veritas
iustitia
libertas
Freie Universität Berlin

Seminar “Ausgewählte Beiträge zum Software Engineering”

Psychology of Programming 1 – History and some selected works on the Dynamics of Coding

Sebastian Jekutsch

Freie Universität Berlin
Institut für Informatik
Arbeitsgruppe Software Engineering

Talk outline

1. Why is “Psychology of Programming” of any interest?
2. History of PoP (incl. some insights)
3. Selected works on the Dynamics of Coding
   a) Focal expansion (Rist 1989)
   b) Parsing – Gnisrap (Green et.al. 1987)
   c) Knowledge restructuring (Davis 1989)
   d) Change episodes (Gray, Anderson 1987)
4. References, Conferences
5. What’s in it for the micro-process of software development?

Why “Psychology of Programming (*PoP*)” ?

- Software development is a human activity
  - … and mainly brain activity
  - Cognitive Psychology = “mechanics” of the brain
- Software Engineering
  - Better development environments
    - programming languages, modeling techniques
    - editing environments
  - Better management
    - e.g. social issues
  - Better teaching
  - Better methodology and practice?
- Cognitive Psychology
  - Studying problem solving and learning
  - Programming is continuous talking about planning

PoP history 1/6: Intuitive phase

- First attempt was done by computer scientists
- Goto considered harmful (Dijkstra 1968)
  Structured programming (Dahl et.al. 1972)
  - intuitive, psychological reasoning
  - but no theoretical background
- Weinberg 1971 “The Psychology of Comp. Programming”
  - demanded psychological view
  - focused mainly on team work
  - still no theoretical background
- This (and others) caused empirical studies
  - Prg. languages, prg. style, notations, etc.
  - naive experimental design
  - no theoretical background ⇒ unfocussed experiments

PoP history 2/6: No theoretical background

- Example: Benefit of meaningful variable names (*mvn*)
  - Repeating program fragments with and without *mvn*
    - one experiment found weak support in favor of *mvn*
    - another one found no support
  - Psychological point of view:
    - *mvn* must be a semantic index to schemas (or 'chunks')
    - schemas are long-term memory of typical patterns
      - just like in chess
    - Experiment: Identify missing statements, e.g. `Count := 0` in a loop with remaining statement `Count = Count + 1`
    - with *mvn* much faster than without *mvn*
- Example: Flowcharts help in mental program execution but not in program understanding
  - *PoP* examined these two strategies while debugging

PoP history 3/6: Badly used theory

- Example: Program complexity
  - Short-term memory (*stm*) 7±2 items
  - McCabe and Halstead complexity based on this assumption
  - ⇒ The more branches in code, the more complex the code
  - But matched schemas are stored as *one* item
    - e.g. counting loop is one item, not two
  - ... available building rules as well
    - if-else cascade as a case statement

PoP history 4/6: Psychologist's models

- Shneiderman 1980 "Software Psychology"
  - Mix of experimental findings
  - and guesses
- Psychologists started with *PoP* in early 80ies
- Observing programming as complicated problem solving
  - Coding, Debugging, Comprehension
  - Novices versus Experts
- Applying cognitive models on programming tasks
  - problem solving, narrative text comprehension
- Building supportive programming environments
  - structured editors, macros, better languages
  - often based on Lisp, Prolog, Basic, artificial languages

PoP history 5/6: Mixing experts and novices

- Teaching idea: Novices should imitate experts
- Observation: Experts do their work top-down
  - Pascal and Top-Down-Development invented
  - Bad performance sometimes even for experts
  - Its too restrictive! (Experts are not experts every time.)
- Being an expert is knowledge, not a habit!
  - Top-Down-Development is a symptom

- Question: Do design patterns help novices in construction?

PoP history 6/6: Two communities

- Still two communities side by side
- Software Engineering: No interests in explanations, experiments only for validating new methods
- Psychology: Only small enhancements, deduction from theory, experimental validation
  - By the way: Bad external validity in *PoP*
    - That's not unusual for experiments in psychology
    - … but bad practice in Software Engineering
- Software Engineering is inventing,
  Psychology sometimes explains afterwards
  - e.g. Object-based: Technically invented, intuitively applied
  - Psychology on Object-based: People tend to think in data-flow rather than control-flow.

Talk outline

1. Why is “Psychology of Programming” of any interest?
2. History (incl. some insights)
3. **Selected works on the Dynamics of Coding**
   a) **Focal expansion (Rist 1989)**
   b) **Parsing – Gnisrap (Green et.al. 1987)**
   c) **Knowledge restructuring (Davis 1989)**
   d) **Change episodes (Gray, Anderson 1987)**
4. References, Community
5. What’s in it for the micro-process of software development?

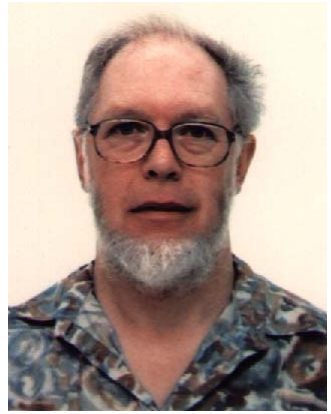
Cognitive “Software”

- General idea often used: Programming as Planning
- Long-term memory (*ltm*) contains production rules
- Conditions are entries in *stm*
- Actions are:
  - changing/adding/removing/reordering *stm* entries
    - which are often links into *ltm*
  - changing knowledge about program (in *ltm)*
  - changing or writing out code (to external memory (*em*))
- Order of rules is changing
  - e.g. according to latest invocations
- Rules are learned
  - rules are problem specific: expertise is problem dependant
  - learning = adding and generalization of rules

a) Focal expansion

- Coding dynamics = evolution of *em*, especially code
- Rist (1989) refined production model, based on observation:
  - plan creation: 1. Calculation, 2. Initialising, 3. Output
    - backwards dynamics, bottom-up
    - “focal expansion”
  - plan retrieval: 1. Initialising, 2. Calculation, 3. Output
    - forward dynamics, top-down
    - “schema expansion”
  - Expansion type depends on knowledge/expertise
    - forward / top-down / schema expansion being expert-like
- Based on small examples only

b) Parsing – Gnisrap 1/2



- Green et.al. (1987) in response to Rist's focal expansion
  - Research on adequacy of programming languages
- Any non-linear and non-top-down progress is an indicator for a problem
  - seems that programmer lacks a suitable schema
  - chaotic programming rises workload (*stm* "full")
- Programming is performed as follows
  1. Making a plan mentally
  2. Gnisrap: As soon as it becomes to complex, it is written down. "Releasing" of *stm*.
  3. Parsing: As soon as programmer needs to expand own written code, it is read = parsed = reconstructing the plan.

b) Parsing – Gnisrap 2/2

- Usually, programmers write down many focal lines up front, leaving unimportant parts for later refinement.
  - limitation of *stm* is the problem
  - leaving these parts uncoded is typical source of error
- Schemas have pre- and post conditions which need to be fulfilled. In first attempt, these are often not checked
  - another source of error
- Too restrictive languages and editors don't allow to "release" *stm*, i.e. do Gnisrap
  - and too small a window doesn't allow to do Parsing
  - both don't allow expert's linear dynamics.
- There's no simple way to judge experts based on the dynamics of coding.

c) Knowledge restructuring 1/2

- Davis (1989) investigated relationship between linear dynamics and expertise. Follow-up to “Gnisrap”.
- Observation: Experts write down focal lines more often and more early than novices.
- Schemas *have* focal lines (besides pre/post cond.)
- Schemas are written down in linear fashion with focal up front
  - that’s *not* the surface (code) linearity.
  - novices have only surface knowledge.
    - Experts can easily switch between programming languages
- Becoming expert mean “knowledge restructuring”
  - Building schemas, i.e. patterns/idioms
  - Enriching them with pre/post/focal points

c) Knowledge restructuring 2/2

- Experiment:
  - Presenting small program to novices and experts to remember them
  - Presenting single lines afterwards.
  - Q: Have they been part of the original program?
    - Many of them were alike, but not exactly present in orig.
  - Performance of novices and experts didn't differ much
  - But experts recognized focal lines much faster

d) Change episodes 1/2

- Gray/Anderson (1987) investigated Parsing-Gnisrap cycles
- Programming is planning:
  - Progressive activities: looking for operator and applying it, i.e. writing down code
  - Evaluative activities: checking operator and changing it = change episode
    - error correction
    - stylistic changes
    - tactical changes
    - (that's *not* performed after dynamic testing)
- Q: When and where do these change episodes take place?

d) Change episodes 2/2

- Change episode = (place, noticing event, changed code)
- Findings:
  - Changes episodes occur at places which required unusual amount of planning, i.e. not simply writing down schema
    - because there is simply no schema
    - because there has been more than one schema and the first chosen was not correct
    - (How do they recognize?)
  - Sorts of change episode start event
    - Interrupt to coding (within 2 sec): Filling focal line skeleton
    - Tag-along (within 4 sec): Doing changes along the way
    - Symbolic exec. (within 52 sec): Mental “execution” of code
  - Sorts of changed code: Two sizes (sub-plan and tactic)
- Method: Mainly vocal protocol

References, Conferences, Publications

- Simon P. Davis. *Models and theories of programming strategy.* Int. Journal Man-Machine Studies, 1993
- Hoc et.al (Hrsg.). *Psychology of programming*. 1990
- Francoise Detienne. *Software Design - cognitive aspects*. Springer 2002
- B. A. Sheil: *The Psychological Study of Programming*. ACM Computing Surveys, 1981

- *Psychology of Programming Interest Group* in Europe
- *Empirical Studies of Programmers* in U.S.A.

- Int. Journal on Human-Computer Studies
- Annual and Bi-Annual Workshops

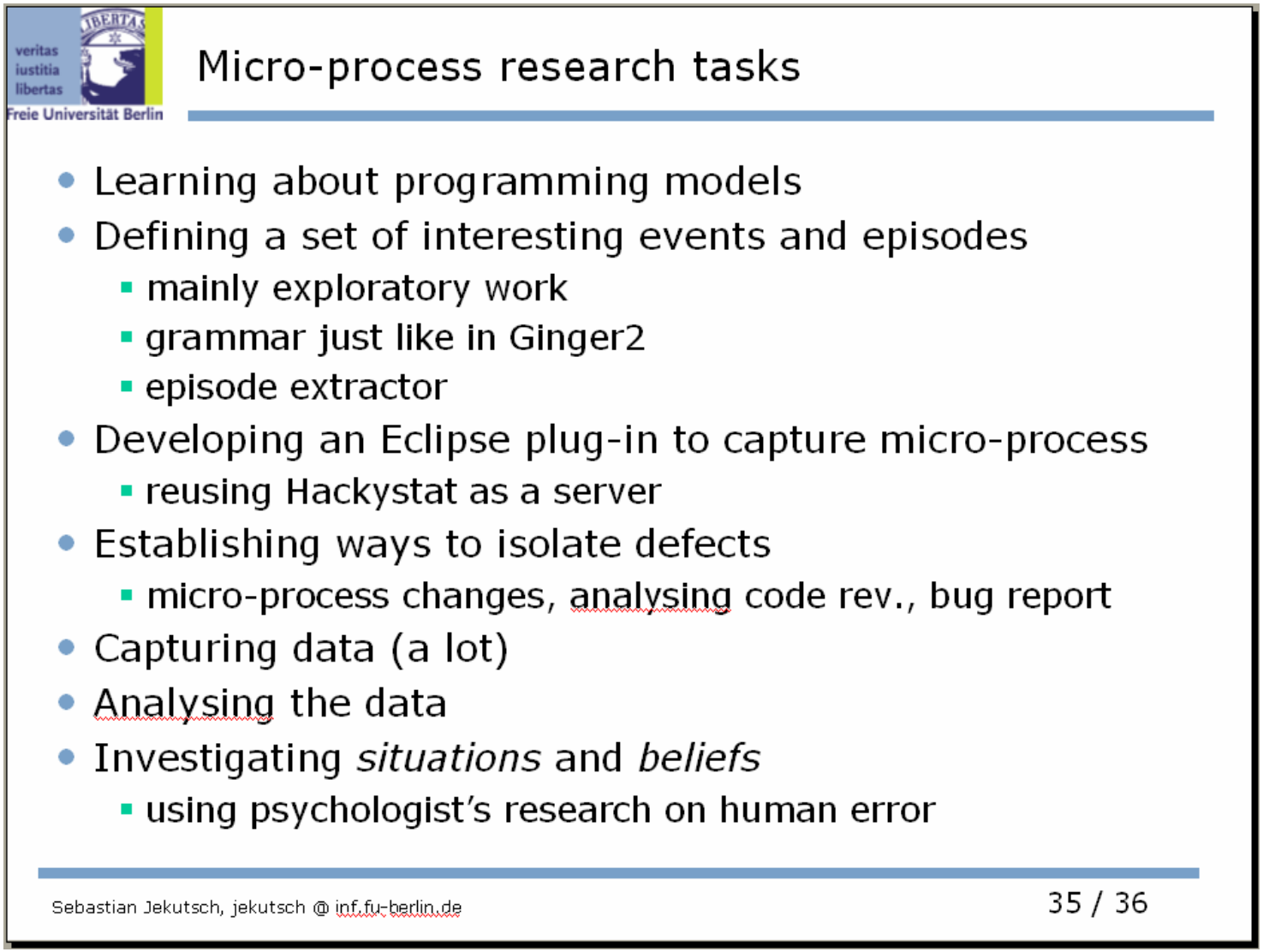
Talk outline

1. Why is “Psychology of Programming” of any interest?
2. History (incl. some insights)
3. Selected works on the Dynamics of Coding
   a) Focal expansion (Rist 1989)
   b) Parsing – Gnisrap (Green et.al. 1987)
   c) Knowledge restructuring (Davis 1989)
   d) Change episodes (Gray, Anderson 1987)
4. References, Community
5. **What’s in it for the micro-process of software development?**

What's in it for micro-process study? 1/2

- We saw *PoP* work on coding dynamics = micro-process
- Topics:
  - How are programs constructed?
  - Behavior of experts compared to novices
- From time to time, every programmer acts like a novice.
- Acting as a novice may point to making errors
- Recognizing novice's behavior using concepts on
  - non-linear progression, bottom-up construction
  - change episodes, etc.
- i.e. "bad" episodes

What's in it for micro-process study? 2/2



Thank you!