Course "Softwareprozesse"

**Pair Programming (PP)**

Lutz Prechelt
Freie Universität Berlin, Institut für Informatik

- **Characterizations/claims:**
  - Williams, Beck, others
- **What happens in PP?**
  - base activities
  - knowledge transfer episodes
    - push, pull, co-produce, pioneering
- **PP session dynamics**
  - S vs. G knowledge
  - session types
- **Good PP: Togetherness**
- **Other results**
  - knowledge transfer effects, motivation, driver/observer
Learning objectives

- Understand the nature of and the dominant effects in PP
  - and how to do PP well

- Understand why quantitative research on PP is problematic
What is Pair Programming (PP)?

- **A practice (in XP):** "Write all production programs with two people sitting at one machine."
- **A work mode:** Work in pairs iff it appears appropriate.

What are your thoughts on this?
- What could be the **benefits**?
- What are potential **problems**?
- Your personal **experience** with PP?
Most well-known characterization

• [WilKesCun00]: "In pair programming, two programmers jointly produce one artifact (design, algorithm, code).
  • The two programmers are like a unified, intelligent organism working with one mind, responsible for every aspect of this artifact.
  • One partner, the driver, controls the pencil, mouse, or keyboard and writes the code.
  • The other partner continuously and actively observes the driver's work, watching for defects, thinking of alternatives, looking up resources, and considering strategic implications.
  • The partners deliberately switch roles periodically.
  • Both are equal, active participants in the process at all times"
Kent Beck's definition, his and others' claimed effects of PP

• Beck: "Pair programming is a dialog between two people simultaneously programming (and analyzing and designing and testing) and trying to program better. Pair programmers:
  - Keep each other on task.
  - Brainstorm refinements to the system.
  - Clarify ideas.
  - Take initiative when their partner is stuck, thus lowering frustration. [PP is more motivating]
  - Hold each other accountable to the team’s practices."

• Further claims by others:
  - Pairs are faster than solo programmers
    - or even: reduce effort
  - Pairs produce better designs
  - Pairs come out with fewer defects
  - Pairs learn from each other
Lecture structure

• There is little research on Beck's PP attributes/claims
• There is a lot of research on the "other" claims
  • some of it provides reasonable evidence
  • much of it is inconclusive, misleading, or both

• We will first look at research of the PP process as such
  • "How does it work?", "What are pairs doing?"
• and then look at the other research
  • to understand the reasonable evidence
  • to understand the problems of the rest.
AG SE work on PP

- AG SE researches PP since 2004
  - "What are pairs doing?"
  - 4 PhD dissertations 2012, 2013, 2018, 2020
    - Laura Plonka, Stephan Salinger, Julia Schenk, Franz Zieris
  - book about basic conceptualization of the PP process 2013
  - several articles
  - Collection of industrial PP session recordings

Julia Schenk
Franz Zieris

Lutz Prechelt, prechelt@inf.fu-berlin.de
PP: How does it work?
AG SE research approach

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

1. First understand the **base activities** of the programmers
2. Then obtain an **understanding of the total PP process**
   - concentrating on only a few aspects at first
     (e.g. knowledge transfer, strategy, role behavior, work modes)
3. and identify **helpful/unhelpful patterns of behavior**
   - PP behavior patterns and anti-patterns
4. Formulate these such as to become a **learnable PP skill**

- To do this, we need detailed data about PP sessions
  - collected 65+ sessions from 13 different companies, 1-3 hours
    (in vivo: professionals, actual problems, own environment)
    - Audio + Video (people and screen activity)
    - plus: interviews with developers after sessions (reflection)
Our research approach (2): The Data
PP: How does it work?
AG SE research approach (3)

Data analysis using the Grounded Theory Methodology (GTM):

• GTM: the *constant comparative method* of qualitative research
  • It leads to theories that are fully grounded in data
  • Its main prerequisite is *theoretical sensitivity*
  • Its main practices are *Memo Writing, Open Coding, Axial Coding, Selective Coding, Theoretical Sampling*
  • Supported by appropriate software (in our case *ATLAS.ti*)

• Rough research phases (super simplified):
  1. Open Coding forms a appropriate **vocabulary**
  2. Axial Coding identifies behavior **patterns**
  3. Selective Coding to describe the most helpful or problematic patterns to **advise practitioners**
Vocabulary: Types of verbal actions found

<table>
<thead>
<tr>
<th>product-oriented concepts</th>
<th>process-oriented concepts</th>
<th>universal concepts</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>amend_design</strong></td>
<td>Amend a given proposal regarding the structure and content of the program without rejecting the proposal.</td>
<td><strong>explain_gap_in_knowledge</strong></td>
</tr>
<tr>
<td><strong>ask_design</strong></td>
<td>Ask for a concrete proposal regarding the structure and content of the program.</td>
<td><strong>agree_gap_in_knowledge</strong></td>
</tr>
<tr>
<td><strong>challenge_design</strong></td>
<td>Reject a given proposal regarding the structure and content of the program and make an alternative proposal instead.</td>
<td><strong>explain_standard_of_knowledge</strong></td>
</tr>
<tr>
<td><strong>decide_design</strong></td>
<td>Select one from among several alternative proposals regarding the structure and content of the program.</td>
<td><strong>ask_standard_of_knowledge</strong></td>
</tr>
<tr>
<td><strong>agree_design</strong></td>
<td>Signal agreement with a given proposal regarding the structure and content of the program.</td>
<td><strong>think aloud_activity</strong></td>
</tr>
<tr>
<td><strong>challenge_step</strong></td>
<td>Reject a given proposal regarding the next tactical work step and make an alternative proposal instead.</td>
<td><strong>propose_hypothesis</strong></td>
</tr>
<tr>
<td><strong>decide_step</strong></td>
<td>Select one from among several alternative proposals regarding the next tactical work step.</td>
<td><strong>propose_finding</strong></td>
</tr>
<tr>
<td><strong>disagree_design</strong></td>
<td>Reject a given proposal regarding the structure and content of the program without making an alternative proposal.</td>
<td><strong>explain_finding</strong></td>
</tr>
<tr>
<td><strong>challenge_step</strong></td>
<td>Reject a given proposal regarding the next tactical work step and make an alternative proposal instead.</td>
<td><strong>propose_activity</strong></td>
</tr>
<tr>
<td><strong>disagree_step</strong></td>
<td>Reject a given proposal regarding the next tactical work step and make an alternative proposal instead.</td>
<td><strong>challenge_activity</strong></td>
</tr>
<tr>
<td><strong>explain_state</strong></td>
<td>Make a statement regarding the degree to which the current strategy or work plan has been worked through.</td>
<td><strong>challenge_finding</strong></td>
</tr>
<tr>
<td><strong>challenge_requirement</strong></td>
<td>Reject a given or proposed requirement and propose an alternative one instead.</td>
<td><strong>challenge_hypothesis</strong></td>
</tr>
<tr>
<td><strong>requirement</strong></td>
<td>Require a pair of a given (pre-specified) functional or non-functional requirement of the program.</td>
<td><strong>challenge_state</strong></td>
</tr>
<tr>
<td><strong>propose_strategy</strong></td>
<td>Propose a new requirement.</td>
<td><strong>challenge_hypothesis</strong></td>
</tr>
<tr>
<td><strong>requirement</strong></td>
<td>Require a given or proposed requirement and propose an alternative one instead.</td>
<td><strong>challenge_requirement</strong></td>
</tr>
<tr>
<td><strong>propose_strategy</strong></td>
<td>Propose a new requirement.</td>
<td><strong>challenge_activity</strong></td>
</tr>
<tr>
<td><strong>challenge_todo</strong></td>
<td>Propose todo.</td>
<td><strong>challenge_todo</strong></td>
</tr>
<tr>
<td><strong>propose_todo</strong></td>
<td>Propose todo.</td>
<td><strong>disagree_activity</strong></td>
</tr>
<tr>
<td><strong>challenge_activity</strong></td>
<td>Propose activity.</td>
<td><strong>disagree_hypothesis</strong></td>
</tr>
</tbody>
</table>

The HHI base concepts from Salinger, Prechelt: "Understanding Pair Programming: The Base Layer", BoD 2013
"Vocabulary":

- Base Activities (previous slide) are the process atoms
  - roughly: decision making (process/product) and knowledge transfer
    - our focus so far has been knowledge transfer
  - example session (with more-than-usual execution):

  ![Diagram showing execution, knowledge transfer, and decision making]

  - **gray**: interruptions, pauses, external help

  [Zieris20, Figure 4.1, p.141]
Knowledge Transfer: happens in per-topic episodes

- driven by knowledge need, pursued by one of the developers

- different modes:
  - **Pull**: "asking"
  - **Push**: "explaining"
  - **Production**: generating new understanding
    - together: Co-Production
    - or alone: Pioneering Production

- some symptoms of good pairs:
  - one topic at a time, finishing topics, splitting complex topics

- [ZiePre14] Zieris, Prechelt: *On Knowledge Transfer Skill in Pair Programming*, ESEM '14
- [ZiePre16] Zieris, Prechelt: *Observations on Knowledge Transfer of Professional Software Developers during Pair Programming*, ICSE '16
Patterns of Session Dynamics: Two types of task-relevant knowledge

- SW development is knowledge-intensive work
  - programming languages, technology stacks, design patterns, ...
  - coding styles, requirements, system architecture, ...
- Two types of task-relevant knowledge [Zieris20]:
  - Zieris, Prechelt: "Explaining Pair Programming Session Dynamics from Knowledge Gaps", ICSE 2020
  - \( S \): specific knowledge about the software system
  - \( G \): generic knowledge about methods and technology

G

expert-for-this developer, unfamiliar part of system

S

experienced-for-this dev., well-known part of system

G

Lutz Prechelt, prechelt@inf.fu-berlin.de
PP Session Dynamics: Pair configurations

- Each developer enters a PP session with a G-S-profile
  - depending on what she already knows about the system (S)
  - and software development in general (G)
  - only as relevant for the task

- Pairs form **constellations**, each with distinct challenges and **session dynamics**:
  1. No Relevant Gaps
  2. One-Sided S Gap
  3. Collective S Gap
  4. Complementary Gaps
  5. Too-Big Two-Fold Gap

- (others might be possible, but are yet to be observed)
PP Session Dynamics:
How to solve a problem as a pair?

• In a session, the pair as a whole needs to reach **high S**:  
  • i.e., complete understanding of the task-relevant system parts.  
  • (otherwise: no systematic solution)

• Reaching high S *individually* might be desirable  
  • but not necessary, if the developers are not expected to continue working on the task alone

• **High G is not necessary**  
  • mid-or-high G is required once the system is understood  
  • too-low G can be a problem (solution becomes too difficult)  
  • G may also help in building up S

• Two ways of dealing with knowledge gaps:  
  1. Transfer or acquire knowledge within the session  
  2. Limit scope of current task (reduce what is "high S and G")
Session Dynamics: Key success factors [Zieris20]

- Pairs must constantly maintain high **Togetherness**
  - joint system understanding (S)
  - joint ideas of how to develop SW (tools, methods)
  - joint tactical plan
  - no obstacles from workspace awareness or language barrier

- Pairs must pick appropriate transfer modes
  - Push, pull, co-produce, pioneer

- Pairs must pursue **One Topic at a Time**
  - Limit scope
  - Explicitly return from subtopics

Williams' definition (driver/observer) is an anti-pattern!
Session Dynamics Pair Constellations: Type 1 - No Relevant Gaps

- **Characterization:**
  - Both developers understand the system (high S) and possess the required programming skills for the task (high G).

- **Occurrence:**
  - Rare, only if the pair recently worked on the same task together to build up high S.

- **Benefits (theoretical):**
  - Modest, each developer could work on the task alone, and the task provides only few opportunities to learn something.
    - PP appears hardly needed.
    - May be useful if e.g. correctness is critical.
Session Dynamics Pair Constellations: Type 2 - One-Sided S Gap

• Characterization:
  • One developer has an S-advantage that needs to be addressed if the two should work as a pair.

• Occurrence:
  • Common, e.g.: Developer A started working on a task, B joins later \(\rightarrow\) A has S-advantage

• Challenges:
  • B might not be aware of the gap and might not understand A's ideas.
  • Until the gap is closed, there is an asymmetry. A can help B, but B might have personal preferences for how to close a knowledge gap.

• Solutions:
  • Make sure the S gap becomes visible: Let A explain what she did.
  • Try different modes: Push, Pull, reading aloud [ZiePre14]

Lutz Prechelt, prechelt@inf.fu-berlin.de
Session Dynamics Pair Constellations: Type 3 - Collective S Gap

- **Characterization:**
  - Both developers lack relevant portions of S. Pairs needs to build up S to solve the task.

- **Occurrence:**
  - The pair starts on a new task together: Both need to find out which parts of the system are relevant.

- **Challenges:**
  - Many plausible ways for approaching this.
  - Often, either of the two will have an insight first: Need to stay on the same page.

- **Solutions:**
  - Integrate partial understanding often: Co-Production [ZiePre14]
  - Let the partner take his time if he lags behind at some points: let partner think aloud, maybe offer Pushes.
Session Dynamics Pair Constellations
Type 4 - Complementary Gaps

- **Characterization:**
  - Developer A understands the system, but lacks general SW development skill.
  - B doesn't know the system (well) and has better development skills.

- **Occurrence:**
  - Not so uncommon:
    Since S- and G-levels are task-dependent: Pair may choose a task (or amend its goals) such that they complement each other.

- **Benefits:**
  - Session can be mutually satisfactory
    - B may help A to understand the system faster
    - A may pick up some G knowledge along the way
Session Dynamics Pair Constellations: Type 5 - Too-Big Two-Fold Gap

- **Characterization:**
  - Both developers know too little about the system to make meaningful changes *and* lack background knowledge to do much about it.

- **Occurrence:**
  - Happens: New technology (no G knowledge) and author unavailable (no S knowledge)

- **Challenges:**
  - PP process can break down entirely:
  - G knowledge too low to acquire enough S knowledge.
  - For unexperienced pairs: having a partner might make it worse
    - PP is a skill in itself.

- **Solutions:**
  - For this task: Different pair, or try alone
  - For this pair: Different task, or radically limit the scope
PP Session Dynamics: Summary

- Relative and absolute **S gaps** dominate PP session dynamics
  - Core difficulty: Reach high S as a pair
- **Complementary situations** is when PP pays off best
  - Since relevant knowledge is task-dependent: can be achieved by choosing the "right task" for a pair
- Real world: **System understanding trumps programming skills**
  - Luckily, PP is great for improving one's system understanding
- Problem with many PP studies: Students and **isolated tasks**
  - i.e., there is no system and hence **no relevant S knowledge**
  - only general problem solving and programming skills G
Distributed Pair Programming (DPP)

• The partners are not physically in the same room and use a separate computer each

• Their interaction is supported by a collaborative editor and audio conferencing, perhaps also video.
  • Saros (Eclipse, IntelliJ)[AG SE], VS Code, other things
  • Allows >2 participants (Distributed Party Programming)
  • Allows concurrent edits (Distributed-Pair programming)
    • Schenk, Prechelt, Salinger: "Distributed-Pair Programming (DPP) is not just Distributed Pair-Programming (dPP)"
      Capable pairs use this judiciously for slightly higher Fluency without loss of Togetherness.

• Sufficient workspace awareness is critical:
Workspace awareness in Saros

- Saros is an IDE plugin that couples multiple IDEs remotely, syncs local files, and creates remote workspace awareness:

Lutz Prechelt, prechelt@inf.fu-berlin.de
What about other PP research?

Two big secondary studies:

The questions. Participants are often students!

(we ignore this part)
Research methods used, Quality

  - Surveyed 154 research articles on PP in industry
    - Research approach, exercise vs. project, #subjects, ...
  - Identified 608 statements about the 18 PP aspects
    - ranked by relevance:
      1 – fair, 2 – moderate, 3 – good, 4 – excellent
      - only 8% had anything excellent, another 13% had anything good
    - based on: rigor of data collection, comparative data, #subjects, realism of context, study duration, length of discussion, ...

Lutz Prechelt, prechelt@inf.fu-berlin.de
"The effectiveness of pair programming: A meta-analysis" [HanDybAri09]

- Meta-Analysis of all PP **experiments** with available data
  - incl. students as proxies for developers

Experiments provide high internal validity. (generalization unclear, but we believe the results as they stand.)

[→ V+Ü Empir. Methoden im SW Eng., SoSe]
Is PP faster? Does it produce better quality?

- **Quality**: "small positive effect" (PP has little effect on quality)
  - Effect size: 0.33
    - CI$_{95}$: [0.07, 0.60]
- **Duration**: "medium positive overall effect" (PP is faster)
  - Effect size: 0.53
    - CI$_{95}$: [0.13, 0.94]
- **Effort**: "medium negative overall effect" (PP costs more)
  - Effect size: -0.52
    - CI$_{95}$: [-1.18, 0.13]
- **Overall**: mixed results
  - inter-study variance (heterogeneity):
    - *medium* for Quality and Duration; *high* for Effort
  - One-study-removed analysis: considerable changes to effect sizes

Effect Sizes: 2 0.7 0.25 (fabricated data, for illustration only)
Problems of quantitative black-box perspectives

- **Pair programming as a "black box":**
  - Some work alone, others "use PP" (independent variable)
  - Tasks are finished within some time with a certain quality (dependent variable)

- **Problem 1:** Plethora of context variables to control, including
  - Experience, Personality
  - Task complexity, type of task, system domain
  - Roles, degree of collaboration
  - Workspace, infrastructure

- **Problem 2:** Hard-to-measure long-term outcomes, such as
  - Avoided architectural flaws and avoided information silos

- **Problem 3:** No explanation of how outcomes come to be
  - No idea how many pairs used PP well

- **Conclusion:** employ other methods than experiments
Benefits from more people being familiar with code?

- 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 ("truck number")

- 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
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 of PP he could train another newcomer alone on parts of the 600-class code base he had never seen.
  • [Belshee05]: Promiscuous PP (changing pairs every 90 minutes) led 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?

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.

- [CaoXu05] on competence-level combinations:
  - high+low: less enjoyable for the more competent participant
    - while the less competent participant took benefit
  - high+high: leads to "deep-level thinking"
    - and both participants enjoy the experience

- Some programmers reject PP completely
  - usually without even trying it out
  - Programmers with longer experience tend to be more skeptical
What about Driver and Navigator/Observer?

- Classic "definition" of PP, from [WilKesCun00]:
  - One partner: "driver", controls keyboard, is writing code
  - The other "actively observes" ... "watching for defects, thinking of alternatives, looking up resources, and considering strategic implications"
  - Pair: like a "coherent, intelligent organism working with one mind"

- Empirical: 24 one-hour sessions from 4 companies [BryRomBou08]
  - Analyzed: level of abstraction of 14k+ sentences (e.g. syntax, blocks, domain)
  - Compared: Expected distribution per definition vs. actual distribution

### #utterances at different abstraction levels

- Driver and observer do **not** seem to think on **different levels** of abstraction.
Summary

• PP can provide huge learning benefits

• It leads to focused work, spreads knowledge, and tends to produce better designs and fewer defects
  • Raw speed comparisons are therefore misleading

• The process is usually dominated by acquiring the task-specific system knowledge (S)
  • PP is most useful if this is difficult
  • or if the pair's knowledge is complementary.
  • In the real world, system understanding trumps progr. skills
    • Speed comparisons ignoring this are irrelevant

• PP done badly can be inefficient
  • There is a PP skill separate from programming skill

• The key success factor is maintaining Togetherness
  • joint system understanding, joint approach, good communication
Thank you!

This was too much material to digest. Please review these slides again!