Course "Spezielle Themen der Softwaretechnik"

Agile Methods: Pair Programming (PP)

Lutz Prechelt
Freie Universität Berlin, Institut für Informatik
http://www.inf.fu-berlin.de/inst/ag-se/

- Definition
  - Related methods
- An example study of PP
- Questions and empirical results:
  - Relative raw productivity?
  - Quality of resulting design and code?
- Benefits from more people being familiar with code?
  - ... from interruptability?
  - Influence on motivation?
  - Recommendations for learning pairing, changing?
- Our PP research approach
  - data gathering, evaluation
Learning objectives

• Understand which factors influence PP
• Understand the potential benefits from PP
• Get an overview of the current state of knowledge about these benefits
  • and some deficiencies of the studies so far
• Understand why they are difficult to measure
• Understand the research approach of AG Software Engineering with respect to understanding PP
Definition: Pair Programming

From [WilKesCun00]
"In pair-programming,

- two programmers jointly produce one artifact (design, algorithm, code, etc.).
- The two programmers are like a coherent, intelligent organism working with one mind, responsible for every aspect of this artifact.
- One partner is the 'driver' and has control of the pencil/mouse/keyboard and is writing the design or code.
- The other person continuously and actively observes the work of the driver -- watching for defects, thinking of alternatives, looking up resources, and considering strategic implications of the work at hand.
- The roles of driver and observer are deliberately switched between the pair periodically.
- Both are equal, active participants in the process at all times and wholly share the ownership of the work products whether they be a morning's effort or an entire project."
Related methods

**Distributed Pair Programming (DPP):**
- The partners are not physically in the same room and use a separate computer each
  - [http://www.saros-project.org](http://www.saros-project.org) (Saros, see last few slides)
- Their interaction is supported by a collaborative editor and chat, audio conferencing, perhaps video.

**Side-by-side Programming (SbS [Cockburn05]):**
- A task is assigned to a pair of programmers
- Each programmer has his/her own computer
- Allows them to split the task into subtasks and work on each subtask individually, but in close coordination
- The partners are sitting close to each other in one room and interact directly
Related methods (2)

**Ping-Pong Pair Programming (PPPP):**
- PP with frequent role change, driven by Test-Driven Design
- Driver/observer role switching follows a fixed rule:
  - one partner writes a test,
  - the other the implementation,
  - then vice versa (in rather tiny increments)

**Distributed Party Programming (DPP):**
- work style like Side-by-Side programming
- but with more than 2 participants (say, 5)
- only possible with distribution and appropriate tool support
Ways towards understanding PP: Blackbox perspectives (quantitative)

**Independent variable**
(Prosecut collaboration)
- Individual programming
- Partner programming
- Pair programming
- Team collocation

**Dependent variables**
(Outcomes)
- Time
- Cost
- Quality
- Information and knowledge transfer
- Trust and morale
- Risk

**Context variables**

<table>
<thead>
<tr>
<th>Subject variables</th>
<th>Task Variables</th>
<th>Environmental variables</th>
</tr>
</thead>
<tbody>
<tr>
<td>- Education and experience</td>
<td>- Type of development activity</td>
<td>- Software development process</td>
</tr>
<tr>
<td>- Personality</td>
<td>- Type of task</td>
<td>- Software development tools</td>
</tr>
<tr>
<td>- Roles</td>
<td></td>
<td>- Work space facilities</td>
</tr>
<tr>
<td>- Communication</td>
<td></td>
<td></td>
</tr>
<tr>
<td>- Switching partners</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

*Figure 1: An initial framework for research on pair programming.*

source: [GalAriDyb03]
Studies on PP

- source: [HulAbr05]: "A Multiple Case Study on the Impact of Pair Programming on Product Quality"
[Nosek98] "The case of collaborative programming" (1)

Survey+experiment, compared pairs to individual programmers:

- "A ... field experiment ... using experienced programmers who worked on a challenging problem important to their organization, in their own environments, and with their own equipment."
  - 15 full-time system programmers from a program trading firm, developing a consistency-checking program for a Sybase DB; Unix, X-Windows, C

- Hypotheses:
  - "Programmers working in pairs will produce more readable and functional solutions"
    - READABILITY variable: 0 unreadable, 2 fully readable, 1 in between
    - FUNCTIONALITY variable: degree 0...6 of achievement of objectives
  - "Groups will take less time on average to solve the problem"
  - "Programmers working in pairs will express higher levels of confidence about their work (CONFID) and enjoyment of the process (ENJOY)"
[Nosek98] "The case of collaborative programming" (2)

- 15 programmers randomly assigned into
  - 5 pairs
  - 5 individual programmers
- All solve the same task: DBCC (database consistency check)
- Worktime limited to 45 minutes

- Afterwards, each person answered several questions regarding CONFID and ENJOY
  - The exact questions used are not indicated in the article
  - The scale used for the results is also not explained

- READABILITY and FUNCTIONALITY were each judged by two grader for each solution (and had over 90% agreement)
### [Nosek98] Results

**Comparison of Individual and Team Measurements**

<table>
<thead>
<tr>
<th>Variable</th>
<th>Control Group (Individuals) mean (st. dev.)</th>
<th>Experimental Group (Teams) mean (st. dev.)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Performance Scores:</td>
<td>n = 5</td>
<td>n = 5</td>
</tr>
<tr>
<td>READABILITY</td>
<td>1.40 (0.894)</td>
<td>2.00 (0.000)</td>
</tr>
<tr>
<td>FUNCTIONALITY SCOR</td>
<td>4.20 (1.788)</td>
<td>5.60 (0.547)*</td>
</tr>
<tr>
<td>SCORE</td>
<td>5.60 (2.607)</td>
<td>7.60 (0.547)*</td>
</tr>
<tr>
<td>TIME (minutes)</td>
<td>42.60 (3.361)</td>
<td>30.20 (1.923)</td>
</tr>
<tr>
<td>Satisfaction Measures:</td>
<td>n = 5</td>
<td>n = 10</td>
</tr>
<tr>
<td>CONFID</td>
<td>3.80 (2.049)</td>
<td>6.50 (0.500)*</td>
</tr>
<tr>
<td>ENJOY</td>
<td>4.00 (1.870)</td>
<td>6.60 (0.418)*</td>
</tr>
</tbody>
</table>

*indiv. take 41% longer, but difference is not significant according to two-sided t-test*

*less than 1 in 20 that results are due to chance*
[Nosek98] Conclusions

- Hypothesis 1 (pairs' solutions are more readable and more functional, \( \text{SCORE} = \text{READABILITY} + \text{FUNCTIONALITY} \)):
  - Is confirmed

- Hypothesis 2 (faster):
  - Is not confirmed, as the difference is not statistically significant

- Hypothesis 3 (CONFID, ENJOY):
  - Is considered confirmed
[Nosek98] Strengths & weaknesses

- **Good:**
  - "real" programmers
  - "real" task

- **Bad:** A lot of interesting information is missing in the short 4-page article:
  - Did the participants have any experience in PP?
  - Were they prepared for PP in some way?
  - What attitude did they have towards PP before the experiment?
  - To what degree was design relevant for solving the task?
  - How did the pairs work together (communication, role switching, etc.)?
  - Did the partners learn from each other during the PP session?

Questions about PP

- Raw productivity compared to two separate programmers?
  - Cost or time per functionality
- Quality of resulting design?
- Quality of resulting code?
- Benefits from more people being familiar with code?
- Benefits from interruptability?
- Influence on motivation?
- Calculation of the overall cost-benefit
- Learning process?
- Recommendations for pairing, changing, etc.?

Subsequent slides explain the questions and the respective state of knowledge.
PP: Raw productivity?

Processing time:

- **PP vs. solo**
  - Completion was 41% longer for individuals than for pairs, but not statistically supported.
    J. T. Nosek [Nosek98]
  - Pairs complete their assignments 40-50% more quickly.
    Laurie Williams, Robert R. Kessler, Ward Cunningham, and Ron Jeffries [WilKesCun00]

- **SbS vs. PP vs. solo**
  - SbS 60% and PP 75% of the time of solo.
    Jerzy R. Nawrocki, Micha Jasiński, Lukasz Olek, and Barbara Lange [NawJasOle05]
PP: Raw productivity?

Effort

- **PP vs. solo**
  - Pairs spent approximately only 15% more effort on a task than solo developers (Williams [Williams02])
  - 10% increase (Ciolkowski and Schlemmer [CioSch02])
  - 21% increase (Lui and Chan [LuiCha03])
  - New team members who were added to a delayed project require less assimilation and mentoring time and thus improve the productivity of the whole team (survey + case study, Williams, Shukla, Antón [WilShuAnt04])
  - Pair programming is less productive than "XP done by solo developers" (Nawrocki and Wojciechowski [NawWoj01])

- **SbS vs. PP vs. solo**
  - Overhead for side-by-side programming was as small as 20%, while for PP it was about 50% (Nawrocki, Jasinski, Olek, Lange [NawJasOle05])
PP: Quality of resulting design/code?

Defects

- Decreased defect rates [Williams01], [Jensen03], [Tomayko02]
- Fewer failures in automated test [WilKesCun00], [CocWil01]:
  - The programs produced by pairs had about 15% fewer failures according to the automated test cases run by the teaching staff
PP: Quality of resulting design/code?

- Functionality:
  - [Nosek98]: Higher degree to which pairs solved the problem

- Readability:
  - [Nosek98]: Improved readability from pairs

- Fewer lines of code:
  - [CocWil01]: "We believe this is an indication that the pairs had better designs."
PP: Quality of resulting design/code?

- **Design**
  - Pair programming improved productivity most in demanding design tasks (experiment, [LuiCha03])
    - design of study can hardly be taken serious as PP
    - participants performed intelligence tests
  - Pairing was not found useful in simple, rote tasks (case study, Matthias M. Müller and Walter F. Tichy [MulTic01])

- **Coding standards**
  - Coding standards are followed more accurately with the peer pressure to do so (experiment + interviews, [CocWil01])
Many projects have strong individual code ownership: For each code module, only one programmer understands it well and only that person makes all modifications

- and only this person can do so with usually no errors.
- This often hampers project progress when corrections need to be made by someone who is already overworked
  - or hamper quality if a complex module is often misused

PP will greatly reduce that problem

How big is this benefit in terms of progress and quality?

- No quantitative results are known, as this is immensely difficult to measure
  - It requires project-level observations
PP: Benefits from interruptability?

- Frequent interruptions of programmers' work is known to be problematic for productivity and quality
  - Lost time to get back into the problem again afterwards
  - High probability of committing an error
- Potentially in PP, productivity hardly breaks down when an interruption occurs
  - one person handles the interrupt, the other continues
- In interruption-rich situations, this might be a big advantage
  - depending on phase/activity (most useful during pure coding)
  - depending on who is driver (most useful if the observer can handle the interruption)
- But the effect is very difficult to measure quantitatively
- No quantitative empirical results are known
PP: Benefits from learning from one another?

- Only anecdotal evidence is available:
  - [Belshee05]: New programmer without OOP knowledge came into a PP project heavily using C++ template metaprogramming. After only four weeks he was fit enough to train another newcomer all alone, at the same time tackling even parts of the 600-class code base he had never seen.
  - [Belshee05]: Promiscuous PP (changing pairs every 90 minutes) lead to all 11 members of the team learning a neat IDE editing feature within just 1 day
    - the paste stack, which had been discovered only accidentally

- Again, the effect is very difficult to measure quantitatively
  - It requires project-level observations

- No quantitative empirical results are known
PP: Influence on motivation?

All studies agree that PP is generally rather motivating

- A survey [WilKesCun00] explains that with a positive form of "pair pressure":
  - Both partners want to show their talent and quality work
  - The participants are highly concentrated on their work and keep each other on task
    - no reading emails or surfing the web etc.

Exceptions:

- Some programmers reject PP completely
  - usually without even trying it out
  - Programmers with longer experience tend to be more skeptical
- Pairing people with extremely different skills is problematic
  - The more capable partner may be slowed down too much
Summarization of the effects of pair programming and calculation of the overall cost-benefit ratios for adopting PP

- Increase of 5% on the total project costs caused by PP
  - experiment, [Müller03]

- Pairs have a higher efficiency and overall productivity rate compared to individual developers, and pair programming increases the business value of a project
  - experiment, [WilKes03]

- Both results lack ecological credibility and are highly preliminary
PP: Learning process?

What happens when trying to start with PP?
How best to start with PP?

• Some authors claim that PP beginners quickly find their way into the process [WilKes00]
  • At the university, the students were generally adjusted to PP after the first assignment
  • In industry, this adjustment period has historically taken hours or days, depending upon the individuals

• Details of the learning process are hardly known and little constructive advice is available so far
• We may need a description of steady-state PP first
PP: Recommendations for pairing?

- [KatWil04, KatWil05] claims that personality type (MBTI) and self-esteem are not critical for pair compatibility
  - but members prefer to pair with someone who has similar technical skills

- [Domino03] suggests that the members may need some level of specific interpersonal skills, in particular conflict resolution skills
  - quasi-experiment found some correlation of performance with results of Rahim Organizational Conflict Inventory (ROCI-II)

- [MuelPad04] suggests that subjectively feeling comfortable with each other correlates with shorter development times
  - but causality is unknown (experiment postmortem interview)
PP: Recommendations for pairing?

- [CaoXu05] suggests
  - pairing members of high and low competence levels was less enjoyable for the more competent participant
    - while the less competent participant took benefit
  - high competence level leads to deep-level thinking
    - and both participants enjoy the experience

- [Belshee05] claims that switching the pairs very frequently is highly beneficial in a low-skill situation
  - "Promiscuous Pair Programming": Almost continually one partner is new to the task
  - Claimed to lead to continuous "beginner's mind" and fast learning
PP: Recommendations for changing roles?

- There are some informal recommendations to keep the frequency of changes sufficiently high
  - But based on mostly subjective evidence

- Hardly any specific recommendations are known
PP: How does it work?

- No studies answer this question
  - Neither how nor why nor when PP works well or not-so-well
  - Today PP is a Black Box

We need decision criteria for when to use PP.
We need guidance for PP process improvements:
- Catalog of best practices, regarding
  - communication and knowledge exchange
  - decision making
  - role switching
- How to optimize PP towards the various goals:
  - Knowledge transfer,
  - integrating newcomers,
  - optimizing design,
  - optimizing correctness,
  - etc.
PP: How does it work?
Our research approach

- Basic idea: Look into the process
  - Not just at its outcomes: Investigate the PP microprocess

- First obtain a detailed description of a typical PP microprocess
  - Perhaps concentrating on only a few aspects at first (e.g. knowledge transfer, strategy, role behavior, work modes)

- To do this, we need detailed data about PP sessions
  - Audio, Video (people and screen activity)
  - ElectroCodeoGram (ECG)

- We need a mix of data sources to satisfy all our requirements:
  - Laboratory observations (e.g. of students) allow to see multiple instances of equivalent problem-solving sessions for comparison.
  - Field observations of professional programmers allow to improve and validate the ecological validity of resulting models.
PP: How does it work?
Our research approach (2)

Data evaluation approach:
1. Conceptualize a few videos
   - Thus form a description vocabulary
   - Research method: Grounded Theory
2. Use visualization to obtain an overview of the flow of events
3. Form models and hypotheses
4. Validate and refine them with further sessions
   - Use visualization of ECG data to find the relevant episodes quickly
5. Identify helpful and damaging behaviors
6. Describe them as patterns

Insights, models
Defining consistent concepts ("codes") is very difficult

1. A perspective on the data is needed
   - to avoid drowning in detail.
   - Acts as a filter: what to encode, what to ignore

2. It is helpful to perform coding in pairs
   - For challenging vagueness and enforcing consistency

3. It is helpful to use structured code names
   - `person.verb_object: "P1.propose_step", "P2.explain_state"

- We use the following perspective:
  - directly observable actions only, in particular communication
- We ignore anything requiring subjective judgement, such as
  - attribution of "helpful"/"unhelpful" to actions
  - attribution of emotions (happy, annoyed, surprised, focused, ...)
How does it work?
Basic process concepts found
PP: How does it work?
Preliminary insights

1. "Pair Phases" / "Focus phases": sometimes the pair interacts with high frequency
   - short phases (0.5 - 3 minutes)
   - happens at varying intervals

2. Driver and Observer are not on different levels of abstraction

3. The driver often performs "think aloud"

4. Ignoring the partner is not rare
PP: How does it work?

Status and further work

- We have recorded a number of PP sessions
  - 2 sets of student pairs doing (one set each for two tasks)
  - individual sessions from a number of companies (professional programmers)
  - interviews with professional programmers after sessions

- Almost finished: Description of PP basic process concepts
  - See figure above

- We will next look at individual types of phenomena
  - "focus phases", strategic and tactical decisions, misunderstandings, knowledge transfer, etc.

- We will extend the investigation to distributed contexts:
  - Saros, see subsequent slides
Saros: Distributed Party Programming

- Saros is an Eclipse plugin that couples multiple Eclipses remotely for real-time collaborative editing
  - with awareness functionality
Saros: Usage modes

- Saros allows Distributed Pair Programming and
- Supports multiple Observers
  - useful for joint reviews or for training newbies
- Supports multiple Writers
  - useful for Side-by-Side programming
  - useful for Distributed Party Programming (DPP):
    Multiple (say, 5) developers "hang around" in the same session, each doing their own development, and quickly helping each other out when needed

- Plenty of research to be performed on DPP
  - Do developers accept this? What awareness info is needed? How useful is the mode? How to use it well?
- Plenty of Saros development to be performed
  - Complete VoIP, sketching, screen sharing, ...
  - Improve architecture, robustness, automated testing, ...

Interested? Talk to us!

Lutz Prechelt, prechelt@inf.fu-berlin.de
Summary

- Pair Programming (PP) is the joint production of an artifact by two equal, active programmers with fully shared ownership
  - using only one keyboard and switching the driver role repeatedly
  - can be extended to remote settings, loose coupling, N persons

- PP has a substantial number of potential benefits
  - supported by anecdotal evidence
  - Empirical results regarding productivity are somewhat mixed
  - Empirical evidence regarding the other benefits is scarce, because measuring them is very difficult

- More research is needed
  - We are also extending the research to distributed settings
    - Development of Saros plugin
    - Corresponding evaluation research (in OSS projects and in industry)
  - Talk to AG SE if you consider participating in such research
Thank you!