Seminar “Ausgewählte Beiträge zum Software Engineering”

Tools for Capturing Micro-Process Data

Sebastian Jekutsch

Freie Universität Berlin
Institut für Informatik
Arbeitsgruppe Software Engineering
Talk overview

1. Utilizing micro-processes for defect prevention
2. Tools for capturing micro-process data
   1. Hackystat
   2. PROM
   3. Ginger2
3. Research tasks
1. **Utilizing micro-processes for defect prevention**

2. **Tools for capturing micro-process data**
   1. Hackystat
   2. PROM
   3. Ginger2

3. **Research tasks**
Micro-process: Definitions

- “Micro-process” in software development is the process view (i.e. series of events) of actions and activities (i.e. the events) taken by a software developer or developer group while performing a specific sub-task.
  - Work psychology: “Activity analysis”
  - First we focus on coding as a sub-task for a single developer
- “Capturing micro-process data” means taking a log of time-stamped events
- “Episode” is an abstraction of a micro-process’ time interval which forms a typical series of events
  - Similar to “Pattern”
- “Situation” are conditions of the task and environment
- “Belief” is a worker’s knowledge about domain and action
Coding micro-process: Examples

• Events
  ▪ typing, executing code, browsing doc., saving file
  ▪ picking up phone, conversation, going to lunch, pausing

• Episodes
  ▪ trial-and-error, copy-paste-change
  ▪ resuming work after interruption, stack of working tasks

• Situation
  ▪ task description, previous events, used tools, noise
  ▪ workload, stress, tiredness, intelligence, experience

• Beliefs
  ▪ language semantics, design decisions, library usage, requirements, used quality criteria, division of labour
Utilizing micro-process for defect prevention

- “Defect”: Structural property of a software document that makes it (locally) incorrect
  - ?? Any part of a software document which has later been changed, added or removed permanently ??
- “Defect insertion” is the action of creating defective code
  - Hypothesis: An (episode, situation, belief)-triple is associated
  - The defect has a type (taken from a defect taxonomy)

- Research target: To find typical defect insertion triples
  - Correlation (e,s,b)-triple to defect type
  - ... or to present a reason why there are none
  - in general: understanding defect insertion better
Other opportunities while collecting m.p. data

• When X was an defect insertion, ~X may be also
• Tracking evolution of code copies
• “Macro-fying” work episodes
• Suggest places to look at because of past browsing sessions
• Aid for empirical research on psychology of programming
• Re-examining past coding sessions
  • summary possible?
  • learning about personal bad practices
  • learning from colleagues
• Evaluating new micro-process metrics
  • e.g. discriminating novices from experts
Capturing micro-process data

- Capturing micro-process data focuses on the episode part of the triple.
- Possible data sources are
  - programming environments
  - work environment devices (e.g. phone)
  - other indicators of what the programmer is actually doing
- We need a tool which should be
  - able to collect interesting events
  - ... automatically (via instrumentation of data sources)
  - able to compile episodes
  - non-disruptive
  - usable in realistic scenarios
  - extensible to a variety of data sources
Talk overview

1. Utilizing micro-processes for defect prevention
2. Tools for capturing micro-process data
   1. Hackystat
   2. PROM
   3. Ginger2
3. Research tasks
Hackystat – Its origin

• Prof. Johnson at University of Hawaii

• Started with some PSP tools for easy logging of defect and effort data

• “You can’t even ask them to push a button”
  ▪ Automatic collection is a must

• Developing Hackystat since ?, available since 2001.

• New story: Software Project Telemetry
Hackystat architecture

- Mailer
- Emacs
- Ant
- CVS
- Eclipse
- Browser

Analysis results/URLs

Emacs Sensor
FileMetric
Build
Commit
Activity

Ant Sensor

CVS Sensor

Eclipse Sensor

Hackystat Web Server

Raw sensor data is sent via SOAP

XML Database

Generating new data

(Telemetry Display Language)

Telemetry data and drill downs
Hackystat Web interface
Hackystat: Telemetry Control Center
Hackystat: Telemetry – what for?

- Telemetry: “communications process for measuring, monitoring and recording using data collected at inaccessible points”
- Software Project Telemetry requirements
  - automatically
  - stream of time-stamped events
  - immediately available
  - even if measurement started midway through a project
  - in-process monitoring, *in vivo*
- Development telemetry
- Build telemetry
- Execution telemetry, Usage telemetry
  - not our focus
Hackystat: Sensors and Events

- Sensors are “plug-ins” to send data to Hackystat server
  - tool specific
  - data specific
- Emacs, Visual Studio, JBuilder, Eclipse
  - Activity, BufferTransitions, FileMetric
  - Build, UnitTest
- Excel, Word, Powerpoint, Frontpage
  - Activity, BufferTransitions, FileMetric
- Command line
- JUnit
  - UnitTest
- Ant, CVS, Bugzilla
  - Commit, Defect, Build
Hackystat: Events in Eclipse

- Eclipse-Events (as of version June 2004)
  - Project open/close
  - (Java) file open/close/save/activate/change (with file metrics)
  - Breakpoint add/remove
  - Compiler errors
  - Class add/delete/move
  - Method and Attribute add/delete/move
  - Import add/remove
  - JUnit-Run failures/errors
  - Runtime failures
- No code change analysis
- Local buffering of events
Hackystat: Metrics

• Derived metrics:
  ▪ Usage times of a tool
  ▪ Working time of developer
  ▪ Size/lines of files
  ▪ Build attempts per project
  ▪ Number of failures per module
  ▪ Coding time per module
  ▪ Defect frequency per module
  ▪ Complexity of module
  ▪ etc.

• module = set of files
Hackystat: Design

- Completely Java-based (apart from some sensors)
  - CVS, Ant, JUnit, HttpUnit, JSP, Tomcat, JDOM, Cruise Control, JFreeChart
  - No database, XML data plain file.

- Open-Source
- Modular Ant build process
- Lots of unit tests
- Easy installation (apart from sensors, SOAP setup)

- In 5th architectural revision

- Funded by: Sun, IBM, NSF, NASA
Hackystat: Can we use it?

• Events are not fine-grained enough
  ▪ Events are file operation based (open, save)
  ▪ We probably need code changes without file change

• Server and Communication *reusable*
  ▪ No database: scalability?
  ▪ XML representation just fine?

• Sensor data types *reusable* as well

• Different analysis tools necessary
  ▪ Events analysis
  ▪ Episode analysis
  ▪ Defect detection
Talk overview

1. Utilizing micro-processes for defect prevention
2. Tools for capturing micro-process data
   1. Hackystat
   2. PROM
   3. Ginger2
3. Research tasks
PROM: Its Origin

- PROM = PRO Metrics
- Universities Bozen and Genova, Italien
- Also PSP-based research, mainly time estimation

- Only little information, still alive?
PROM: Architecture

![Diagram showing the architecture of PROM with core, tool-specific plug-ins, and third-party tools connected to a server and client.]

Sebastian Jekutsch, jekutsch @ inf.fu-berlin.de

22 / 36
PROM: Plug-ins and metrics

• Plug-ins
  ▪ NetBeans, Eclipse, JBuilder, Visual Studio
  ▪ Together, Rational Rose
  ▪ MS Office, OpenOffice

• Events/Metrics
  ▪ Users logged in
  ▪ Project name
  ▪ Class name
  ▪ File opening
  ▪ Focus time

• More information not available
PROM and Hackystat

- PROM claims to be more general than Hackystat...
- Manual data insertion possible
- Plug-in server
- Database
- No Open-Source (?)

⇒ Nothing (real) new
Talk overview

1. Utilizing micro-processes for defect prevention
2. Tools for capturing micro-process data
   1. Hackystat
   2. PROM
   3. Ginger2
3. Research tasks
Ginger2: Its Origin

- Nara Institute of Science and Technology, Japan
- Prof. Torii, Dr. Mondon

- Following based on TSE paper 1999
- No (readable) information on web site

- CAESE: Computer Aided Empirical Software Engineering
  - repeatable experiments
  - computer based data collection
  - automated data analysis

- Laboratory environment for *in vitro* studies
Ginger2: Experimental environment

Ginger2 measures:
- eye tracks
- skin resistance
- motion
- audio
- video
- typed keys
- tool usage
- file changes
By the way: Pupillometry

- Size of worker’s pupil indicates mental workload
  - stress
  - difficult tasks
  - etc

- Not used by Ginger2
  but used by Center for Media Research, FU Berlin
  - to control computer-based learning sessions
Ginger2: Data display

h hawk
:739713873
w 0x380000c
Wo 400x200+100+100
Ts 20 14
T hawk%\_0 0
C 6 0
:1.5
K cd\_/etc\r
T cd\_/etc
T hawk%\_0 1
C 6 1
:3
K \_r

(Ginger1 log)
Ginger2: Case studies 1-3

• Case study: Debugging process of experts and novices
  ▪ used eye tracking and audio/video
  ▪ Findings:
    • Experts focus on one or two modules faster
    • Novices shift their gaze points rapidly

• Case study: Understanding two-person debugging
  ▪ used audio/video and terminal logging
  ▪ different types of communication observed
  ▪ Findings:
    • asynchronous communication (chat, mail) is more effective than synchronous (verbal) communication
    • division of work is effective: one “understander”, one “locater”: uni-directional communication

• Case study: Evaluation of user interfaces
Ginger2: Case study 4 (1)

- “Analysis of programmer’s behavior when creating bugs”
- Personal communication

- Used Emacs logging, audio/video
- Planned: eye tracking
- Defect detection via unit tests

- 3 subjects, 42 defect insertion
- Subjects coded unit tests as well
Ginger2: Case study 4 (2)

- 6 patterns extracted (written as a grammar)
  1. Copy-Paste-Change
  2. Badly resuming work
  3. Changing a line again and again
  4. Overseeing the second of two defects in one line
  5. Writing a line for a long time
  6. Copying a defective line
- Half of the defects can be described with any one of the patterns
- Overall: 1% probability of defect per line
- Pattern 1 observed => 7.5% pb. of defect insertion
- Pattern 5 observed => 10.5% p.o.d.i.
Ginger2: Summary

- Disruptive environment
- Not “off the shelf”
- No ongoing work (?)

- The only micro-process analysis towards defect prevention I’m aware of
- Non-promising initial results
Talk overview

1. Utilizing micro-processes for defect prevention
2. Tools for capturing micro-process data
   1. Hackystat
   2. PROM
   3. Ginger2
3. Research tasks
Micro-process research tasks

- Learning about programming models
- Defining a set of interesting events and episodes
  - mainly exploratory work
  - grammar just like in Ginger2
  - episode extractor
- Developing an Eclipse plug-in to capture micro-process
  - reusing Hackystat as a server
- Establishing ways to isolate defects
  - micro-process changes, analysing code rev., bug report
- Capturing data (a lot)
- Analysing the data
- Investigating situations and beliefs
  - using psychologist’s research on human error
Thank you!