Agile Offsharing: Using Pair Work to Overcome Nearshoring Difficulties

Prof. Dr. Lutz Prechelt
Institut für Informatik
Freie Universität Berlin

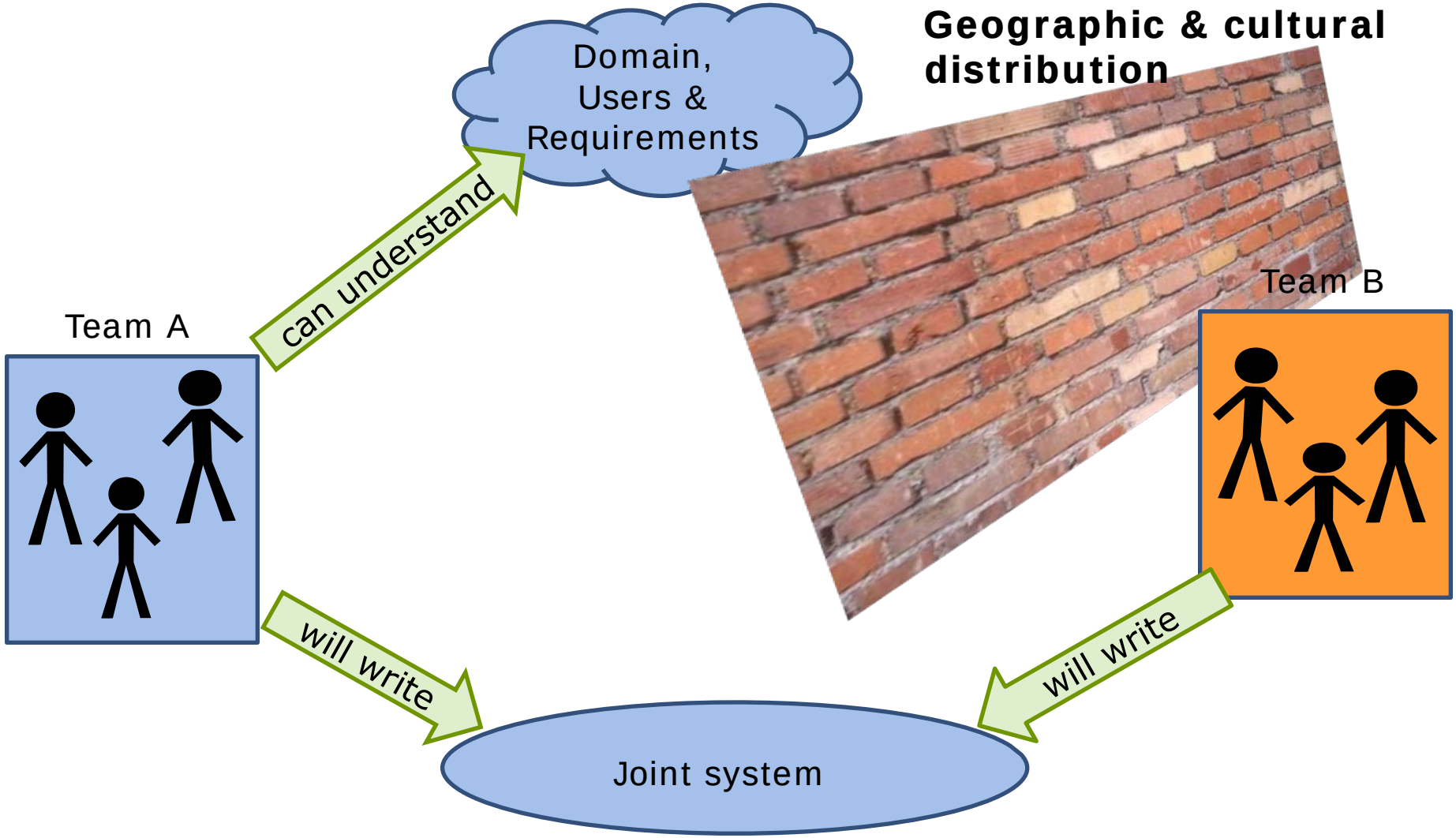
Outline

- Pair Programming: What and why?
- Agile Offsharing: Why, what, how?
- "But we are not using offshoring!"

Pair Programming

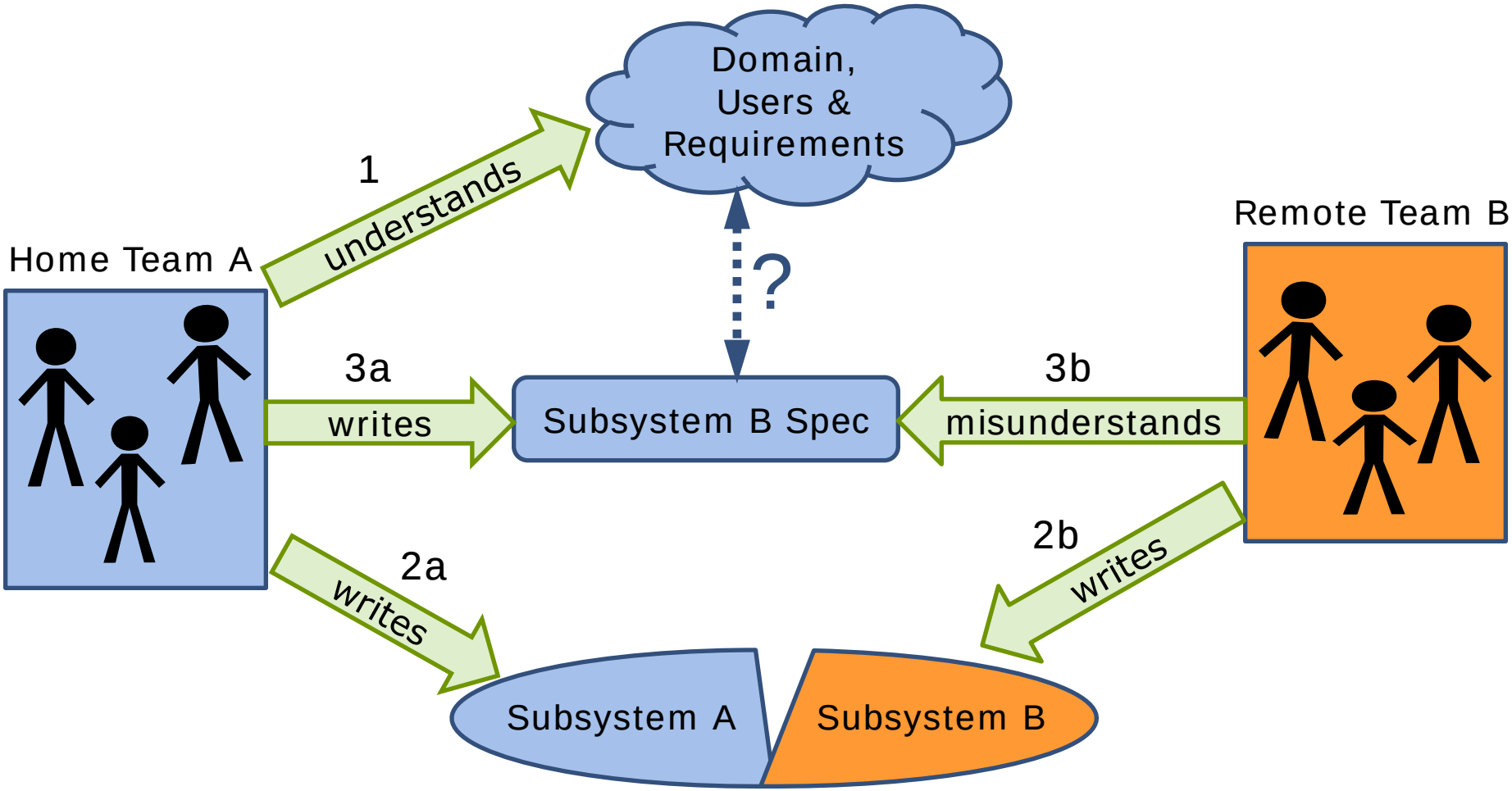
- Claimed advantages:
  - Faster than solo
  - Fewer defects
  - Better software design
  - More concentrated work
  - 2 people familiar with code
  - Can employ joint knowledge
  - Faster learning
  - Enjoyable
- Use it for:
  - Shorter time-to-market
  - Higher quality
  - Reducing code knowledge bottlenecks
  - **Knowledge transfer**
- Disadvantages/Issues:
  - May require more resources
  - Some programmers do not like it

Agile Offsharing: Assumed situation/problem



Waterfall model thinking

Belief: "Modularization helps!"



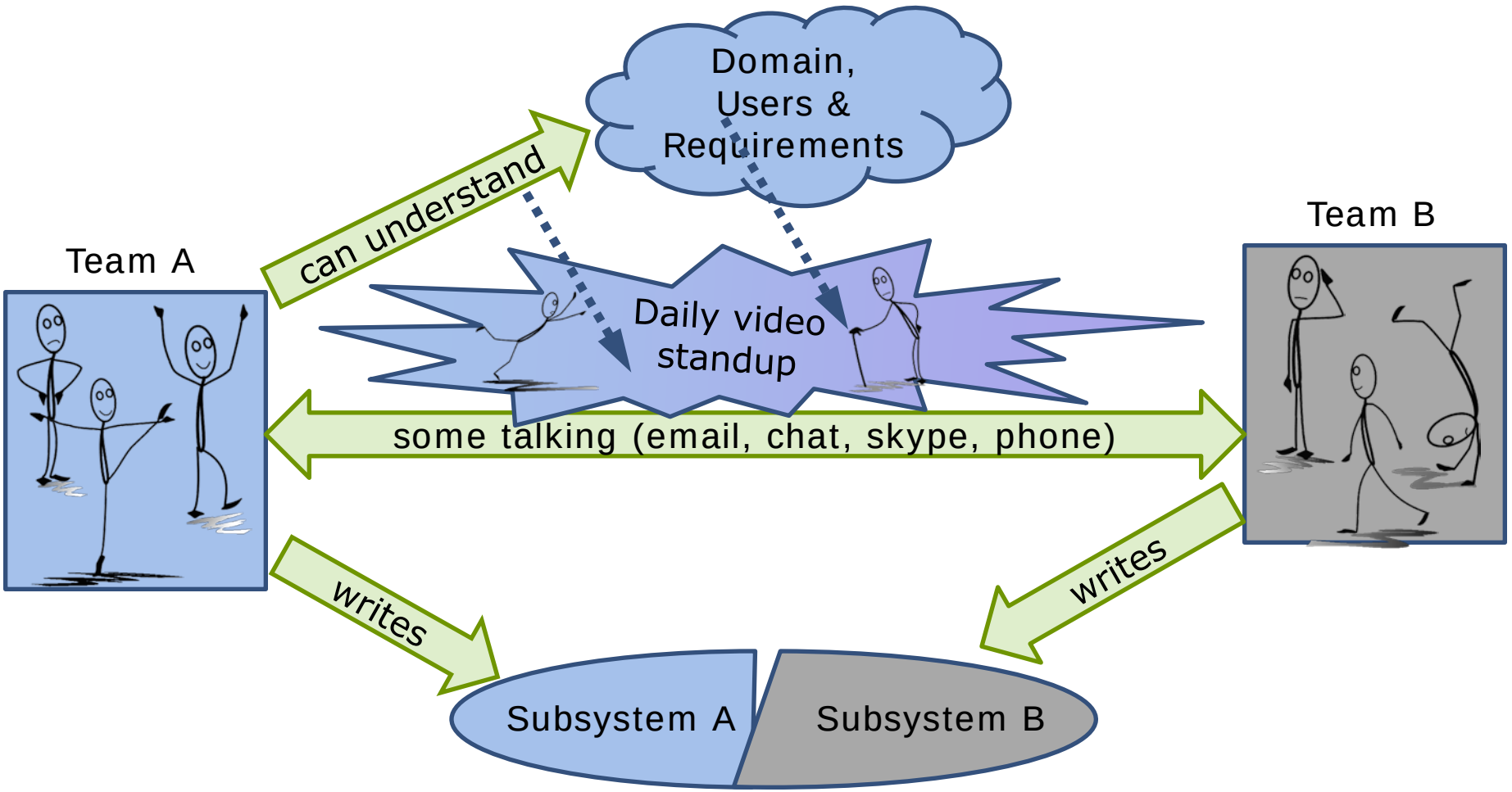
Why misunderstandings happen

Freie Universität Berlin

We tend to underestimate what needs explanation

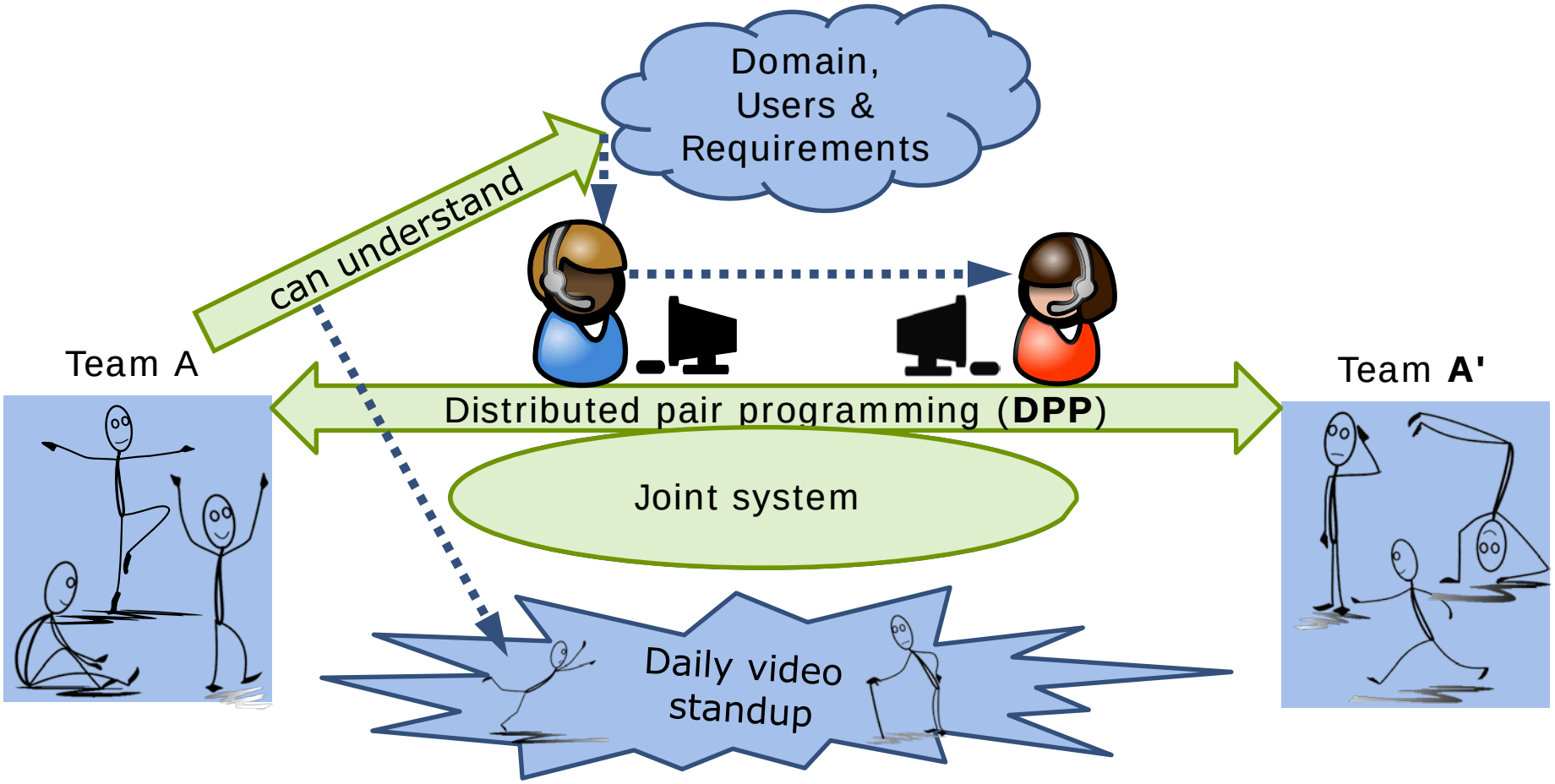
The Agile process illusion

Belief: "Modularization plus talking helps"



Radical idea: Agile Offsharing

Belief: "Only *close* collaboration works"



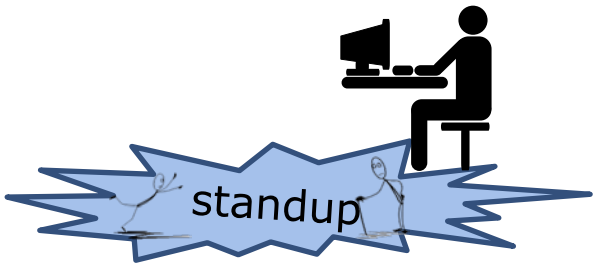
Agile Offsharing: The radical step

Freie Universität Berlin

Again:

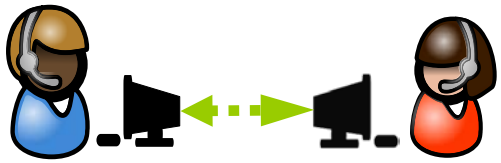
The new idea of Agile Offsharing is the following

- If cross-site communication is difficult ← it always is!



- do <u>not avoid</u> it
  - i.e. don't go for technical work **locality**

- rather, maximize it
  - i.e. *voluntarily* introduce **cross-site** technical work



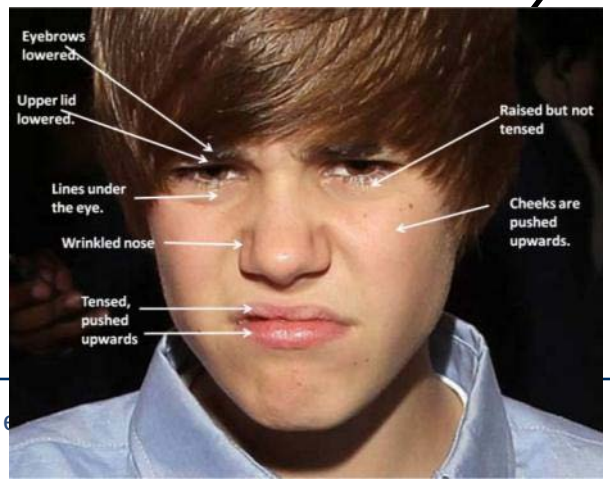
Questions so far?

Preconditions for using Agile Offsharing

Freie Universität Berlin

Technical / organizational:

- Small time zone difference
- Technical communication infrastructure
  - Strong network connection
  - Joint repositories
  - Distributed-IDE tools
  - Audio, perhaps video equipment
- Pair-work-ready workplace



Not ideal!

Social:

- Collaboration readiness:
  - One-team thinking
    - Non-competitive attitude
    - Transparency & trust
    - Team stability
  - Willingness for pair work
  - Compatible development processes
- Collaboration skills:
  - Strong skills in a joint natural language
  - Recognizing gaps in requirements understanding
  - Requirements explanation skills

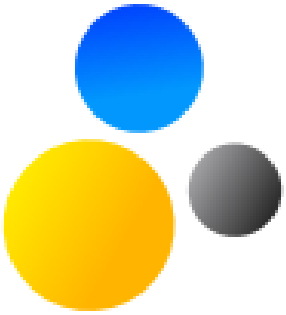
Establishing Agile Offsharing (1/4): People

- Find pairing volunteers
  - at least two on each side
  - strong language and communication skills
  - preferably strong technical skills
- Do not push anybody

Photograph: Corbis RF / Alamy/Alamy

Establishing Agile Offsharing (2/4): Technology

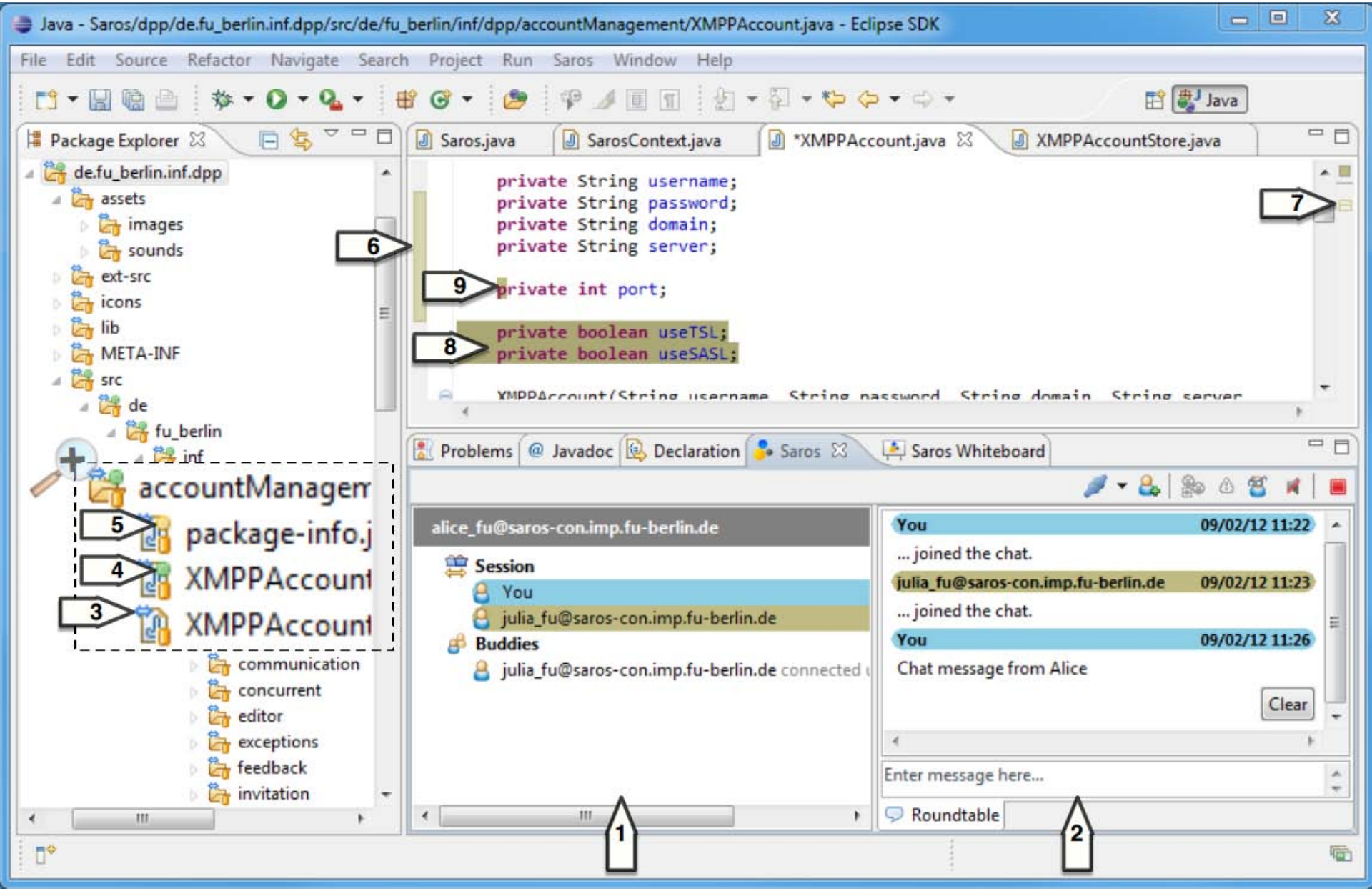
- Establish distributed editing:
  - Eclipse: Saros
  - Visual Studio: VS anywhere
  - Mac OS X: SubEthaEdit
    - editor only
  - IntelliJ IDEA: -
  - Start with local trials at both sites
  - then set up an official server, firewall rules, etc.
- Invest in comfortable, high-quality headsets
- Find handy, reliable audio software, e.g.:
  - SIP VoIP, Skype
  - Phone conferencing systems
  - mumble



Saros

Eclipse plugin for distributed pair programming: Saros

Freie Universität Berlin



Establishing Agile Offsharing (3/4): Tasks & Getting Going

- Create list **K**: Reqirements knowledge to be transfered
  - identify an expert
  - estimate explanation time
    - should be 0.2 to 2 hours each
  - note closest-match volunteer
- Somewhat like story cards

- Create list **T**: Development task
  - list eligible developers
  - list required knowledge pieces
  - estimate required time
    - should be 1 to 5 hours each
- Find **K-T** matches
  - that have suitable volunteer pairs
- Arrange and execute sessions
  - Consciously perform them for knowledge transport

Establishing Agile Offsharing (4/4): Refine

Freie Universität Berlin

- Learn how to arrange sessions with little effort
- Learn how to cope with conflict
  - e.g. detect and repair power imbalances
- Learn how to cope with language skill limitations
- Learn how to optimize knowledge transport
  - Reflection session after DPP session
  - Better task selection
  - Draw in more volunteers

"But we are not using offshoring!"

- Are your relevant colleagues always available locally?
- The pairing idea also applies if the knowledge is not about requirements:
  - remote technology expert
  - remote usability expert
  - remote expert for existing code
  - remote partner for
    - design discussion
    - debugging

Furthermore:

1. Saros sessions can have up to 5 participants

2. Side-by-side programming:
   - "Hanging around" in the same Saros session
   - short bursts of pairing as needed

We call the resulting behaviors *"Distributed Party Programming"* (DPP)

Some questions of mine

- Who has done distributed pair programming:
  - via screen sharing?
  - via Saros (or similar)?
- Who wants to try out Distributed Party Programming?
- Who wants to try out Agile Offsharing?
- We are available for consulting or research cooperation

- Some questions of yours?

Thank you!

Agile Offsharing: Current research status

Freie Universität Berlin

- Agile Offsharing idea:
  - Various experts all found Agile Offsharing plausible
- Distributed Pair Programming (DPP):
  - We have seen and recorded isolated cases
  - There appears to be plenty of Saros use in the wild
- An EU research proposal is under review
  - for 3 years in-vivo research
- Partners from
  - Germany, Netherlands, Italy
    - research institutions and companies
  - Austria, Bulgaria, India, Italy, Lithuania, Spain, Switzerland, Ukraine
    - companies

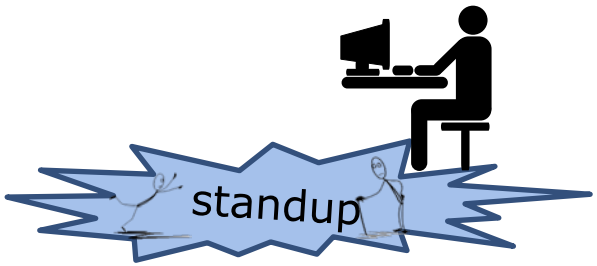
Agile Offsharing: Make a radically new assumption

Freie Universität Berlin

Context:

- Distributed software development, but with agile methods
- Only the "home" team builds the requirements knowledge

- Traditional assumption:
  Transfer reqs knowledge explicitly, but maximize **locality for technical work**



- Agile Offsharing assumption:
  Must transfer reqs *during* technical work or else bandwith is too low
  - therefore, introduce **cross-site technical work**