Course "Softwareprozesse"

Agile Methods: Pair Programming (PP)

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

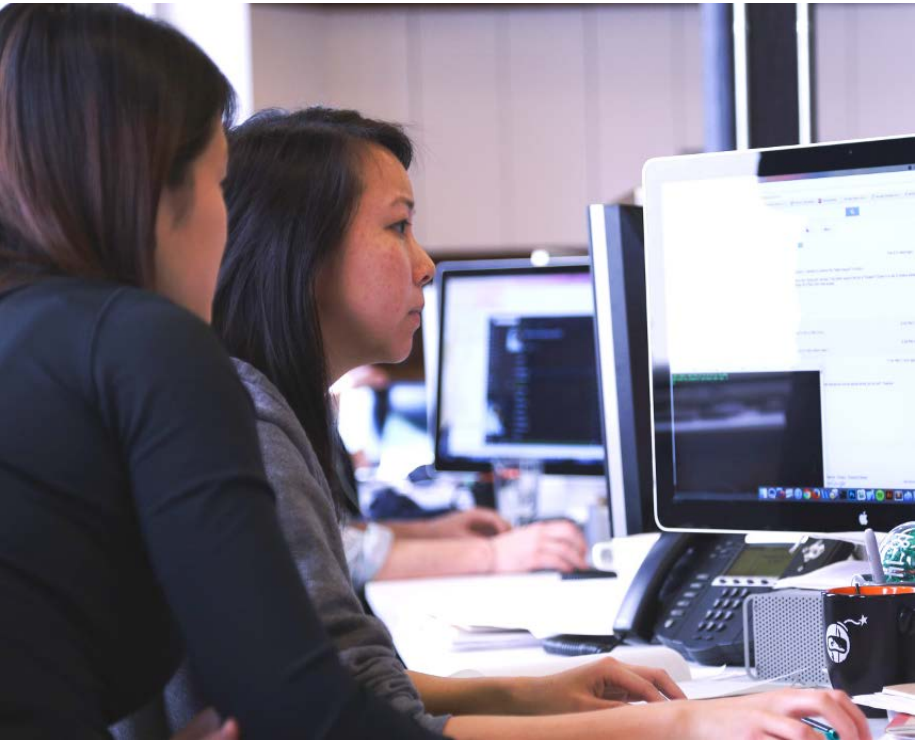
- Aspects of Pair Programming
- Empirical results
  - An example study of PP
  - PP effectiveness: Meta-Analysis
  - Issues with research regarding PP as a black box
  - Overall: State of PP Research
  - Benefits from more people being familiar with code?
  - Influence on motivation?
  - Recommendations for pair forming and process?
- Our PP research approach
  - Data gathering, evaluation
  - Role of knowledge in PP session dynamics
  - Distributed Pair Programming

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

What is Pair Programming?

- **Idea:** Two developers work together on a task
  - usually: co-located, on one machine
- **XP**: All production code must be written in PP manner



<u>What are your thoughts on this:</u>

- What could be the **benefits**?
- What are potential **problems**?
- Do have personal **experience** with PP?

Aspects of Pair Programming

Freie Universität Berlin

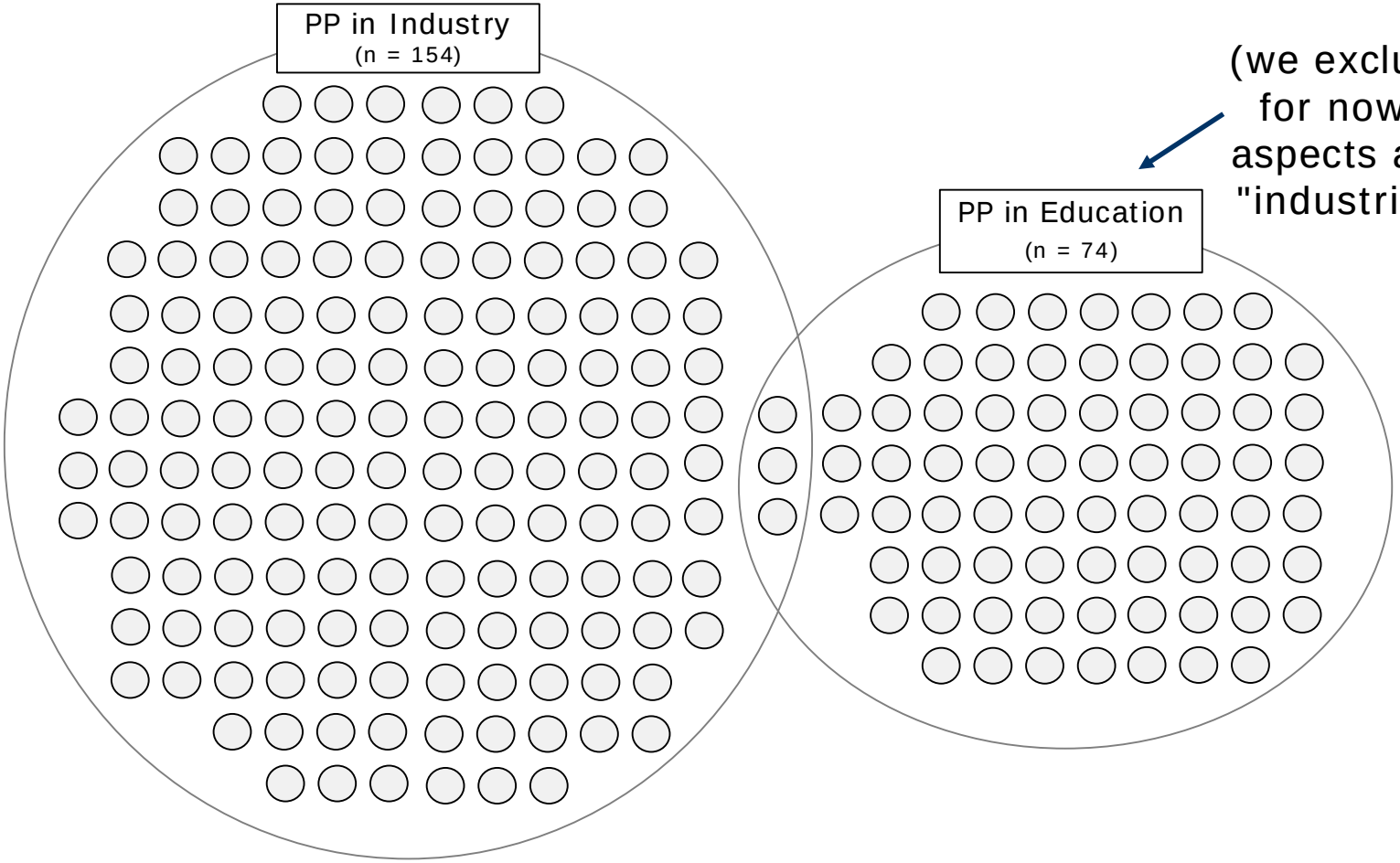
1. Main Effects
   - **Productivity**: Effort/duration, work done
   - **Quality**: defect density, readability, design quality
2. Developer
   - (Effect of & effect on) **Knowledge of Work:** SW System, Tools, Domain
   - **Feeling of Work**: Satisfaction / Enjoyment, Exhaustion, Pair Pressure
   - **Feeling of PP**: Resistance / Enjoyment
3. Environment
   - **Dev. Process**: support of other practices, process conformance
   - **Infrastructure**: HW/SW, Office
4. Preparations for PP
   - **Adoption**: resistance, learning
   - **Managing PP**: for what, how often, how close
   - **Pair formation**: with whom, rotation, unfinished tasks
   - **Targets**: activities & situations
5. PP session
   - **Pair combination**: personality, expertise, experience
   - **Roles**: leader, level of thinking, switching
   - **Communication**: content, amount, issues, relationship
   - **Breaks**: distractions vs. breaks, when and how long
6. Utilization rate
   - Breadth and depth of use

Framework by [VanMan13]

What do we know about PP?

Two big secondary studies:

- PP in industrial settings [VanMan13] vs. PP in education [SalMenGru11]



(we exclude this part for now, since the aspects above are of "industrial" interest)

What do we know about *industrial* PP?

- "*A Systematic Mapping Study of Empirical Studies on the Use of Pair Programming in the Industry*" [VanMan13]

- 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
  - based on: *rigor of data collection, comparative data, #subjects, realism of context, study duration, length of discussion, …*

- We take a look at *good* and *excellent* sources →

PP Main Effects: Productivity & Quality

Freie Universität Berlin

- Productivity metrics used:
  - **Duration**: wall-clock time, or *How long until task is done?*
  - **Effort**: person hours, or *How much does it cost?*
    - -50% duration → same effort
    - -15% duration → +70% effort
  - **Efficiency:** lines of code per person hour
- Quality metrics used:
  - **Correctness**: fulfilled requirements, passed (pre-defined) tests, relative defect density
  - **Readability**: code style violations, comment ratio

- Evidence: only 1x "excellent" and 4x "good", including
  - [AriGalDyb07]: 295 hired consultants, 1 work day
  - [Nosek98]: 15 subjects, work-related tasks, <1 hour ➔

"The case of collaborative programming" [Nosek98]

- Experiment (+survey): 5 pairs vs. 5 individual programmers
  - Experienced programmers, randomly assigned
  - Work in their environments, using their own equipment
  - Same challenging task for all: database consistency check
- Hypotheses (excerpt):
  1. "Groups will **take less time** on average to solve the problem"
  2. "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
- Time limit: 45 min
- Quality metrics: two graders judged solutions
  - (>90% agreement)

[Nosek98] Results & Conclusions

Variable	Control Group (Individuals) mean (st. dev.)	Experimental Group (Teams) mean (st. dev.)
Performance Scores:	n = 5	n = 5
READBILITY	1.40 (0.894)	2.00 (0.000)
FUNCTIONALITY	4.20 (1.788)	5.60 (0.547)*
SCORE	5.60 (2.607)	7.60 (0.547)*
TIME (minutes)	42.60 (3.361)	30.20 (1.923)

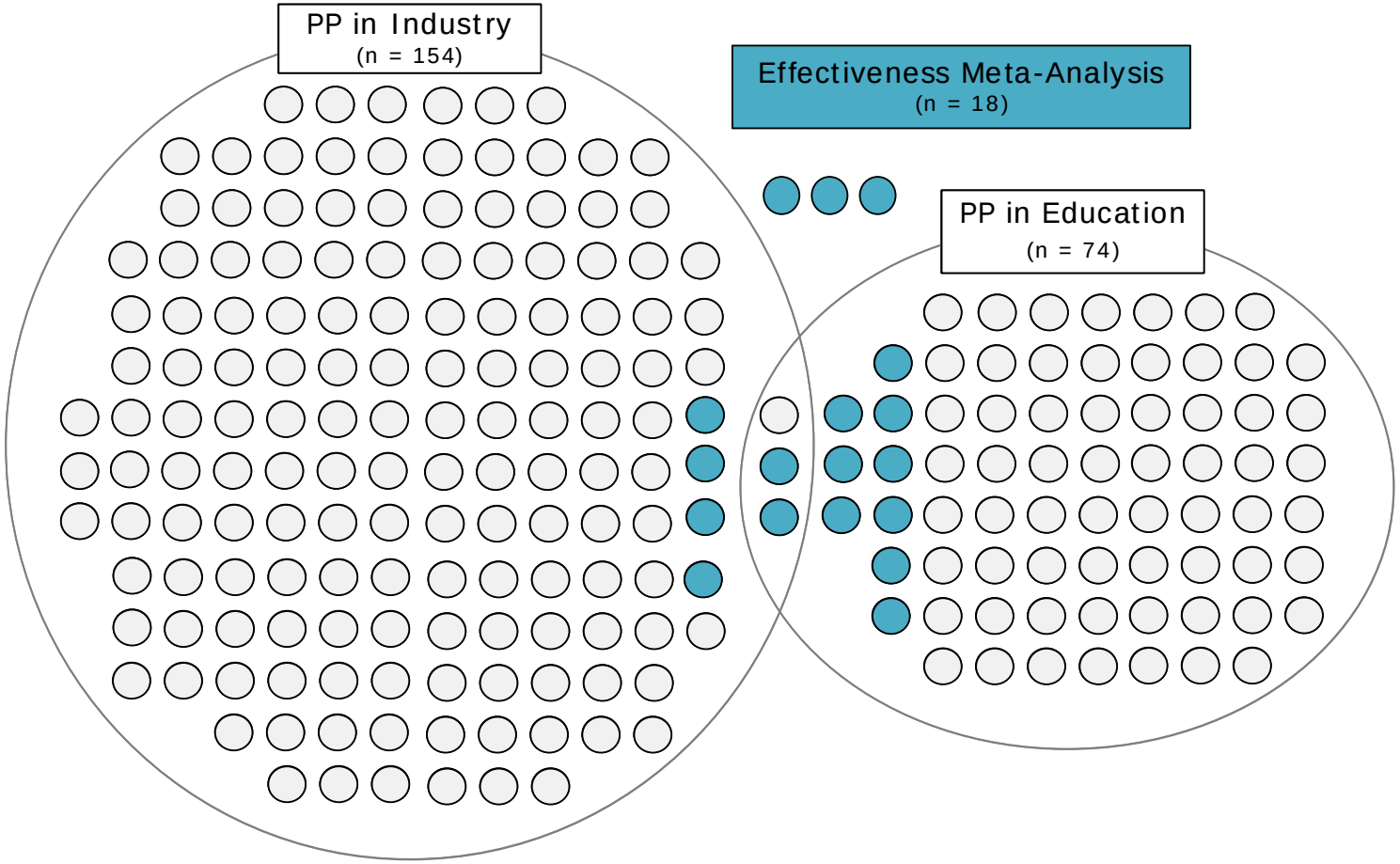
*less then 1 in 20 that results are due to chance

indiv. take 41% longer, but difference is not significant

- Hypothesis 1 (pairs' solutions are more readable and more functional, SCORE = READABILITY + FUNCTIONALITY):
  - Is confirmed
- Hypothesis 2 (faster):
  - Is not confirmed, as the difference is not statistically significant
- So: What about other PP experiments?

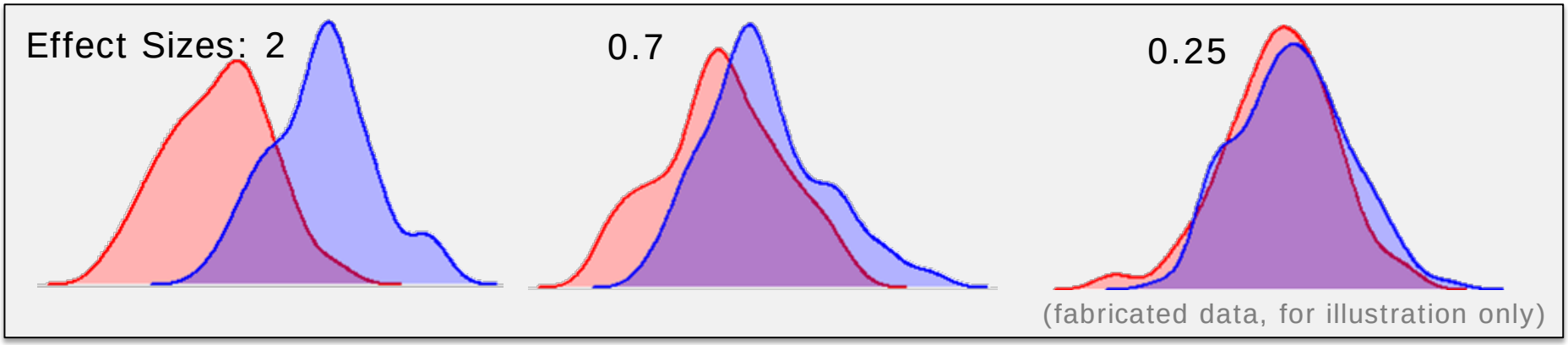
"The effectiveness of pair programming: A meta-analysis" [HanDybAri09]

- Meta-Analysis of all PP **experiments** with available data
  - incl. students: as a proxy for developers, convenience sample



[HanDybAri09] Results

- **Quality**: "small positive overall effect" of PP
  - 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



[HanDybAri09] Conclusions

- **Quality**:
  - "current state of knowledge suggest that pair programming is beneficial for achieving correctness on highly complex programming tasks"
- **Productivity**:
  - "Pair programming may also have a time gain on simpler tasks"

- But:
  - "pair programming is **not uniformly beneficial or effective**", "inter-study **variance** is **high**"
  - "**moderating factors** play an important role" ... "**expertise** and **task complexity**", "amount of **training in pair programming**, motivation, team climate, etc."

Note: In [AriGalDyb07], only 10 out of 196 subjects had prior PP experience.

Ways towards understanding PP: Blackbox perspectives (quantitative)

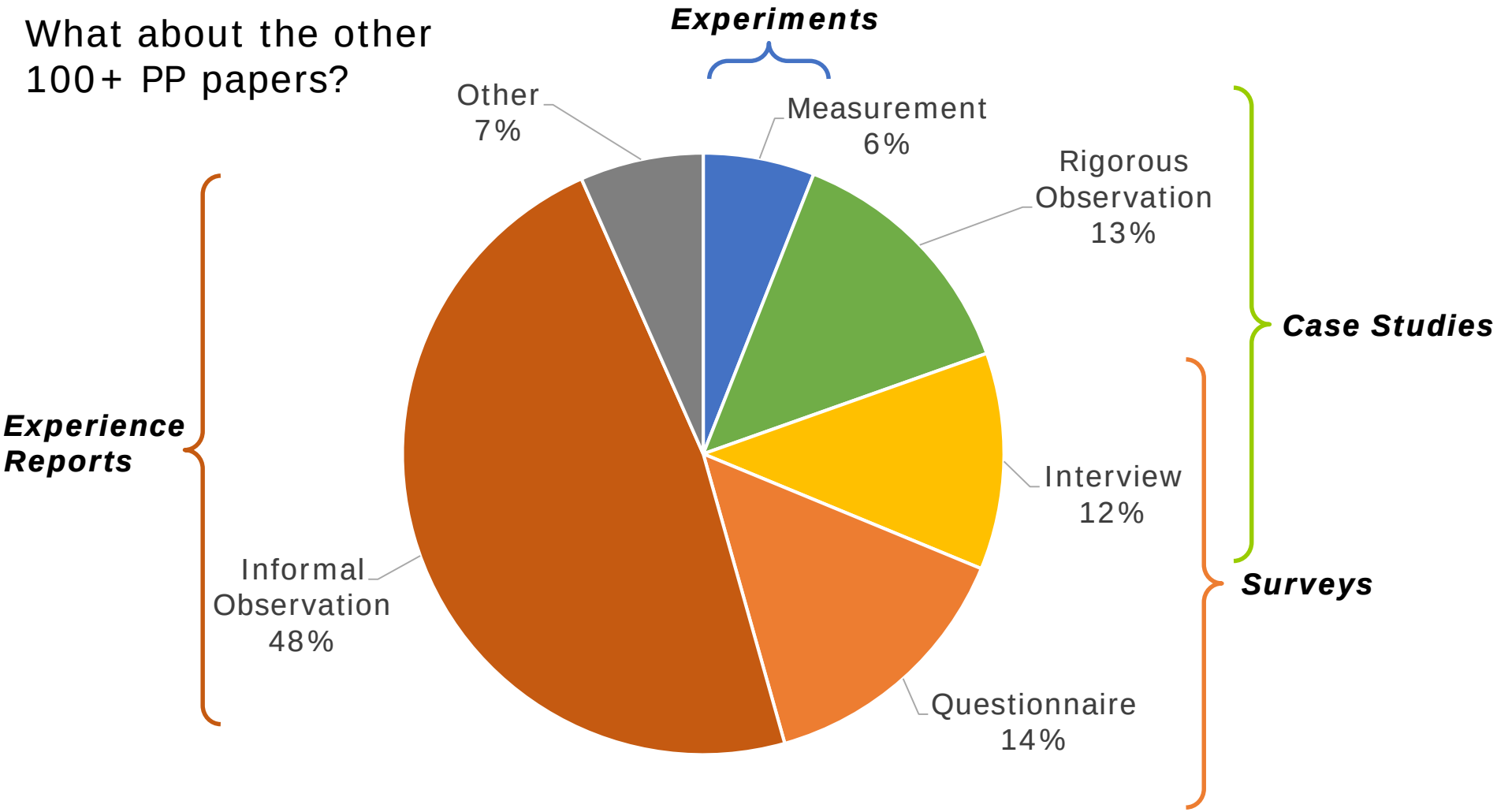
Freie Universität Berlin

- 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
- **Problem 4**: No constructive advice on how to pair program

- **Conclusion**: employ other methods than experiments

Research Approach & Data Collection

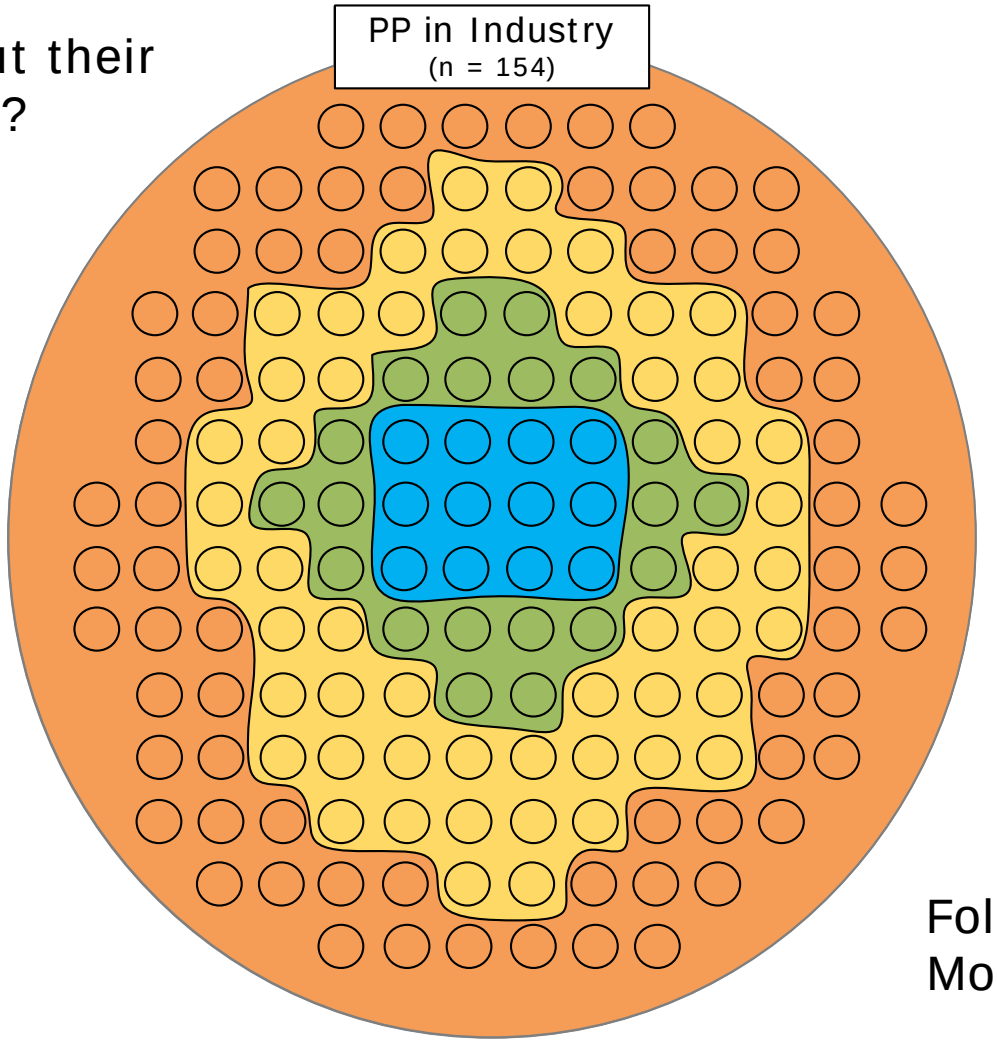
What about the other 100+ PP papers?



Data from [VanMan13]

Relevance of papers on industrial PP

How about their relevance?



Most relevant statement in paper

12x excellent

20x good

52x moderate

70x fair

Following slides:
Mostly anecdotal evidence

Data from [VanMan13]

Questions about PP

- Already discussed (with mixed results):
  - Raw productivity compared to two separate programmers?
  - Quality of resulting design?
  - Quality of resulting code?
- Other aspects:
  - Benefits from more people being familiar with code?
  - Benefits from learning from another?
  - Influence on motivation?
  - Recommendations for forming pairs?
  - Recommendations for how to pair program?

Subsequent slides explain the questions and the respective state of knowledge

PP: Benefits from more people being familiar with code?

Freie Universität Berlin

- 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

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:

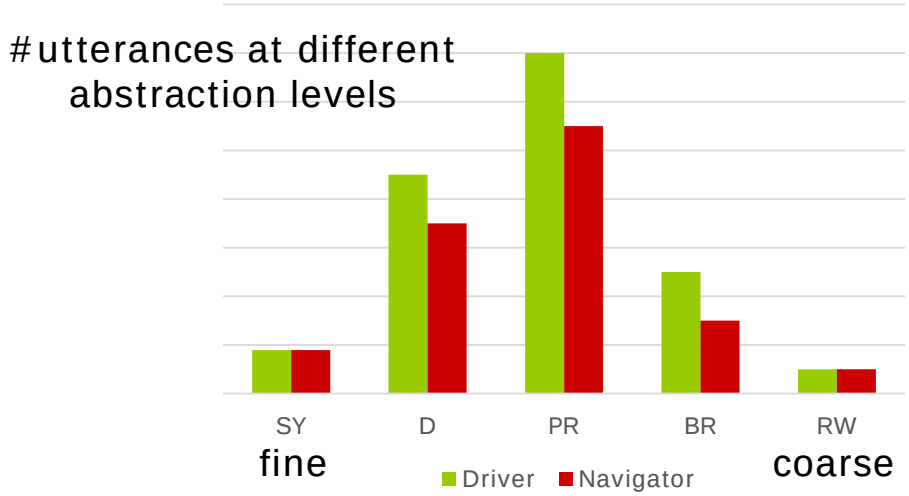
- Some programmers reject PP completely
  - usually without even trying it out
  - Programmers with longer experience tend to be more skeptical
- Pairing people with very different skills is problematic
  - The more capable partner may be slowed down too much
    - (except when teaching is the goal)

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
- [MulPad04] suggests that subjectively feeling comfortable with each other correlates with shorter development times
  - but causality is unknown (experiment postmortem interview)
- [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

PP: How does it work? 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**



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

PP: How does it *actually* work?

- Very few studies address this question
  - Neither how nor why nor when PP works well (or not-so-well)
  - PP is usually treated as a black box
    - [BryRomBou08] (previous slide) is one of very few notable exceptions

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
  - using Side-by-Side/distributed PP styles etc.
- How to optimize PP towards the various goals:
  - Knowledge transfer,
  - integrating newcomers,
  - optimizing design,
  - optimizing correctness,
  - etc.

PP skill

PP: How does it work? Our research approach

Freie Universität Berlin

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

- First understand the **base activities** of the programmers
- 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)
- Then identify **helpful / unhelpful patterns of behavior**
  - PP behavior patterns and anti-patterns
- Formulate these such as to become a **learnable PP skill**

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

Java - _Pro/src/de/ /pro/featurelayer/att

passau

File Edit Source Refactor Navigate Search Project Tomcat Run Window Help

Package Explorer

- _Join_Selector_Core 52599 [https://s
- _Join_Selector_Pro 52599 [https://svn
- _Map 53331 [https://svn01.erg.
- _Pro 53337 [https://svn01.erg.
  - src 53337
    - de. 53337
      - 52523
      - 53337
        - pro 53337
          - attributemodification 53337
            - demo 52186
            - table 52180
            - virtual 53337
              - test 52183
              - VirtualAttribute.java 53
              - VirtualAttributeObjectU
              - VirtualColumnAttribute.
              - VirtualColumnFactory.ja
            - UserDefinedColumnAttribut
            - UserDefinedColumnAttribut
          - authorization 52316
          - dataaccess 51302
          - dbadapter 52317
          - dblegend 51302
          - dbservices 40284
          - demo 50873
          - development 52438
          - edit.dialog.demo 32111
          - featurelayer 53337
            - attributes 53337
              - action 44146
              - demo 52426
              - edit 53337
                - EditColumnAttribute
                - EditColumnAttribute
              - test 50873
              - AttributeNameSorter.ja
              - AttributeTableActionFa
              - AttributeTableSelection
              - AttributeTableViewTool
              - BlinkMouseListener.jav
              - CopyAttributeTableActi
              - DisplayNameFeaturePro
              - FeatureAttributeTableM
              - FeatureLayerAttributeT
              - FeatureLayerAttributeT
              - FeatureLayerAttributeV
              - FeatureLayerAttributeV
              - FeatureLayerTableView
              - FeatureLayerTableView
              - FeatureProxiesTableMo
              - FeatureProxyTableMod
              - ViewMapSelectionCo
              - IAttributeTableActionF
              - IAttributeTableActionF
              - IAttributeTableActionF
              - IAttributeViewConfigur.
              - IFeatureLayerAttribute
              - TermAttributeTable
              - IToSelectionFilterable.j
              - NullAttributeTableActio
              - ShowAttributeTableAct

IFeatureAttributeConfiguration.java | IVirtualColumn.java | VirtualAttribute.java | *EditColumnAttributesAction.java

```
import de. lib.gui.resources.BasicActionIconResources;

import net. commons.core.util.ArrayUtilities;
import net. commons.core.util.ITransformer;
import net. commons.swing.action.SmartAction;
import net. commons.swing.dialog.userdialog.UserDialog;

// NOT_PUBLISHED
public class EditColumnAttributesAction extends SmartAction {

  private final IFeatureLayerAndThemeLayerId featureLayerAndThemeLayerId;

  public EditColumnAttributesAction(final IFeatureLayerAndThemeLayerId featureLayerAndThemeLayerId) {
    super("Spalteneigenschaften bearbeiten...", BasicActionIconResources.PROPERTIES_ICON);
    this.featureLayerAndThemeLayerId = featureLayerAndThemeLayerId;
    setToolTipText("Spalteneigenschaften bearbeiten...");
  }

  @Override
  protected void execute(final Component parentComponent) {
    final IFeatureLayer featureLayer = featureLayerAndThemeLayerId.getFeatureLayer();
    final List<ColumnAttributeModificationRow> attributeRows = new ArrayList<ColumnAttributeModificationRow>();
    final IFeatureAttributeConfiguration attributeConfiguration = featureLayer
        .getAttributeConfiguration();
    attributeConfiguration.setVirtualColumns(null);
    for (final IColumnAttribute attribute : attributeConfiguration.getAllColumnAttributes()) {
      attributeRows.add(ColumnAttributeModificationRow.createFor(attribute, featureLayer
          .getUserAttributeModifications()));
    }
    final ColumnAttributeModificationRow[] rowArray = attributeRows
        .toArray(new ColumnAttributeModificationRow[attributeRows.size()]);
    final IVirtualColumn[] virtualColumns = attributeConfiguration.getVirtualColumns();
    final VirtualAttribute[] virtualAttributes = getVirtualAttributes(virtualColumns);
    final UserDefinedColumnAttributeModificationDialogPage dialogPage = new UserDefinedColumnAttributeModificationDialogPage(
        new UserDefinedColumnAttributeModificationSetup(
            featureLayer.getBaseGeometryType(),
            rowArray,
            virtualAttributes));
    final UserDialog userDialog = new UserDialog(parentComponent, dialogPage);
    userDialog.show();
    if (userDialog.isCanceled()) {
      attributeConfiguration.setVirtualColumns(virtualColumns);
      return;
    }
    attributeConfiguration.setVirtualColumns(VirtualColumnFactory.createVirtualColumns(dialogPage
        .getSelectedVirtualAttributes(), attributeConfiguration.getAllColumnAttributes()));
    for (final ColumnAttributeModificationRow row : attributeRows) {
      row.applyChangesTo(featureLayer.getUserAttributeModifications());
    }
    featureLayer.getFeatureProxySet().fireTableStructureChanged();
  }

  private VirtualAttribute[] getVirtualAttributes(final IVirtualColumn[] virtualColumns) {
    return ArrayUtilities.transform(
        virtualColumns,
        VirtualAttribute.class,
        new ITransformer<IVirtualColumn, VirtualAttribute>() {

          @Override
          public VirtualAttribute transform(final IVirtualColumn input) {
            return ((VirtualColumnAttribute) input.getColumnAttribute()).getVirtualAttribute();
          }
        });
  }
}
```

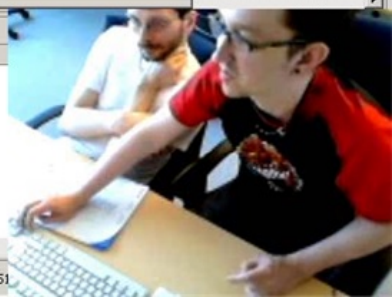
Outline | Search

'de .pro.attributemodification.virtual.VirtualAttribute' - 41 references in workspace (no JRE) (5 matches filtered from view)

- _Pro 53337 [https://svn01.erg. .net/svn/
  - src 53337
    - de. .pro.attributemodification 53337
      - UserDefinedColumnAttributeModificationDialogPage 52189
        - addPanels(ISmartDialogPanelsBuilder) (3 matches)
        - getSelectedVirtualAttributes()
        - selectionModel
        - UserDefinedColumnAttributeModificationDialogPage(User
      - UserDefinedColumnAttributeModificationSetup 52186
        - getVirtualAttributes()
        - UserDefinedColumnAttributeModificationSetup(BaseGeom
        - virtualAttributes
    - de. pro.attributemodification.demo 52186
    - de. pro.attributemodification.virtual 53337
    - de. pro.attributemodification.virtual.test 52183
    - de. pro.featurelayer.attributes.edit 53337
      - EditColumnAttributesAction 53337

Problems | Javadoc | Declaration | Console | History | SVN Repositories | Progress

```
<terminated> - generate hibernatemapping [Ant Builder] D:\java\jdk\1.6.0\bin\javaw.exe (07.05.2008 10:53:01)
Buildfile: F:\PRJ31\ Integrated\build.xml
generate.hibernatemapping:
     [copy] Copying 1 file to F:\PRJ31\ Integrated\resources\mappings\
BUILD SUCCESSFUL
Total time: 297 milliseconds
```



11:55:21 07.05.2008 18:43.600

Writable | Smart Insert | 51

PP: How does it work?
Our research approach (2)

Freie Universität Berlin

- We started studying PP and collecting data in 2004:
  - We perform field observations of professional programmers allow to develop *relevant* theories
  - so far: 60+ sessions from twelve different companies, 1-3 hours (in vivo: professionals, actual problems, own environment)
    - plus: interviews with developers after sessions (reflection)

- We also collected data under more controlled conditions:
  - Laboratory observations of students,
    to allow seeing *multiple instances of equivalent problem-solving* for comparison

PP: How does it work? Our research approach (3)

Freie Universität Berlin

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 processes are *memo-writing, open coding, axial coding, selective coding, theoretical sampling*
  - Supported by appropriate software (e.g. ATLAS.ti)

- Our goals:
  1. Form a appropriate **vocabulary** (~open coding)
  2. Identify helpful and problematic behavioral **patterns** (~axial coding)
  3. Describe most relevant ones to **advise practitioners** (~selective coding)

PP: How does it work?
Types of verbal actions found

Freie Universität Berlin

product-oriented concepts

amend_design
Extend a given proposal regarding the structure and content of the program without rejecting the proposal.

ask_design
Ask for a concrete proposal regarding the structure and content of the program.

challenge_design
Reject a given proposal regarding the structure and content of the program and make an alternative proposal instead.

agree_design
Signal agreement with a given proposal regarding the structure and content of the program.

decide_design
Select one from among several alternative proposals regarding the structure and content of the program.

propose_design
Make one or several alternative proposals regarding the structure and content of the program.

disagree_design
Reject a given proposal regarding the structure and content of the program without making an alternative proposal.

remember_requirement
Remind the pair of a given (pre-specified) functional or non-functional requirement of the program.

challenge_requirement
Reject a given or proposed requirement and propose an alternative one instead.

agree_requirement
Signal agreement with a given or proposed requirement.

propose_requirement
Propose one or several alternative program characteristics that should be considered to be a requirement.

miscellaneous

mumble_sth
Make an incomprehensible utterance (highly fragmentary or acustically unclear).

say_off topic
Make an utterance that has nothing to do with solving the programming task.

process-oriented concepts

amend_step
Extend a given proposal regarding the next tactical work step without rejecting the proposal.

ask_step
Ask for a concrete proposal regarding the next tactical work step.

explain_completion
Make a statement regarding the degree of completion of the current tactical work step.

challenge_step
Reject a given proposal regarding the next tactical work step and make an alternative proposal instead.

agree_step
Signal agreement with a given proposal regarding the next tactical work step.

agree_completion
Signal agreement with a statement regarding the degree of completion of the current tactical work step.

decide_step
Select one from among several alternative proposals regarding the next tactical work step.

propose_step
Make one or several alternative proposals regarding the next tactical work step.

challenge_completion
Reject a statement regarding the degree of completion of the current tactical work step and make an alternative statement.

disagree_step
Reject a given proposal regarding the next tactical work step without making an alternative proposal.

explain_state
Make a statement regarding the degree to which the current strategy or work plan has been worked through.

amend_strategy
Extend a proposed strategy or work plan without rejecting it.

ask_strategy
Ask for a concrete proposal regarding the strategy or work plan to be chosen.

agree_state
Signal agreement with a statement regarding the degree to which the current strategy or work plan has been worked through.

challenge_strategy
Reject a given proposal regarding the strategy or work plan and make an alternative proposal instead.

agree_strategy
Signal agreement with a given proposal regarding the strategy or work plan.

challenge_state
Reject a statement regarding the degree to which the current strategy or work plan has been worked through and make an alternative statement.

decide_strategy
Select one from among several alternative proposed strategies or work plans.

propose_strategy
Propose one or several alternative strategies or work plans.

propose_todo
Suggest that a certain work item will need to be taken care of later in the process.

disagree_strategy
Reject a given proposal regarding the strategy or work plan without making an alternative proposal.

agree_todo
Signal agreement with a statement saying that a certain work item will need to be taken care of later in the process.

universal concepts

explain_gap in knowledge
Verbalize that certain knowledge is not possessed by either member of the pair.

agree_gap in knowledge
Signal agreement with a given gap in knowledge.

explain_standard of knowledge
Explain or recapitulate one's own level of knowledge with respect to a certain topic.

ask_standard of knowledge
Ask the partner for his/her level of knowledge with respect to a certain topic.

ask_knowledge
Ask the partner for information of type 'declarative knowledge'.

stop_activity
Suggest to stop or abort the current HCI or HEI activity.

explain_finding
Verbalize a new insight; this includes interpreting an observed event.

propose_hypothesis
Formulate a hypothesis or conjecture, e.g. regarding a property of the program, or the environment.

explain_knowledge
Transfer information to the partner that is assumed to be correct declarative knowledge.

think aloud_activity
Verbalize aspects of one's own current HCI or HEI activity.

agree_finding
Signal agreement with a verbalized insight or interpretation.

agree_hypothesis
Signal agreement with a given hypothesis or conjecture.

agree_knowledge
Signal agreement (i.e. judge as correct) knowledge stated by the partner.

agree_activity
Signal agreement with all or part of the current HCI or HEI activity.

challenge_finding
Reject the content of a verbalized insight or interpretation and suggest an alternative one.

challenge_hypothesis
Reject a given hypothesis or conjecture and formulate an alternative one.

challenge_knowledge
Declare transfered knowledge as fully, partially, or potentially wrong by opposing it with one's own knowledge.

challenge_activity
Reject all or part of the current HCI or HEI activity and suggest an alternative activity.

disagree_finding
Declare transfered finding as fully, partially, or potentially wrong without explaining why.

disagree_hypothesis
Reject a given hypothesis or conjecture.

disagree_knowledge
Declare transfered knowledge as fully, partially, or potentially wrong without explaining why.

disagree_activity
Reject all or part of the current HCI or HEI activity.

amend_finding
Extend a verbalized insight or interpretation without rejecting it.

amend_hypothesis
Extend a given hypothesis or conjecture without rejecting it.

amend_activity
Propose an extension to the current HCI or HEI activity.

facade concepts

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

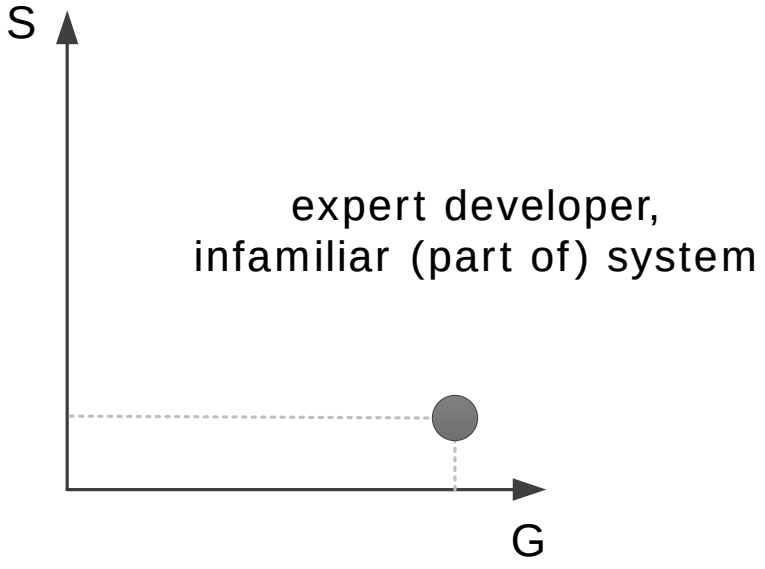
PP: How does it work? Research results so far

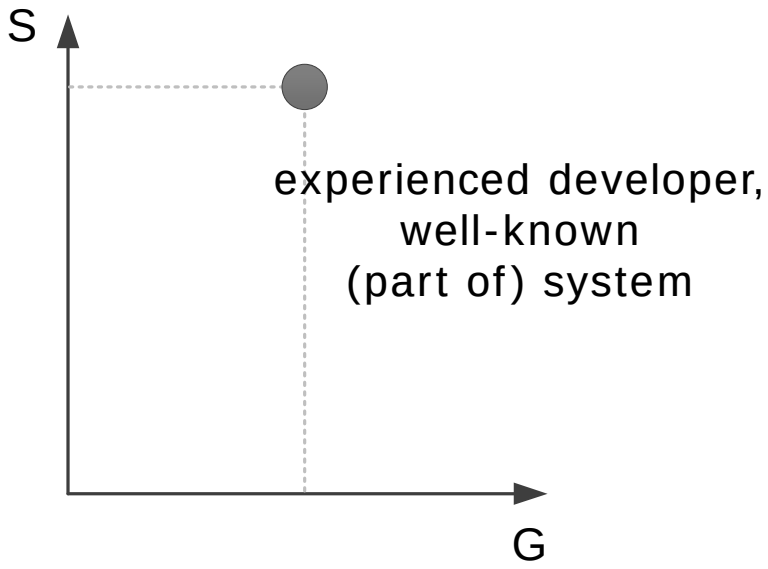
Freie Universität Berlin

- "Vocabulary" and "Patterns"
  - Base Activities (previous slide) as atoms of the PP process
    - roughly: decision making (process/product) and knowledge transfer
    - [SalPre13] Salinger, Prechelt: "Understanding Pair Programming: The Base Layer", 227 pages, 2013
  - 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 or alone)
    - 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
- next: look at the content of transferred knowledge

PP Session Dynamics: Two types of task-relevant knowledge

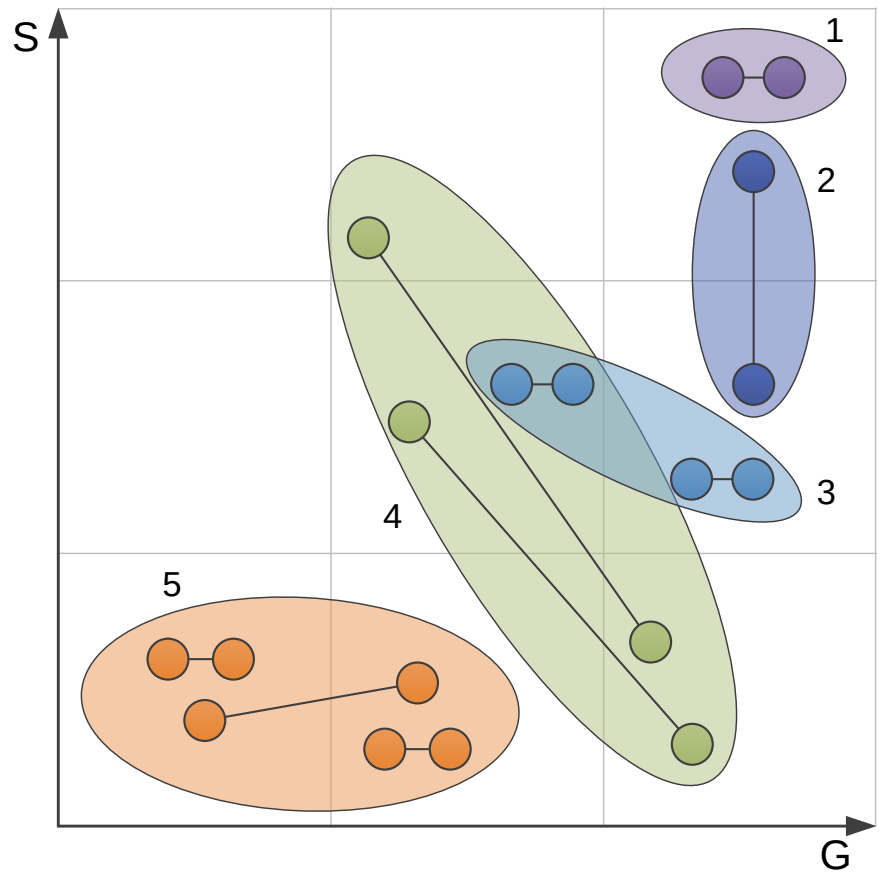
- SW development is knowledge-intensive work
  - programming languages, technology stacks, design patterns, …
  - coding styles, requirements, system architecture, …
- Necessary: knowledge that is relevant for the *current task*
- Two types of task-relevant knowledge:
  - **S**: specific knowledge about the software system
  - **G**: generic knowledge about methods and technology





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)
  - insofar it's 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)

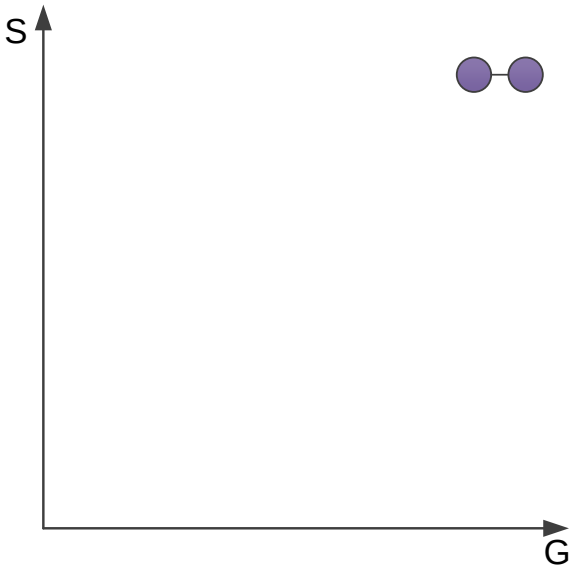
How to solve a problem as a pair?

- 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*
  - but some G is required once the system is understood
  - too-low G can be a problem (task is 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: lower standards for "high S and G"

Pair Constellations: Type 1 - No Relevant Gaps

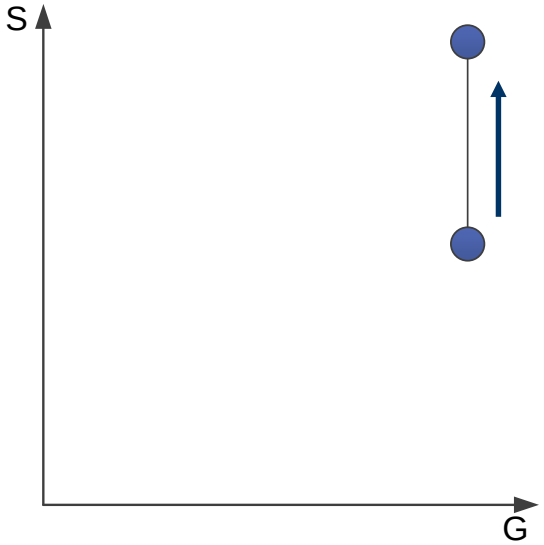
- Characterization:
  - Both developers understand the system completely and possess the programming skills to the solve the task.
- Occurence:
  - Rare, only if the pair recently worked on the same task together to build up high S.
  - Usually: *some part* of the system needs to be understood before the problem can be solved → other constellation



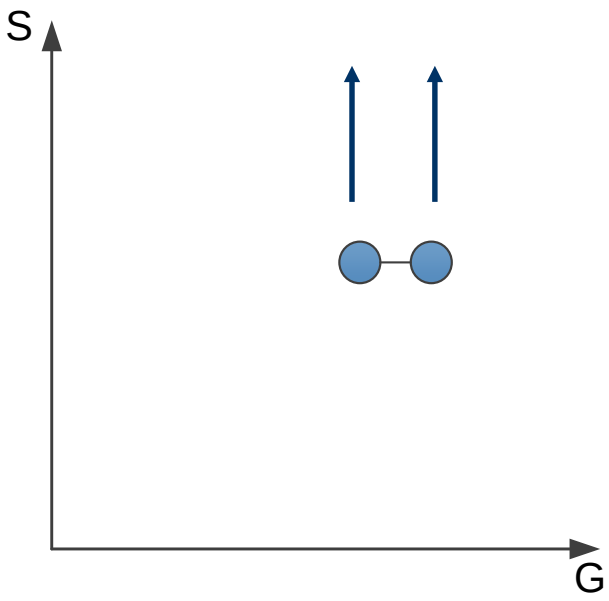
- Benefits (theoretical):
  - Modest, each developer could work on the task alone, and the task has but few opportunities to learn something.

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.
- Occurence:
  - Common, e.g.: Developer A started working on a task, B joins later
    → 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 (see [ZiePre14])

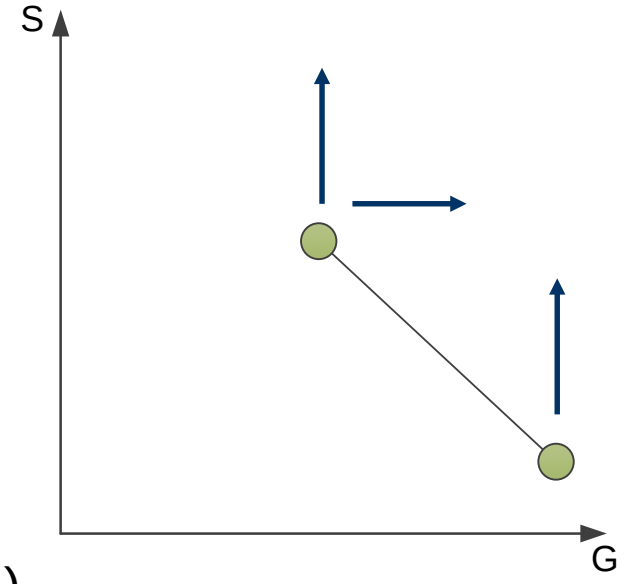


Pair Constellations: Type 3 - Collective S Gap



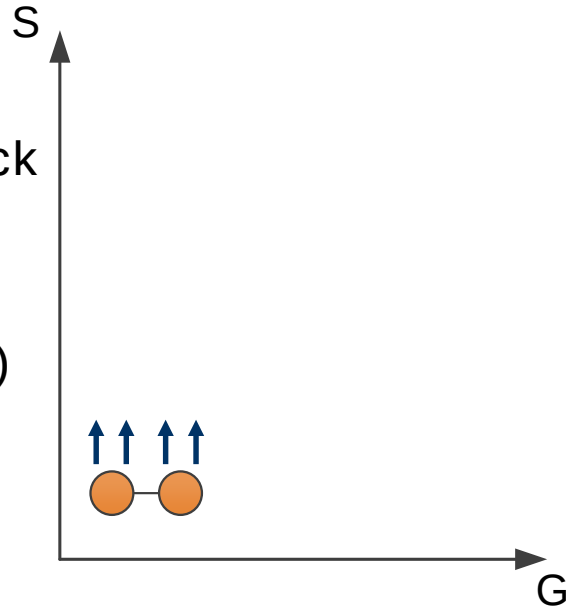
- Characterization:
  - Both developers lack relevant portions of system understanding, which needs to be built up in order to solve the task.
- Occurence:
  - Happens if the pair starts working on a new task together: Both need to find out which parts of the system are relevant.
- Challenges:
  - Usually, either of the two will have an insight first at times: Need to stay on the same page on their way to high S.
  - (But there are more ways for this than for the One-Sided S Gap)
- 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)

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.
- Occurence:
  - May appear uncommon, but:
  - 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

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.
- Occurence:
  - Happens: New technology (no G knowledge) and author unavailable (no S knowledge)
  - Should be avoidable on the team level
- Challenges:
  - G knowledge too low to acquire enough S knowledge.
  - For unexperienced pairs: having a partner might make it worse
- Solutions:
  - For this task: Different pair, or try alone
  - For this pair: Different task, or radically limit the scope

PP Session Dynamics: Summary

- **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

- Relative and absolute **S gaps** dominate PP session dynamics
  - Core difficulty: Reach high S *as a pair*
- **G knowledge** does not seem to play a big role, except for
  - Type 4, G-transfer opportunities, i.e. learning
  - Type 5, too little G, pair programming becomes impossible
- 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

PP: How does it work? Status and further work

Freie Universität Berlin

- Reasonably well done:
  - Develop vocabulary to describe the PP micro-process in general
  - Identify and describe behavioral patterns for knowledge transfer

- Open issues:
  - Identify behavioral patterns for decision making
    - for both of product and process (design vs. strategy)
  - Develop representation suitable for practitioners
  - Understand skillful use of different styles (side-by-side, mob programming, …)
  - …

Interested? Talk to us!

Related methods

Distributed Pair Programming (DPP):

- The partners are not physically in the same room and use a separate computer each
  - http://www.c2.com/cgi/wiki?VirtualPairProgramming
  - http://www.saros-project.org (Saros, see next few slides)
- Their interaction is supported by a collaborative editor and audio conferencing, perhaps also 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

Saros: Distributed Party Programming

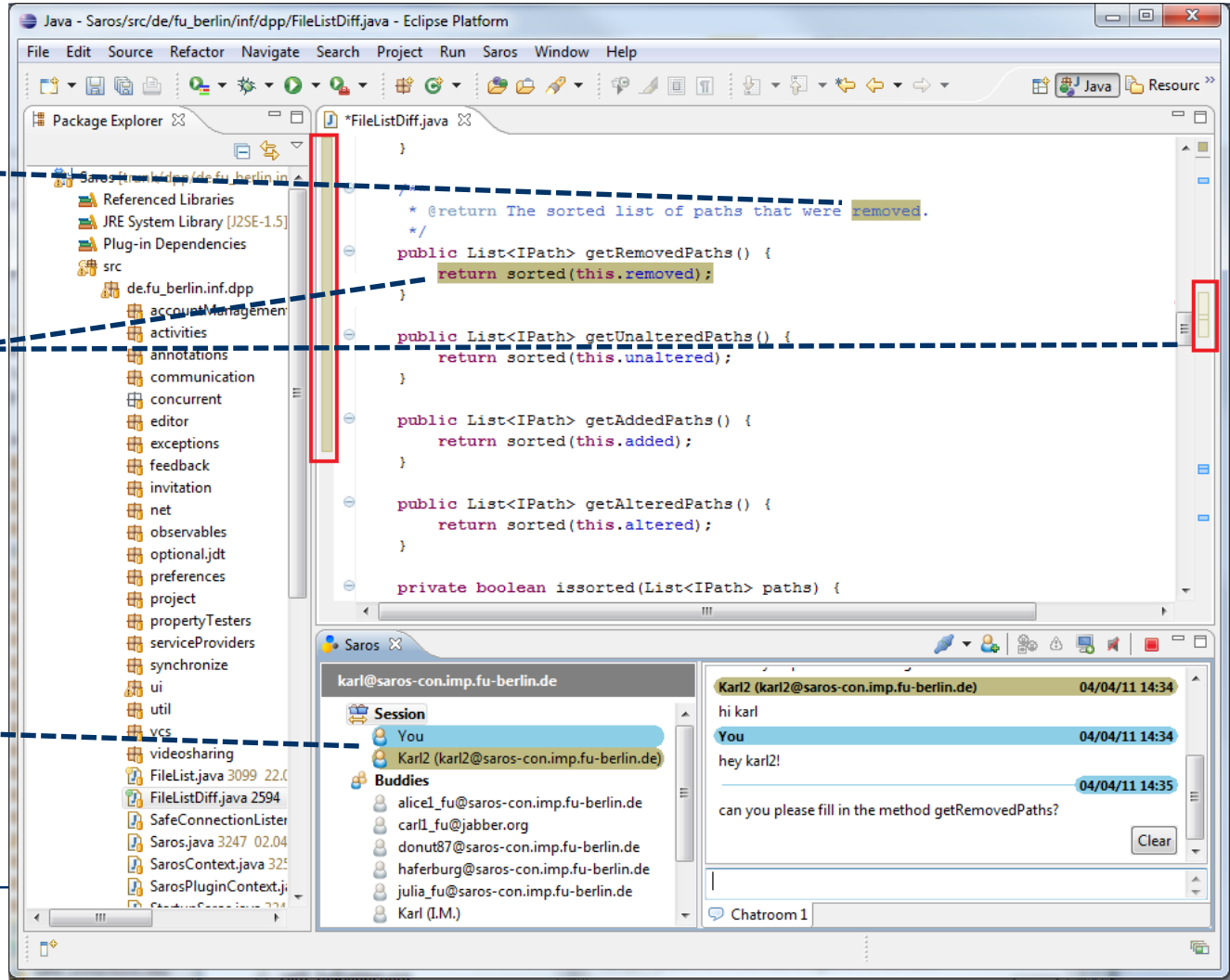
Freie Universität Berlin

- Saros is an IDE plugin (Eclipse, IntelliJ IDEA) that couples multiple IDEs remotely for real-time collaborative editing

Recently written by participant 2

View & highlight by participant 2

Session participants



Saros: Usage modes

- Supports multiple Observers
  - useful for joint reviews or for training newbies

- Supports multiple Writers
  - useful for **Side-by-Side programming**
  - perhaps also useful for **Distributed Party Programming**:
    - Multiple (say, 5) developers "hang around" in the same session, each doing their own development, and quickly helping each other out when needed
  - useful for **Distributed Pair Programming:**
    - First research result [SchPreSal14]: "Distributed-Pair Programming (DPp) is not just Distributed Pair-Programming (dPP)":
    - Capable pairs make judicious use of concurrent editing and viewing to make the process more fluent.

Saros: Research and Development

Freie Universität Berlin

- Plenty of research to be performed on DPP
  - Which developers accept this? When is it most useful?
    How to use it well? Usefulness for offshoring?

Interested?
Talk to us!

- Plenty of Saros development to be performed
  - Better multi-platform support, stand-alone version
  - Improve architecture, robustness, automated testing, …

Interested?
Talk to us!

Summary

- Pair Programming (PP) is the joint production of artifacts by two equally active programmers with fully shared ownership
  - using one computer, often only one keyboard
  - can be extended to remote settings, loose coupling, N persons
- PP has a number of potential benefits
  - for productivity, correctness, design quality, concentration, information flow, learning, truck factor
  - Empirical results regarding productivity are somewhat mixed
  - Empirical evidence regarding the other benefits is scarce, because measuring them is very difficult
- Much more research is needed
  - Describe and operationalize PP skill (via behavior patterns)
  - Must use real industrial pairs working on real tasks
  - Both local and distributed settings

Thank you!