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"*

Yes.

Absolutely not.

If all goes well.

Irrelevant.

Recipe for failure.

Yes.

If all goes well.

Laurie Williams

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

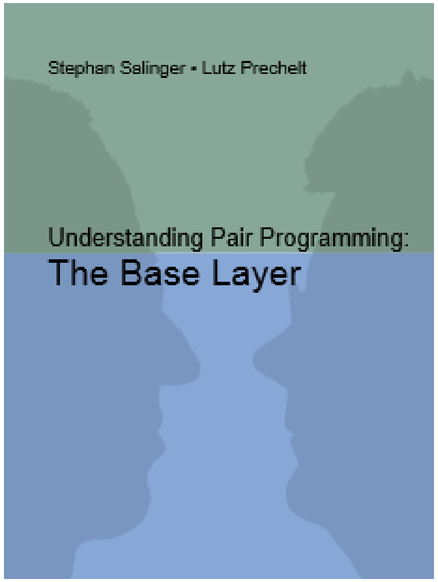
Lecture structure

- 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

arXiv:2002.03121v5 [cs.SE] 15 Feb 2021

PP-ind: Description of a Repository of Industrial Pair Programming Research Data

Franz Zieris
zieris@inf.fu-berlin.de
Freie Universität Berlin
Berlin, Germany

Lutz Prechelt
prechelt@inf.fu-berlin.de
Freie Universität Berlin
Berlin, Germany

Abstract—**PP-ind is a repository of audio-video-recordings of industrial pair programming sessions. Since 2007, our research group has collected data in 13 companies. A total of 57 developers worked together (mostly in groups of two, but also three or four) in 67 sessions with a mean length of 1:35 hours. In this report, we describe how we collected the data and provide summaries and characterizations of the sessions.**

Contents

1. Introduction

Pair programming (PP) is a software development practice in which two developers work closely together on a technical task on the same computer. It was popularized by Kent Beck who sees it as the central practice of eXtreme Programming and describes it as *"a dialog between to people trying to simultaneously program (and analyze and design and test) and understand together how to program better"* [2, p. 100].

Controlled experiments on pair programming have shown mere tendencies in terms of effects on quality and effort with much variation left to be explained [3]. In the words of the authors of a large experiment with almost 300 hired consultants: *"we are still far from being able to explain why we observe the given effects"* [1].

Our research group has been collecting industrial pair programming sessions since 2007. We record pair programming as it happens "in the wild" in order to understand how it actually works and what really matters in everyday practice. In particular, we record the pairs' converstation, their screen content, and a webcam video showing their gestures and posture.

This kind of data data is amendable to different types of analyses. We describe our qualitative approach in [11], [12]. In this report, we describe the technicalities of how we collected the data and provide some metadata for each session. Several researchers have contributed a lot of time to collecting and processing that data, and we want to give credit. The raw data itself cannot be released to the public because of non-disclosure agreements with the respective companies. As a proxy, we characterize the companies, the developers, and their PP sessions.

This report is structured as follows: We discuss our fundamental approach to collecting empirical data on pair programming (Section 2) and describe our generic data collection protocol (Section 3). We introduce some terminology and describe the structure of our data (Section 4). We give an overview of our repository (Section 5) and then discuss the individual contexts and cases (Section 6). We close with a discussion of the properties and limitations of our data collection (Section 7) and an overview of which data has been used in which publications so far (Section 8). In Appendix A, we explain the technical details of how we record and process PP sessions.

We provide repository meta-data, partial transcripts, questionnaires, and additional material as a public data set [20].

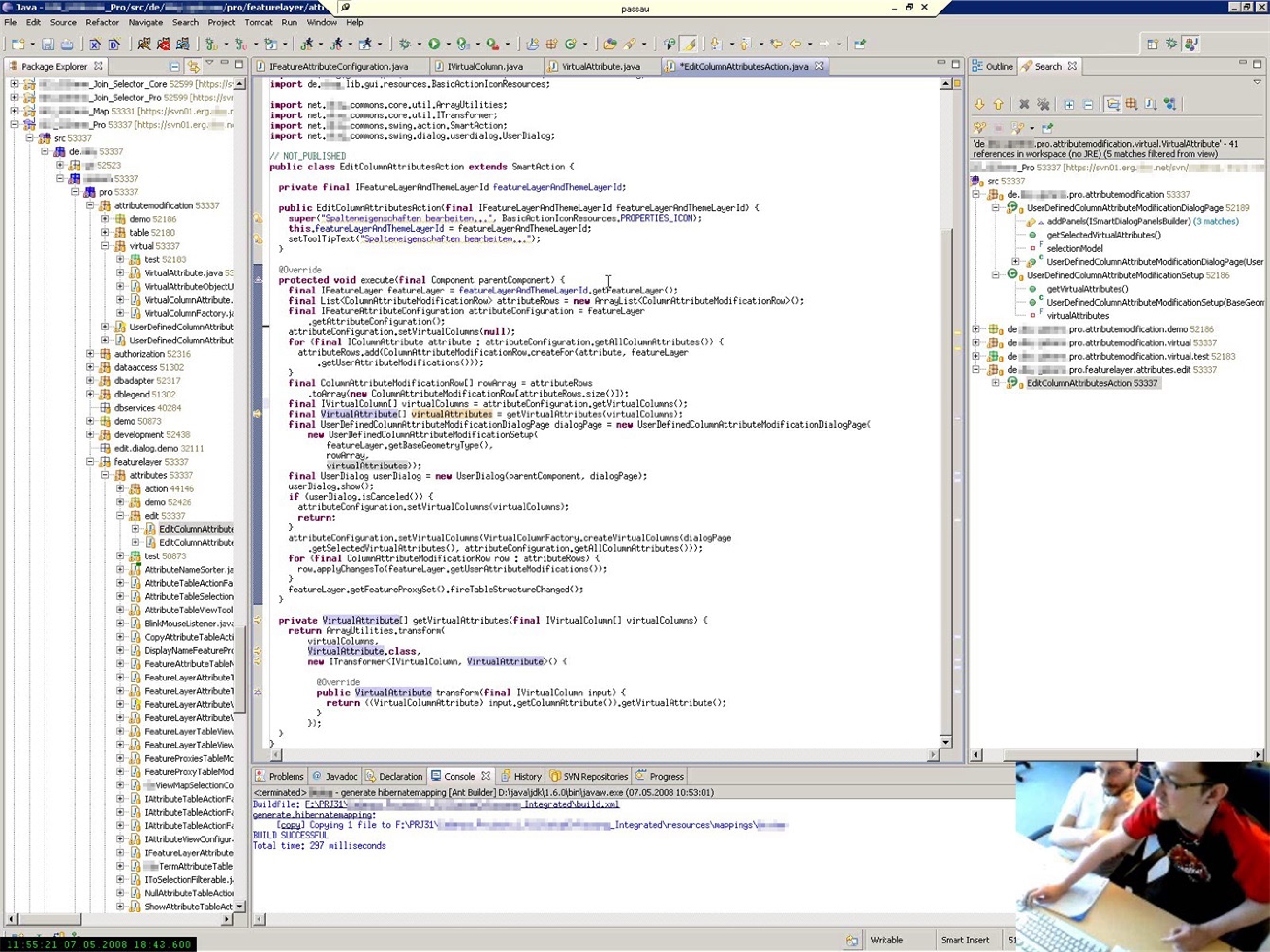
PP: How does it work?
AG SE research approach

Freie Universität Berlin

- Basic idea: Look <u>into</u> 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)



Java - _Pro/src/de/ /pro/featurelayer/attr
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.j
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
EditColumnAttribut
EditColumnAttribut
test 50873
AttributeNameSorter.ja
AttributeTableActionFa
AttributeTableSelection
AttributeTableViewTool
BlinkMouseListener.java
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
Writable
Smart Insert
11:55:21 07.05.2008 18:43.600

PP: How does it work?
AG SE 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 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

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'.
- **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.
- **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.
- **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.
- **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.
- **amend_finding**: Extend a verbalized insight or interpretation without rejecting it.
- **amend_hypothesis**: Extend a given hypothesis or conjecture without rejecting it.

facade concepts

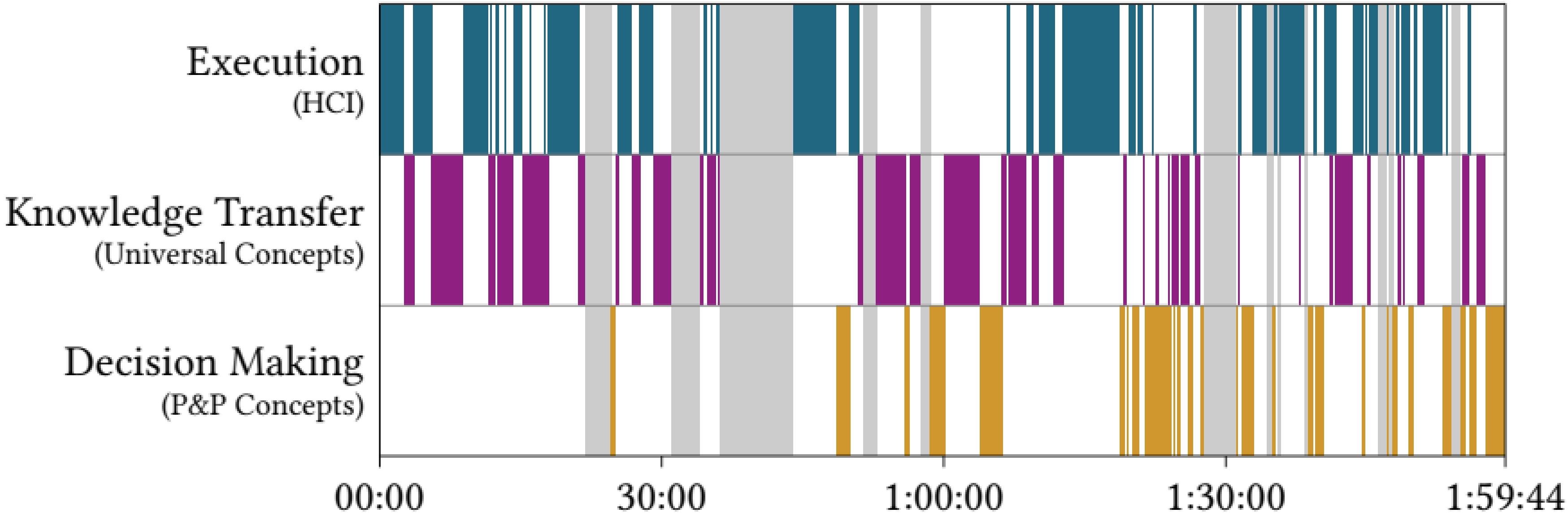
- **stop_activity**: Suggest to stop or abort the current HCI or HEI activity.
- **think aloud_activity**: Verbalize aspects of one's own current HCI or HEI activity.
- **agree_activity**: Signal agreement with all or part of the current HCI or HEI activity.
- **challenge_activity**: Reject all or part of the current HCI or HEI activity and suggest an alternative activity.
- **disagree_activity**: Reject all or part of the current HCI or HEI activity.
- **amend_activity**: Propose an extension to the current HCI or HEI activity.

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

Vocabulary/Patterns: How much of what is going on?

"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):



[Zieris20, Figure 4.1, p.141]

gray: interruptions, pauses, external help

Patterns: Knowledge transfer episode structure

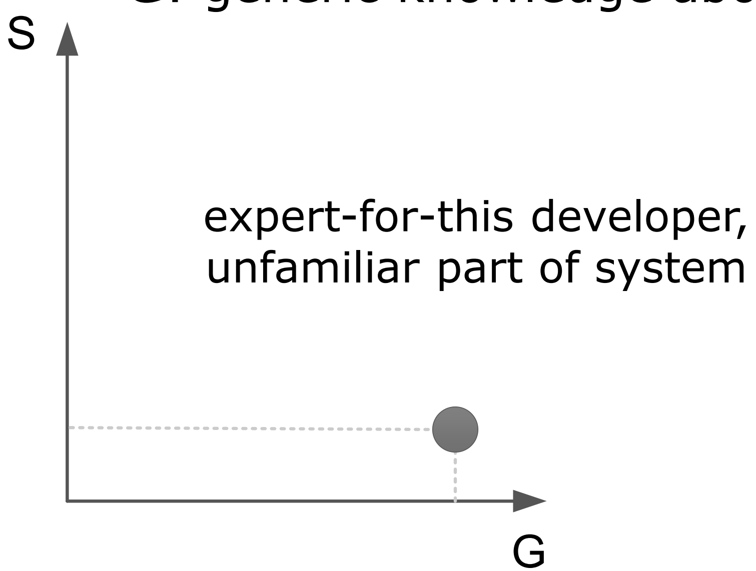
Knowledge Transfer: happens in per-topic episodes

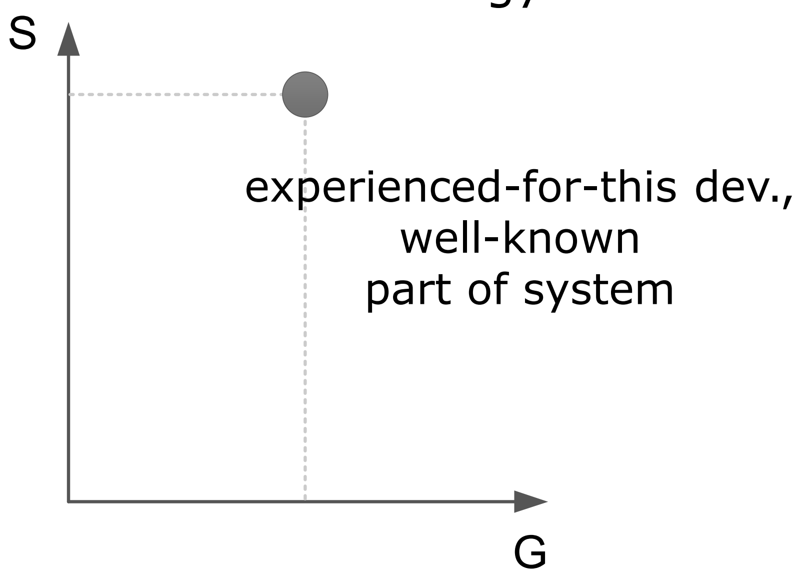
- driven by knowledge need, pursued by one of the developers
- different modes:
  - **Pull**: "asking" • **Push**: "explaining"
  - **Production**: generating new understanding
    - together: Co-Production
    - or alone: Pioneering Production
- some symptoms of good pairs:
  - one topic at a time, 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

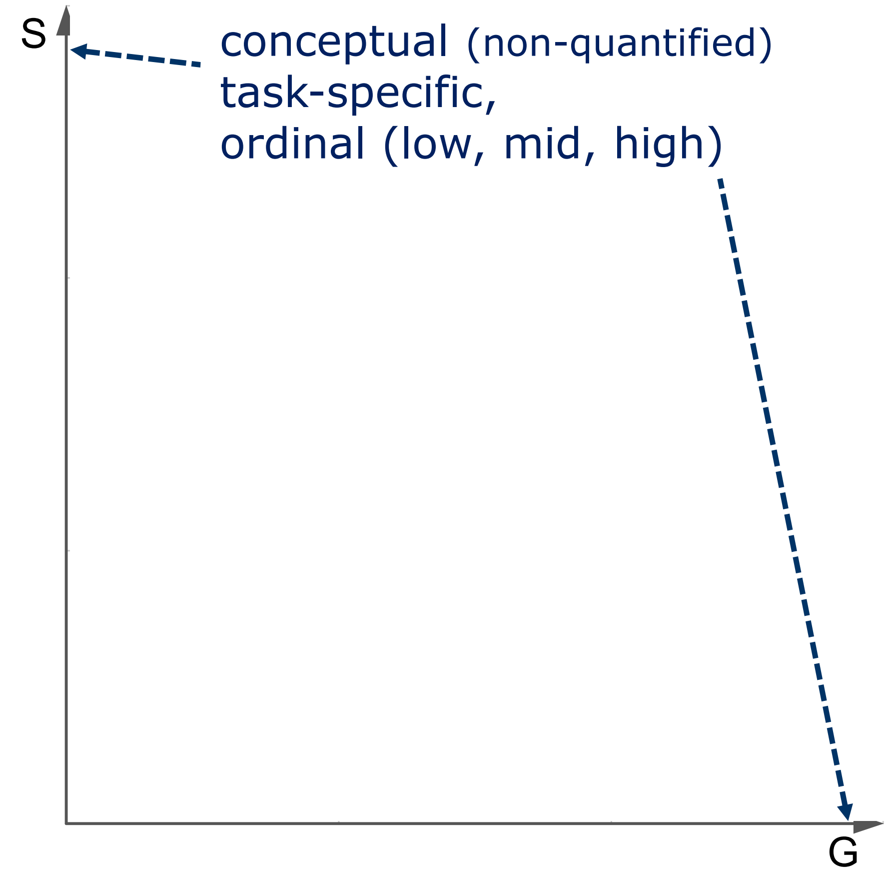
- SW development is knowledge-intensive work
  - programming languages, technology stacks, design patterns, …
  - coding styles, requirements, system architecture, …
- Two types of task-relevant knowledge [Zieris20]:
  - Zieris, Prechelt: "Explaining Pair Programming Session Dynamics from Knowledge Gaps",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





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?

Freie Universität Berlin

- 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]

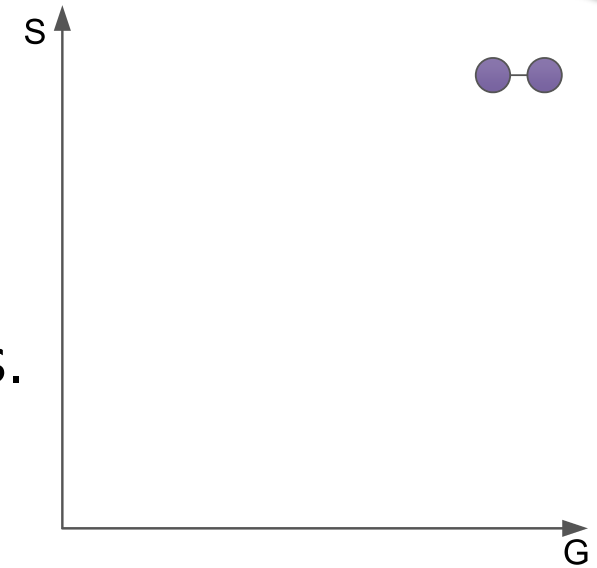
- Pairs must constantly maintain high **Togetherness**
  - joint system understanding (S)
  - joint ideas of how to develop SW (tools, methods)
  - joint tactical plan
  - no obstacles from workspace awareness or language barrier
- Pairs must pick appropriate transfer modes
  - Push, pull, co-produce,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

PP even needed?

- 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.

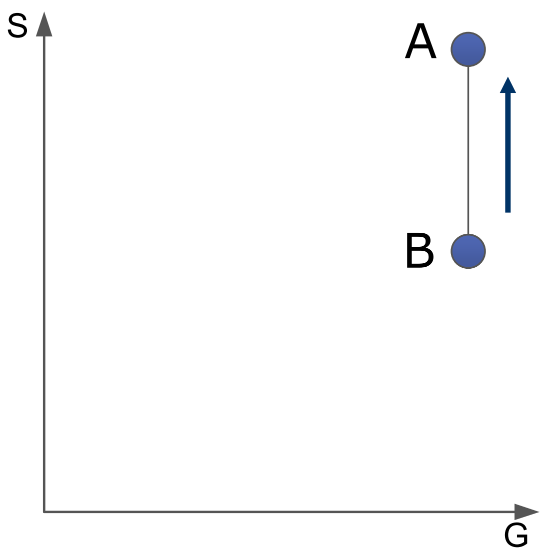


- Benefits (theoretical):
  - Modest, each developer could work on the task alone, and the task provides only few opportunities to learn something.
    - PP appears hardly needed.
    - May be useful if e.g. correctness is critical.

Session Dynamics Pair Constellations: Type 2 - One-Sided S Gap

Most typical case

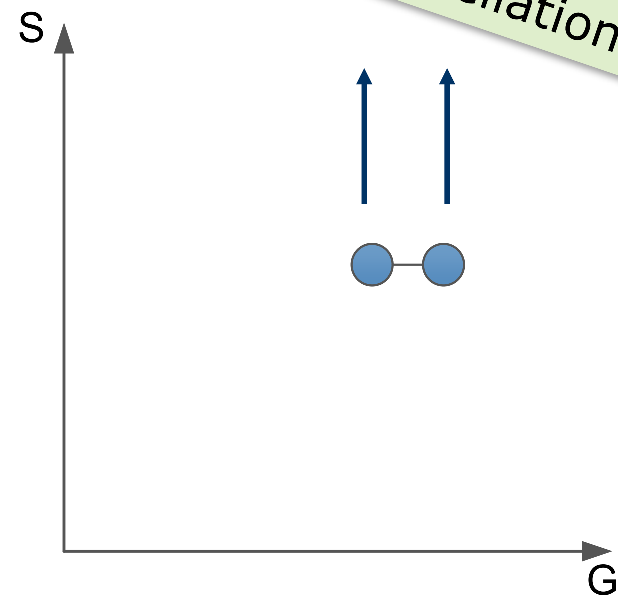
- Characterization:
  - One developer has an S-advantage that needs to be addressed if the two should work as a pair.
- Occurrence:
  - Common, e.g.: Developer A started working on a task, B joins later → A has S-advantage
- Challenges:
  - B might not be aware of the gap and might not understand A's 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]



Session Dynamics Pair Constellations: Type 3 - Collective S Gap

Nice constellation!

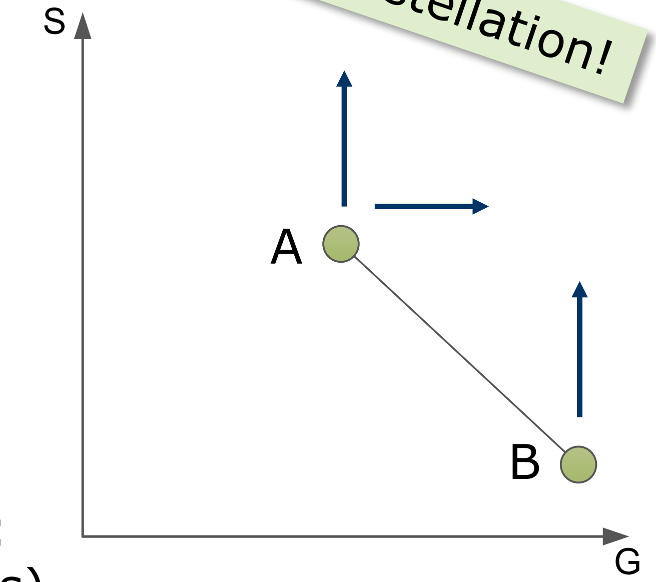
- Characterization:
  - Both developers lack relevant portions of S. Pairs needs to build up S to solve the task.
- Occurrence:
  - The pair starts on a new task together: Both need to find out which parts of the system are relevant.



- 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

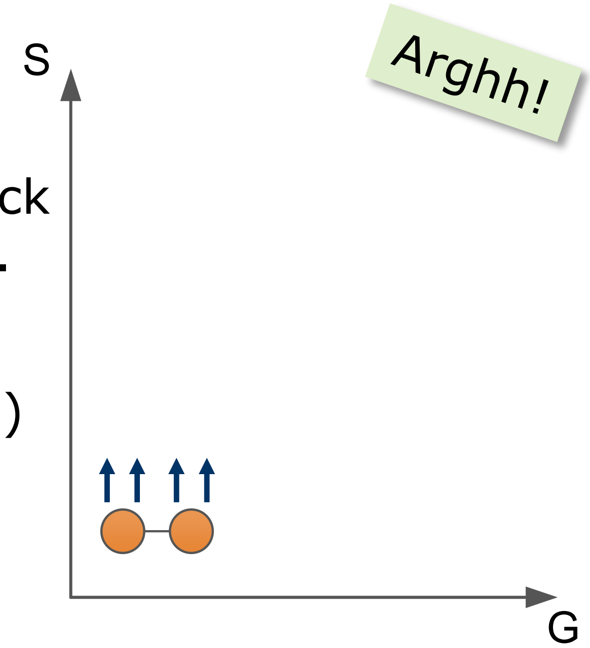
Nice constellation!



- 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)

Freie Universität Berlin

- 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

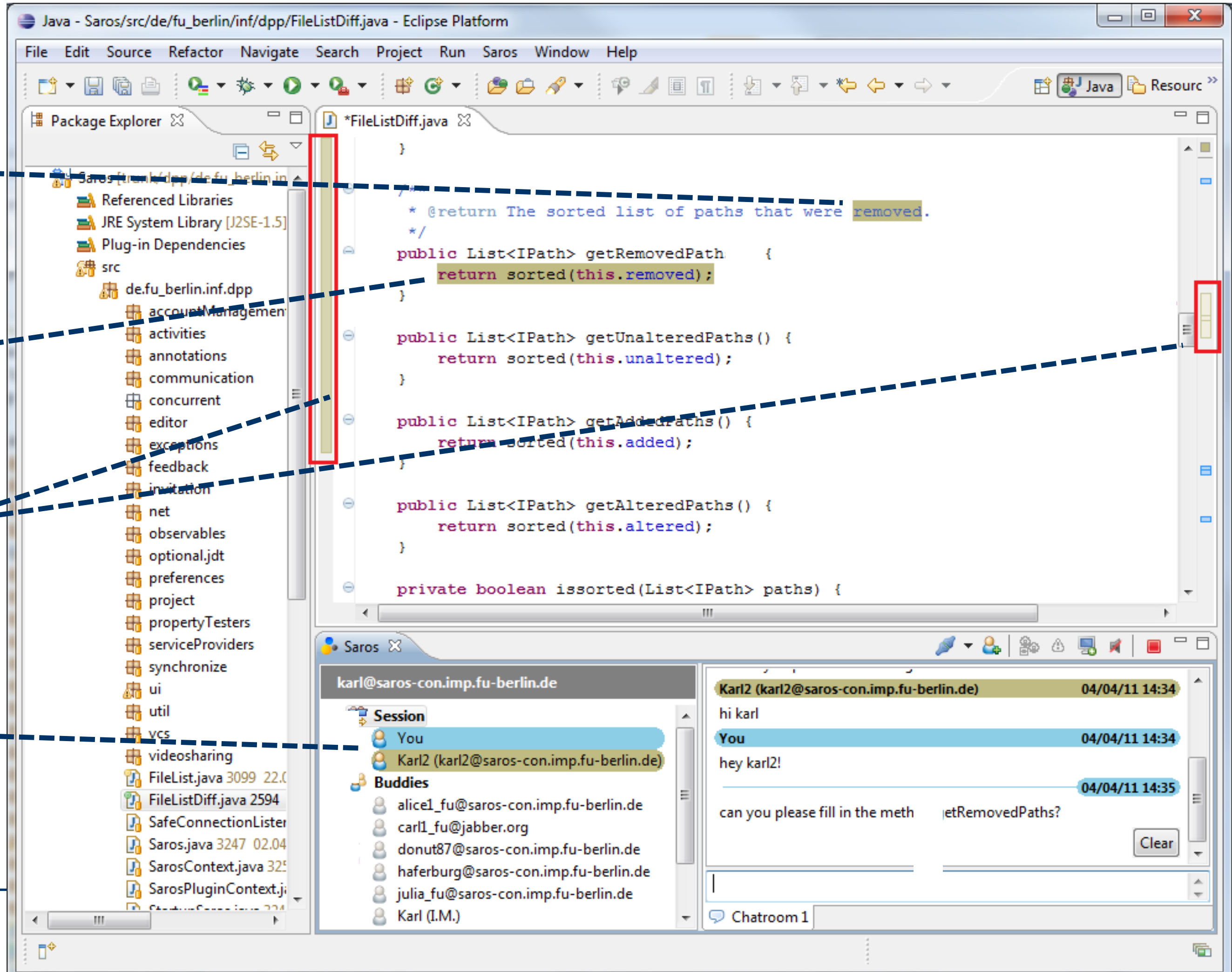
- Saros is an IDE plugin that couples multiple IDEs remotely, syncs local files, and creates remote workspace awareness:

Recently written by participant 2

Highlight by participant 2

Viewport of participant 2

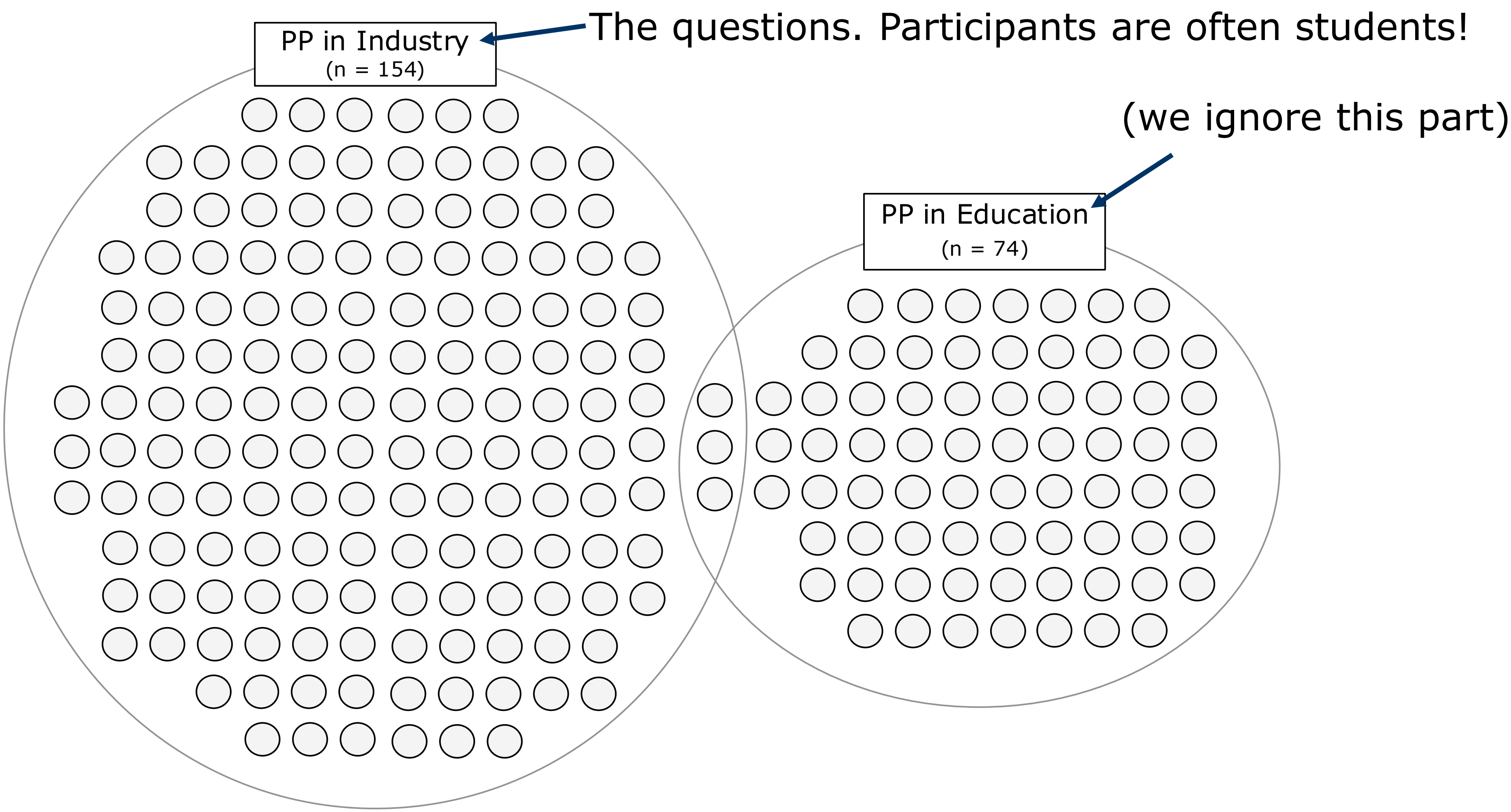
Session participants



What about other PP research?

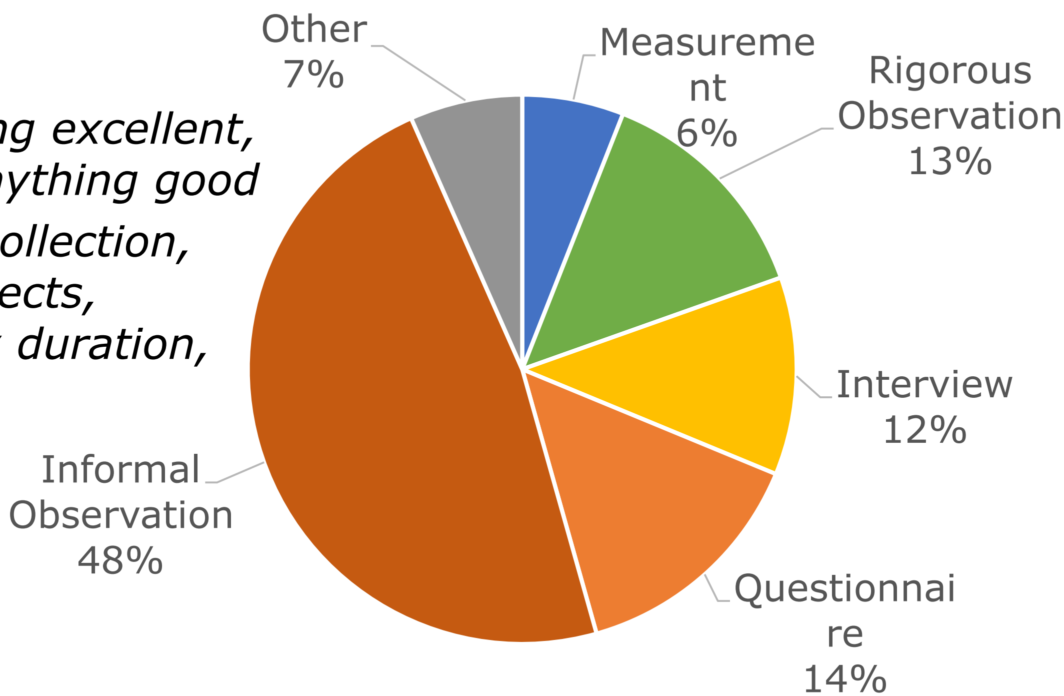
Two big secondary studies:

- Industrial-settings view [VanMan13] vs. education view [SalMenGru11]



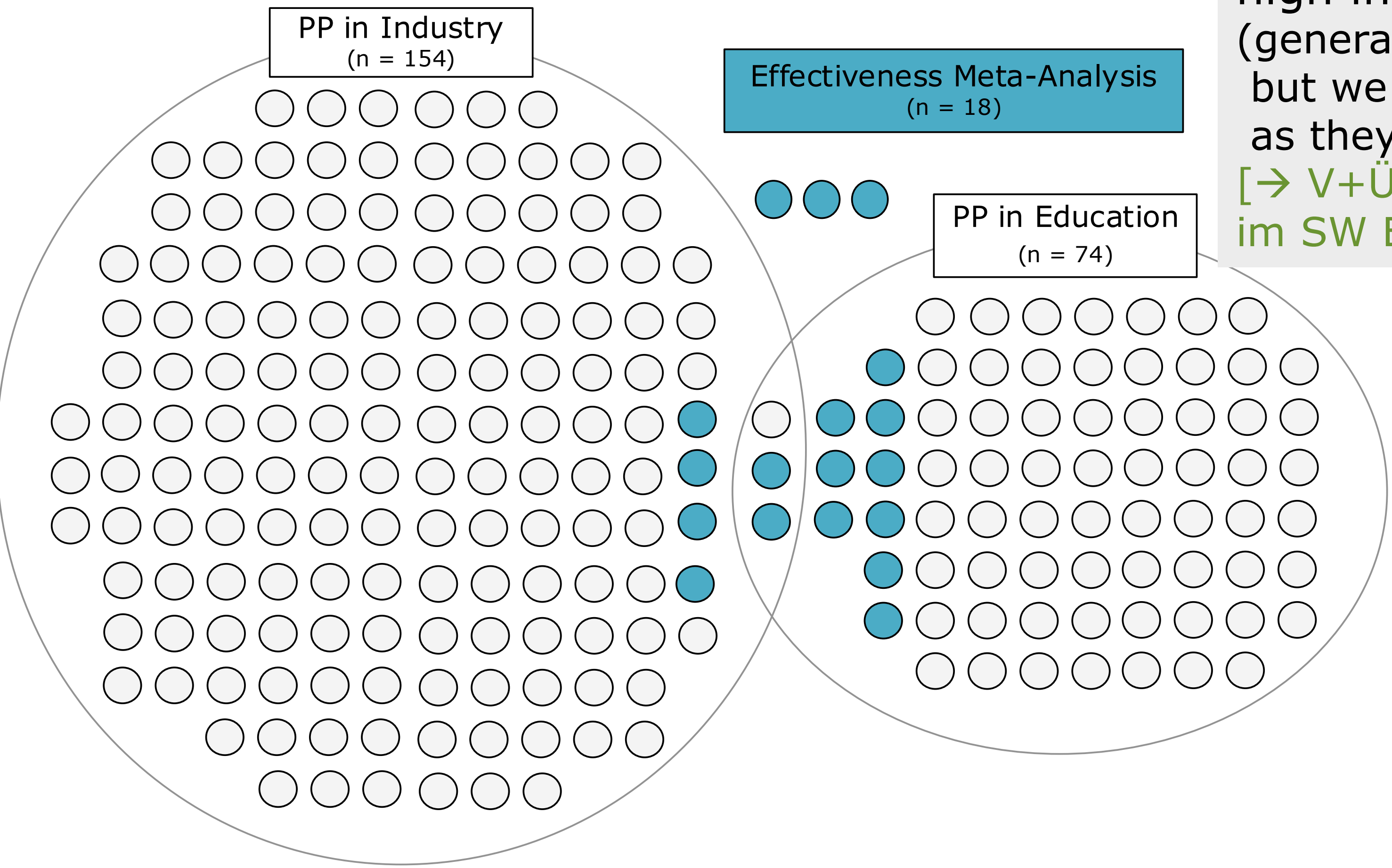
Research methods used, Quality

- *"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
      - *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, …*



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

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

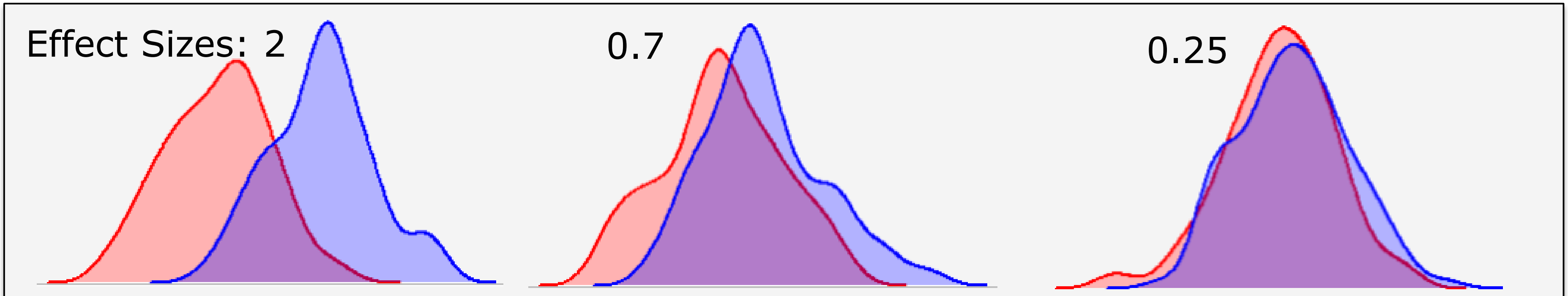


Experiments provide high internal validity.
(generalization unclear, but we believe the results as they stand.)
[→ V+Ü Empir. Methoden im SW Eng., SoSe]

Is PP faster?
Does it produce better quality?

Freie Universität Berlin

- **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



(fabricated data, for illustration only)

Problems of quantitative black-box perspectives

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
  - No idea how many pairs used PP well

?

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

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

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

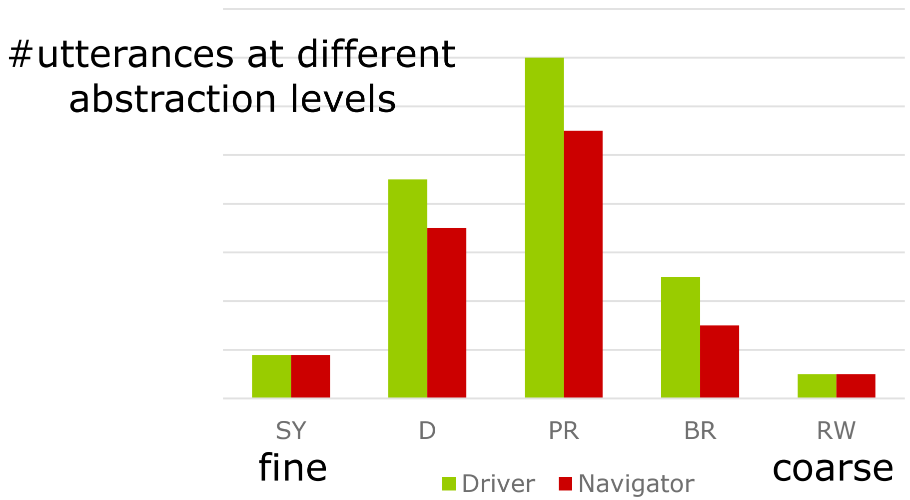
PP influence on motivation?

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**



- 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!

https://dilbert.com/strip/2004-10-20