?
• How appropriate were the choices of “vital signs”?
• How effective were the algorithms for coloring the vital signs?
• How does this approach compare to previous uses of Hackystat to teach software metrics in a classroom setting?
Study setting

- 18 students in a senior-level undergraduate software engineering course
- Course about open source development in Java
- ICU introduced in the **final 4 weeks**
- Hackystat log data
- Online survey during the last week, 17 questions
  - Installation overhead
  - Overhead of sensor use
  - Problems encountered during use
  - Frequency of use
  - Privacy
  - Useful vital signs
  - Usefulness in an industrial setting
Results - misc

• Privacy: mixed, but generally positive feelings (from no problem to „hacky-stalk“)
• Overhead: easier than in earlier versions, though varying from tool to tool. Sensor sending sometimes slow.
• Frequency of use
• Coloring generally seen as accurate, with some general drawbacks
• ICU and drill-downs in particular useful to react to poor health and manage team
Results – industrial possibilities

• Generally considered a good idea
• But
  – does not include non-IDE work (like reading a technical book)
  – Algorithms can never fully judge the health of a program in all contexts
Discussion and conclusions

• Significantly better results than previous Hackystat studies
• ICU metaphor is useful to interpret and understand measurements
  – No more „pretty squiggly lines“
  – Coloring encourages thoughts about validity
• ICU provides a layer of abstraction
  – Normal ranges must be chosen carefully!
  – Too lenient interpretation leads to oversight
  – Too strict interpretation leads to „boy who cried wolf“ syndrome
• Vital sign ranges need to be tweaked further
• Dangerous weakness: measurement dysfunction
Measurement dysfunction

*Using measures competitively as a means to do good at a performance evaluation*

- Individual measurements did not contribute to the grade
- Data was only visible to the assistant, professor only had anonymized data and got to see survey only after semester
- And yet: At least one group had major problems
  - \textit{“I need more dev time because I need an A“}
  - \textit{“oh if he ups his stats more than mine, tomorrow I’m gonna hack all day“}
- \(\rightarrow\) compromised work as a team
Threats to validity

In the paper

- Small sample size (😭)
- Small duration, Small project size
- Subjects with very similar background (senior computer science students)
- Wrong demography for „industry“ questions

Personal

- Relatively short survey
- Students unfamiliar with software measurement
Future directions

• Refine vital signs and ranges
  – More research
  – „crowd-sourcing“
• Use in more environments
  – Industry
  – Different project types/languages/IDEs
• Game-based approach
  – „Dev-cathlon“

Personal
• Comparative studies versus other measurement techniques (PSP/TSP)
Questions?

stormgrounds.com
Appendix - Hudson

- **Continuous integration tool** developed by Kohsuke Kawaguchi
- Builds and tests projects after every commit
- Used in the following application for measurements of coverage, coupling, and complexity
Appendix: PSP

- „Disciplined, data-driven procedure“
- Level-based approach: PSP0 to PSP2.1
- Use „historical“ data/experience from previous level to detect repeated defects
- Requires programmers to log their activities (a lot of manual data collection required, even with tool support)
- Many measures collected and derived: estimation accuracy (size/time), prediction intervals (size/time), time in phase distribution, defect injection distribution, defect removal distribution, productivity, reuse percentage, cost performance index, planned value, earned value, etc etc
Appendix: Complexity

- Authors hint at 2 methods: Halstead complexity measures & McCabe’s cyclomatic complexity
- Judging from the configuration site, ICU uses JavaNCSS, which uses the cyclomatic complexity:
  - Uses flow graph of program
  - Counts number of independent paths through program (Base Path Testing)
  - \( M = E - N + 2P \)
  
  Where:
  - \( M \) = cyclomatic complexity
  - \( E \) = number of edges of the graph
  - \( N \) = number of nodes of the graph
  - \( P \) = number of connected components