Controlled Experiments for the empirical evaluation of programming language constructs: type systems as an example

Stefan Hanenberg
University of Duisburg-Essen, Germany

London, UK, 27.02.2012

Motivation

- Two different audiences for PL research
  - Machines
    - Execution speed, compilation speed, compile time errors, etc.
  - Human
    - Development speed, development errors, etc.
- Nowadays research methods mainly address first audience
- Usability of PLs plays rather minor role

Current situation in Empirical SE

- Theories mainly describe existence of a difference
- Theories typically do not try to quantify differences (for some good reasons)
  - ...empirical knowledge rather low
- Experimenter currently have to „invent situations for language constructs on their own“
  - Example: static type systems....

Long term goal

- Theories
  - Descriptions of situations where certain constructs dominate others (size of difference part of theory)
  - Large number of experiments that try to falsify theories
  - Example (very first initial step):
    - „When using an undocumented API, ...
      ....static typing reduces development time“

- General kind of theory:
  - „*When the code is of kind X, ...*
    ...the use of construct A leads to C
    ...which differs to construct B by factors...“

Conclusion so far...

- We want to do empirical studies with humans...

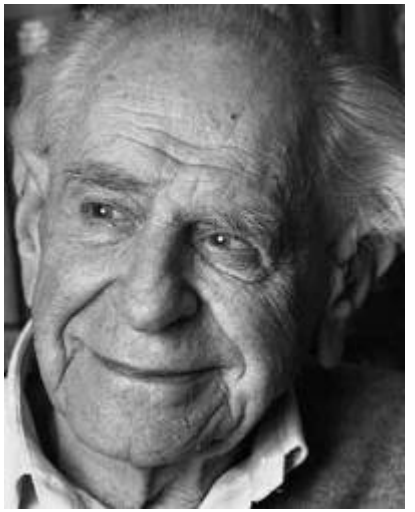
HOW?

Controlled Experiments (1)

- Scientific approach
  - Observation of singular events (sample)
    (e.g. developers using a dynamically/statically typed programming language)
    - Formulation of hypothesis
    - Identification of dependent / independent variables
      (e.g. development time depending on type system)
    - Construction of environment
      (IDEs, tasks, languages, machines, …)
  - Collection of subjects
  - Measurements (e.g. development time to solve a certain task)
  - Analysis (mainly inductive statistics)

Controlled Experiments (2)

- Scientific argumentation
  - Falsification of hypothesis (use of statically typed language decreases development time)
  - More often
    - Exploratory analysis (let‘s see what happens if…)
  - **NO PROOFS / NO GENERALIZABILITY**
    - But always the hope that repeated observations reveal some truth



Where to start?

- Relatively few textbooks available specific to software engineering

Where to start?

- Huge bunch of textbooks outside the domain of software engineering

  - Psychology
  - Social Sciences
  - Medicin
  - …

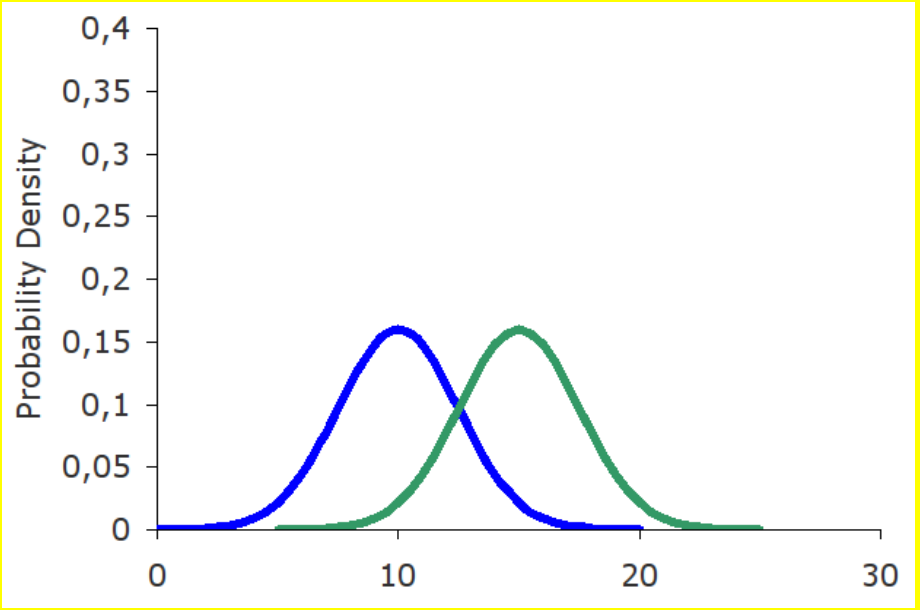
Problems in Experimentation

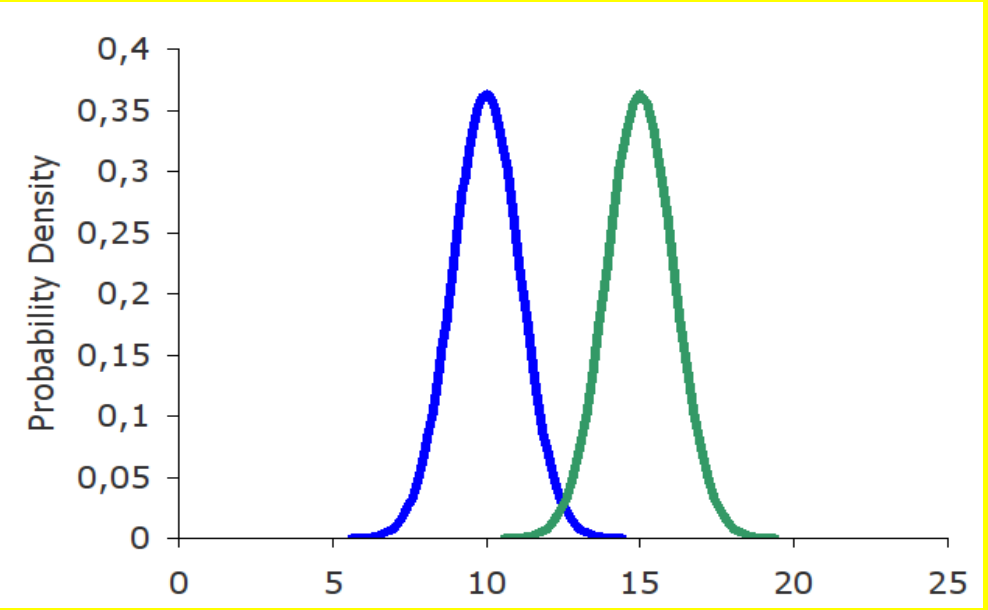
- Main Problem
  - Variability within/ among subjects
  - Huge bunch of possible (unknown) influencing factors
  - „*No measured effect*“ can always be the result of a rather inappropriate experiment setup
- Counteractions
  - Experimental Design
    - Within- / between subject design, Repeated measurement, Blockdesign, Latin Square, etc.
  - Task definitions

Problems: Experiment Design

- Comparison between two samples

Example 1: Same effect size, different deviation

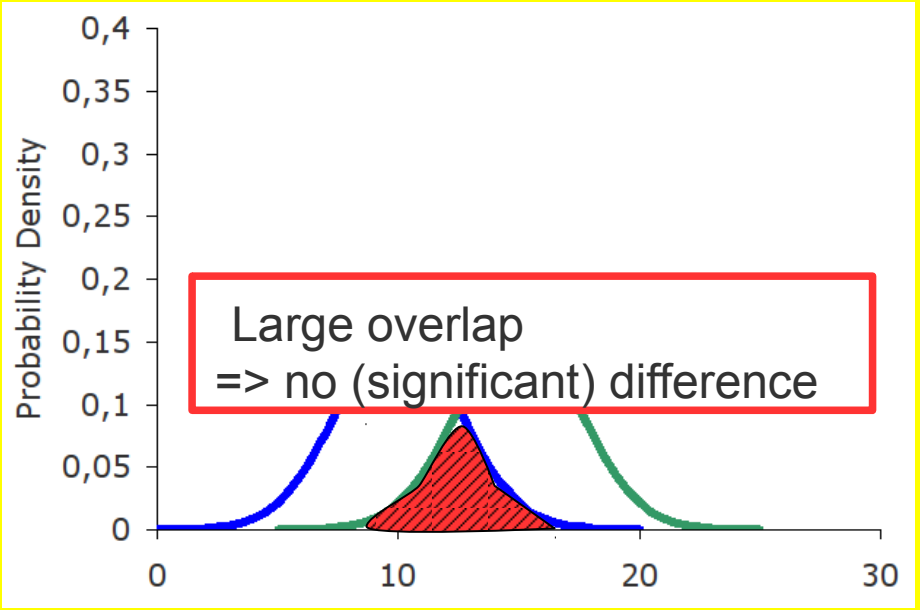


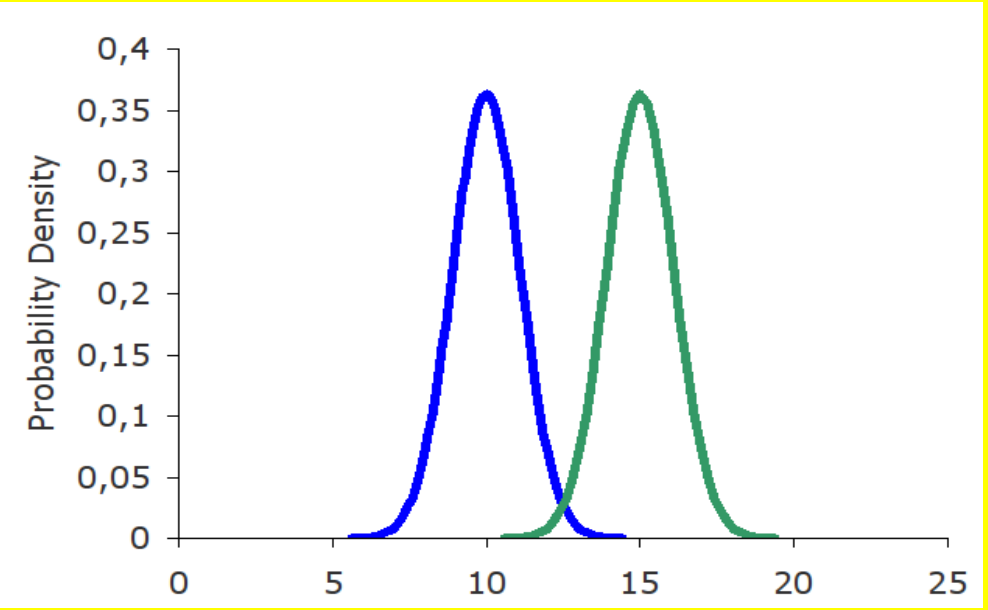


Problems: Experiment Design

- Comparison between two samples

Example 1: Same effect size, different deviation

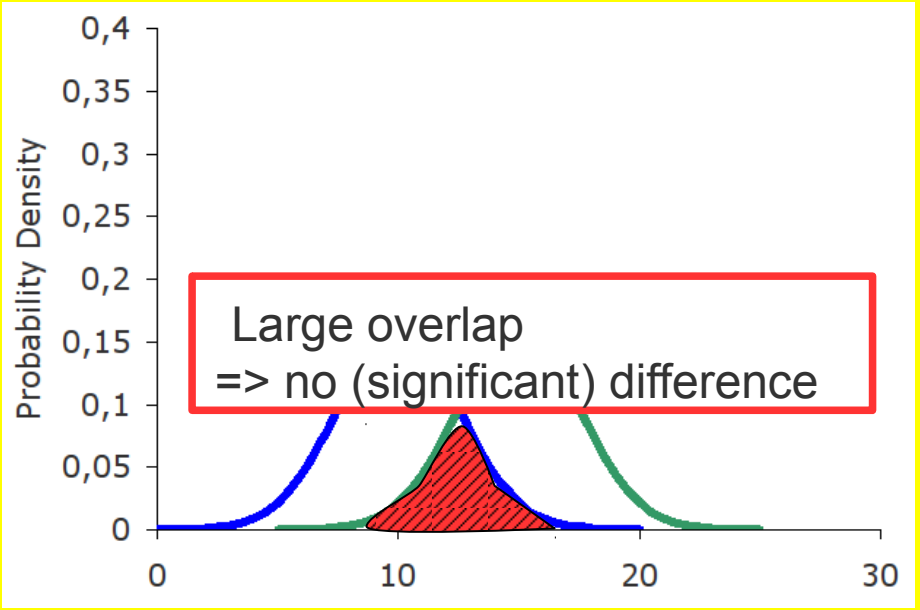


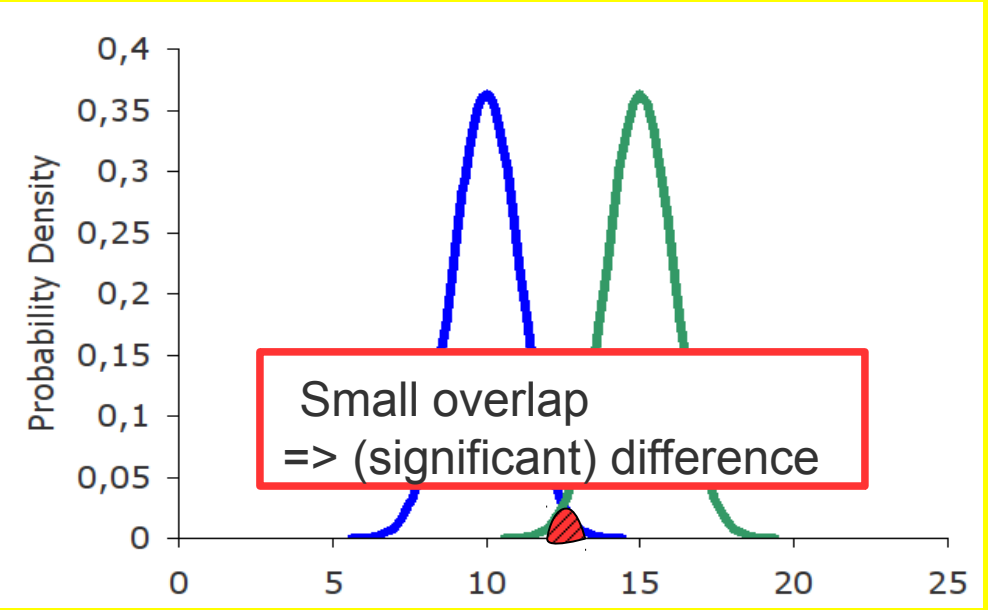


Problems: Experiment Design

- Comparison between two samples

Example 1: Same effect size, different deviation

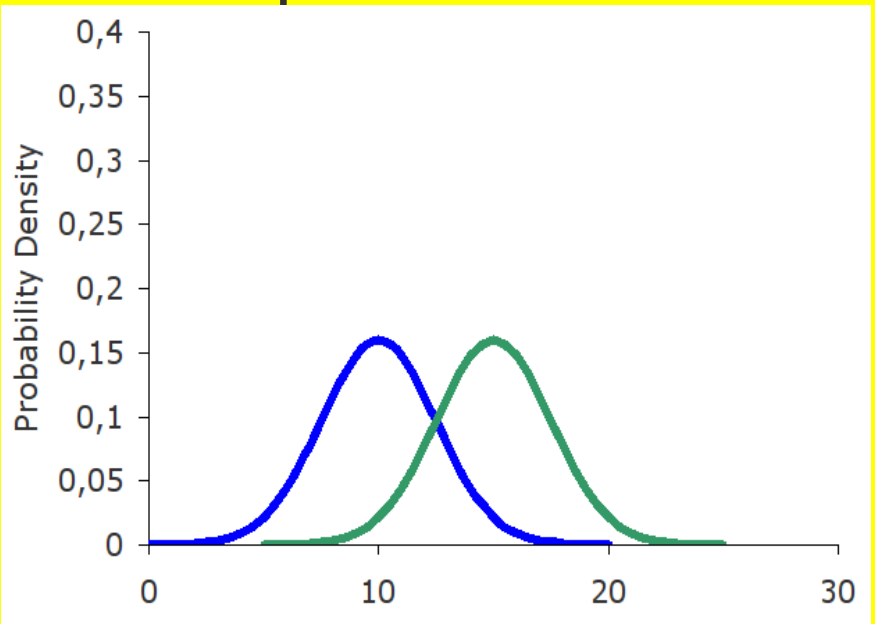


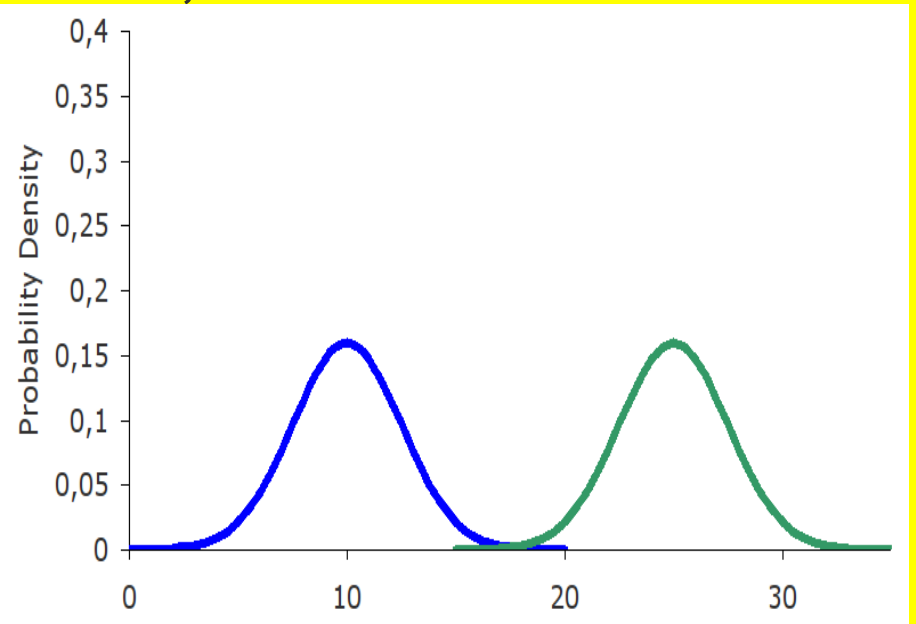


Problems: Experiment Design

- Comparison between two samples

Example 2: Different effect size, same deviation

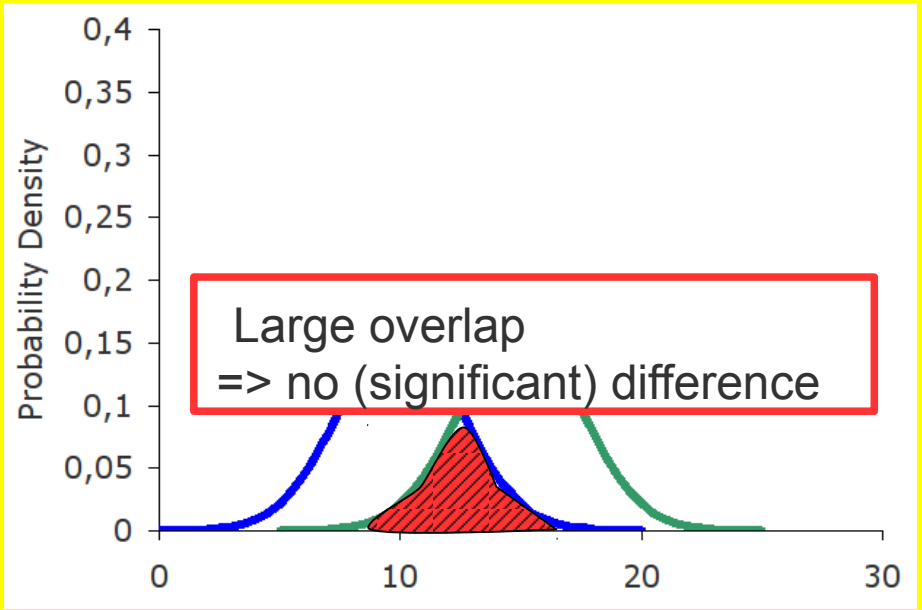


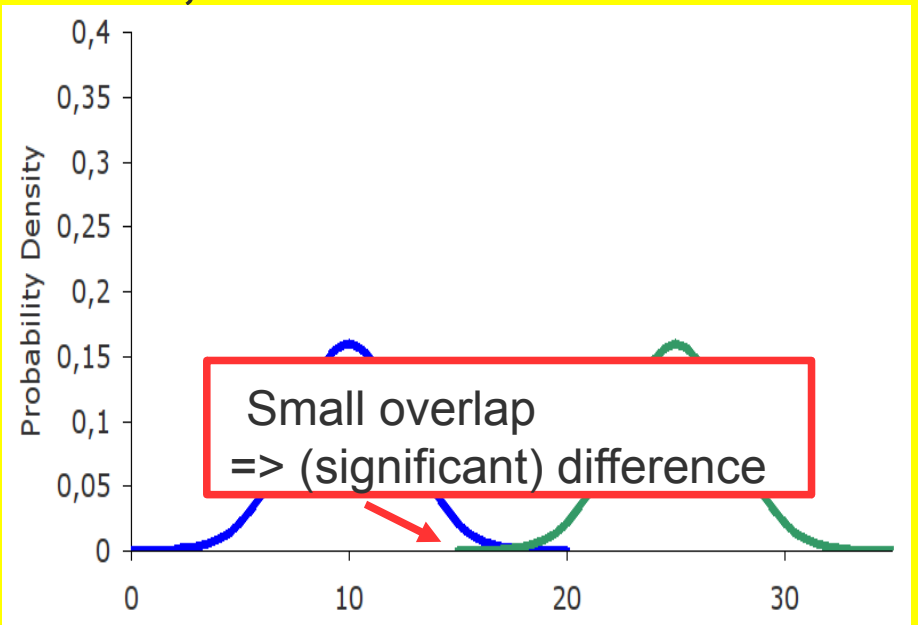


Problems: Experiment Design

- Comparison between two samples

Example 2: Different effect size, same deviation





Problem(s) in Experimentation

Conclusion

Experimenter should try to

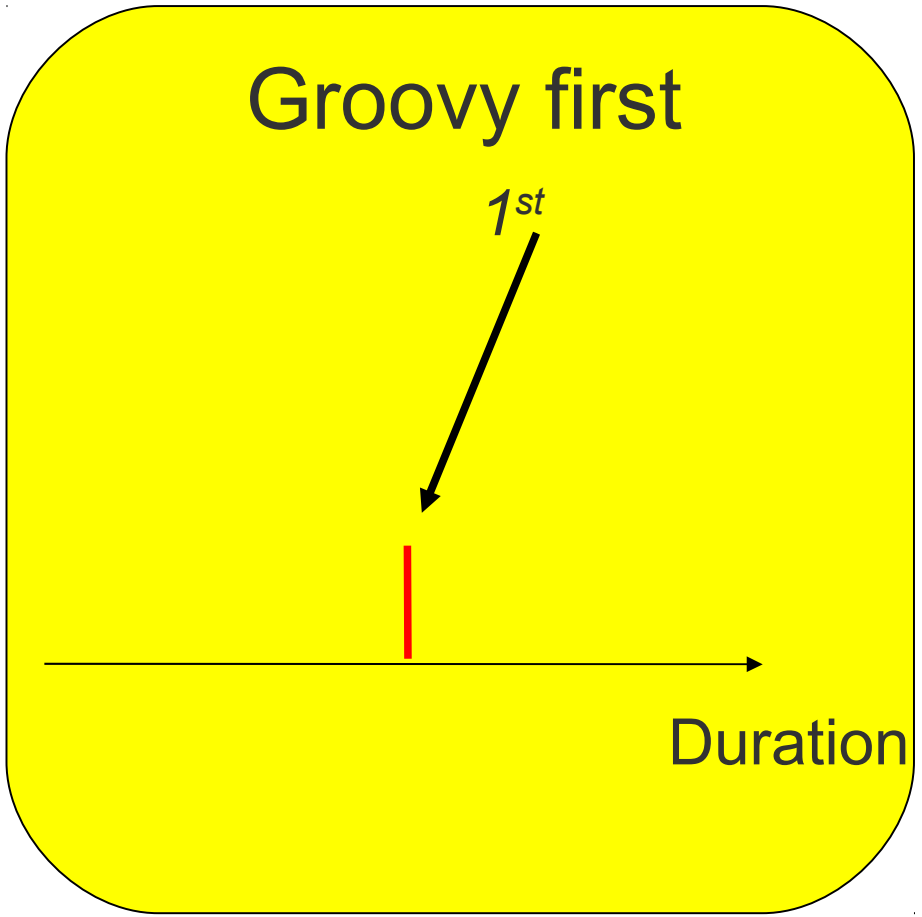
- **reduce deviation, and/or**
- **increase effect size**

- **Possible ways**
  - **Adaptation of experimental design**
    (e.g. within-subject design) => **Reduction of deviation**
  - **Adaptation of tasks**
    (no development „from scratch“) => **Incease effect size**

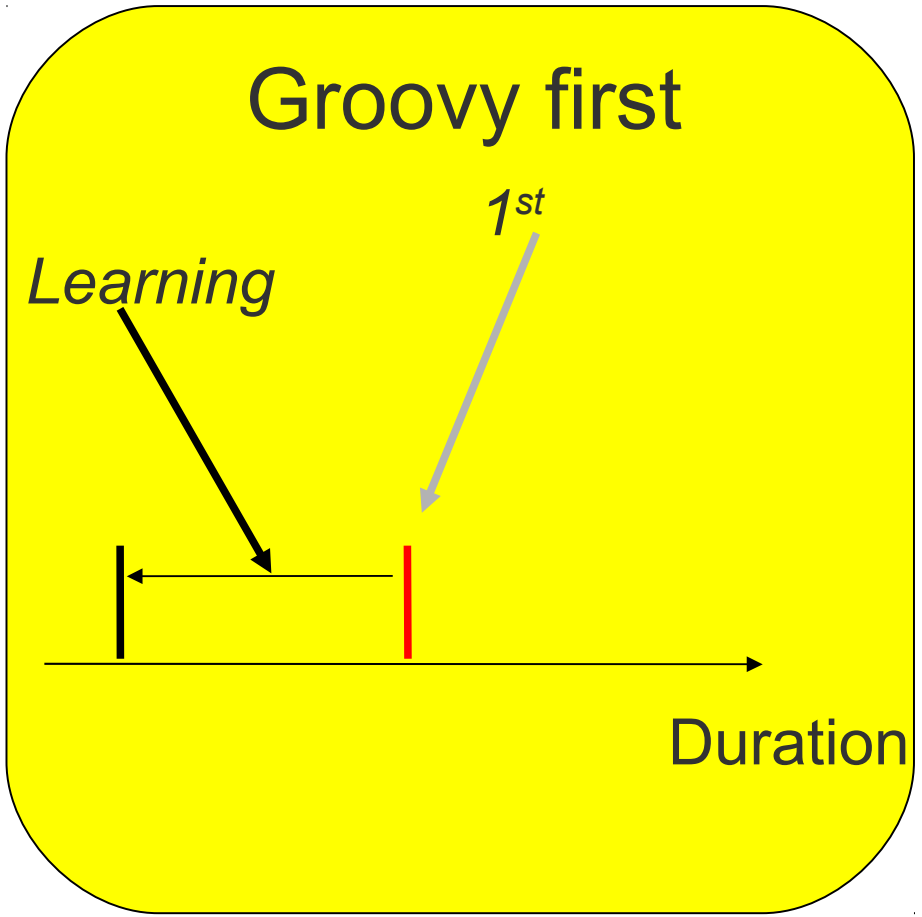
Within-Subject Design: Example

- Question: Do type Casts Matter? **[Stuchlik, Hanenberg DLS 2011]**
  - 21 subjects (~ 5 h/subject)
  - Programming Languages: Groovy & Java
  - 5 simple programming tasks
  - Measurement: time until completion
  - Hypothesis: **devTime(Groovy) < devTime(Java)**
  - Within-subject design
    - Low number of subjects
    - High variance between subjects

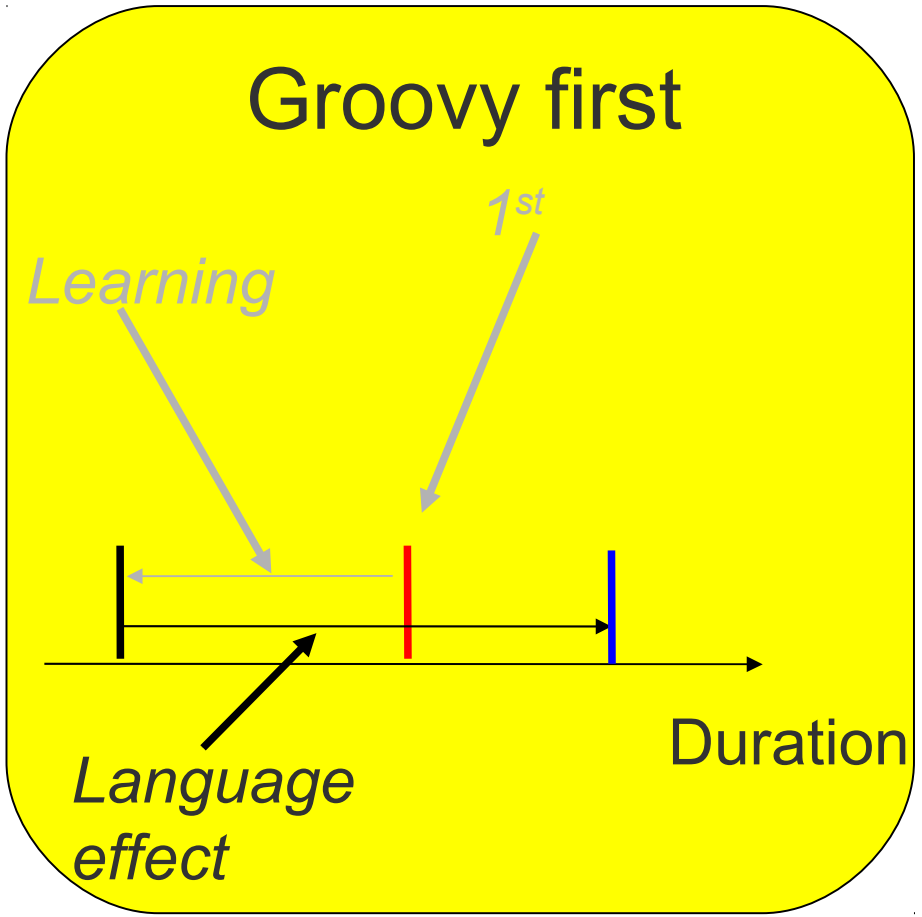
Within-subject measurement



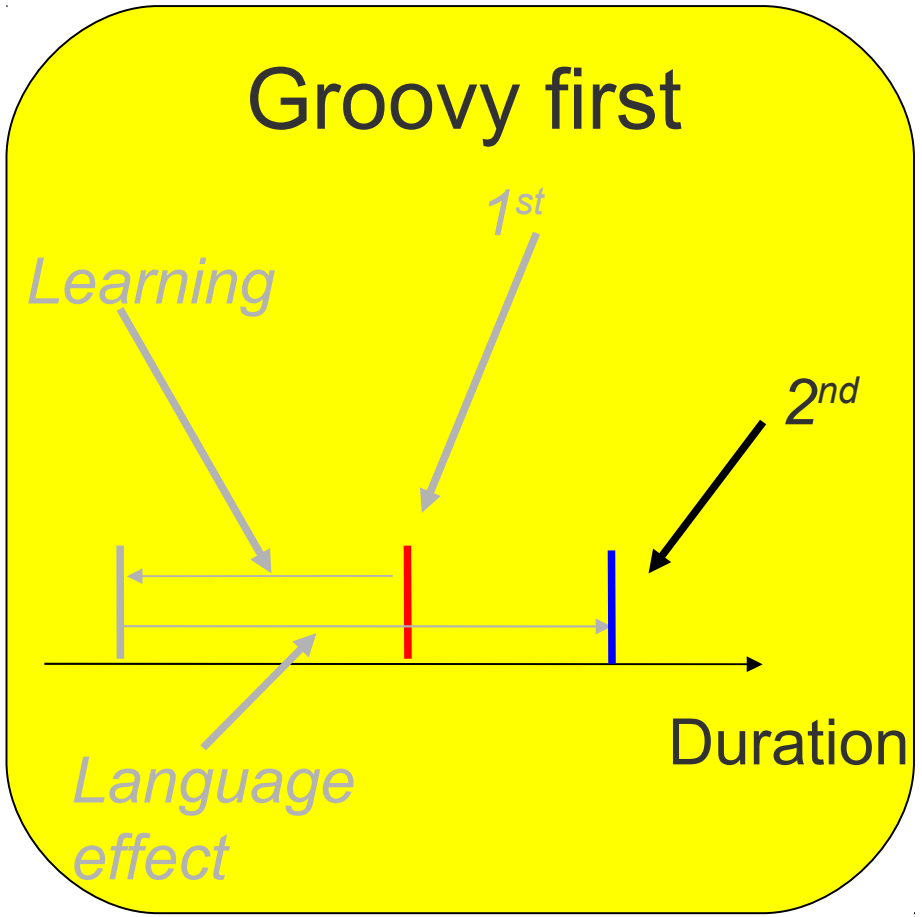
Within-subject measurement



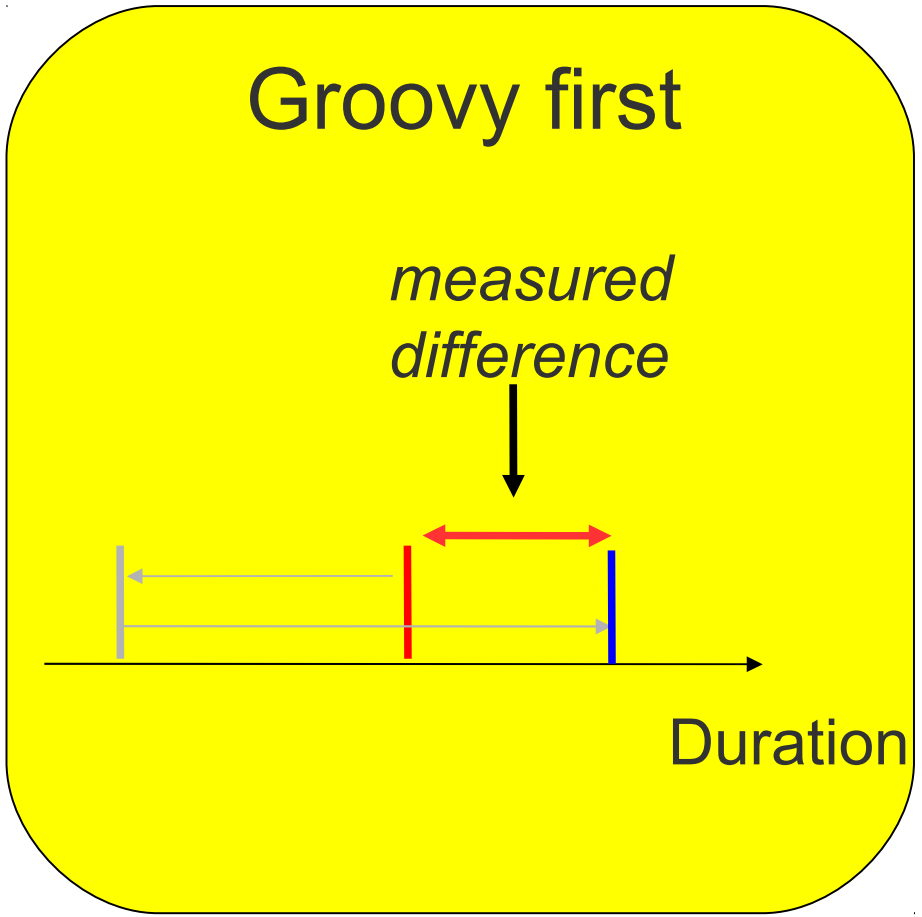
Within-subject measurement



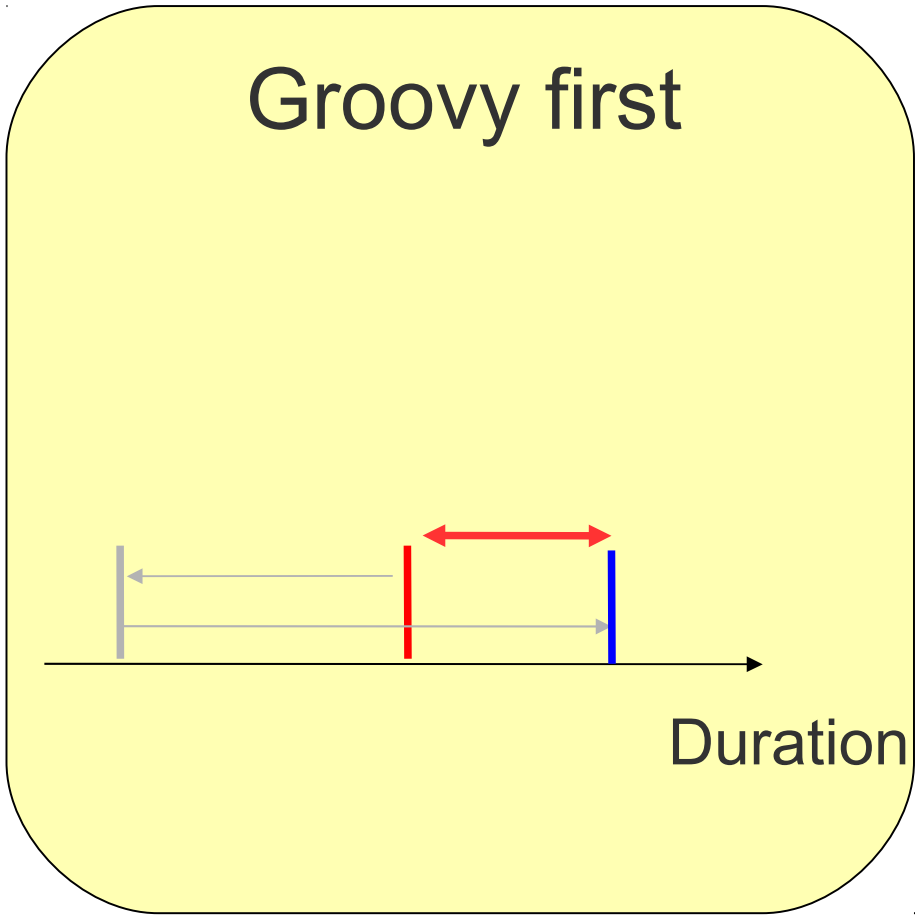
Within-subject measurement

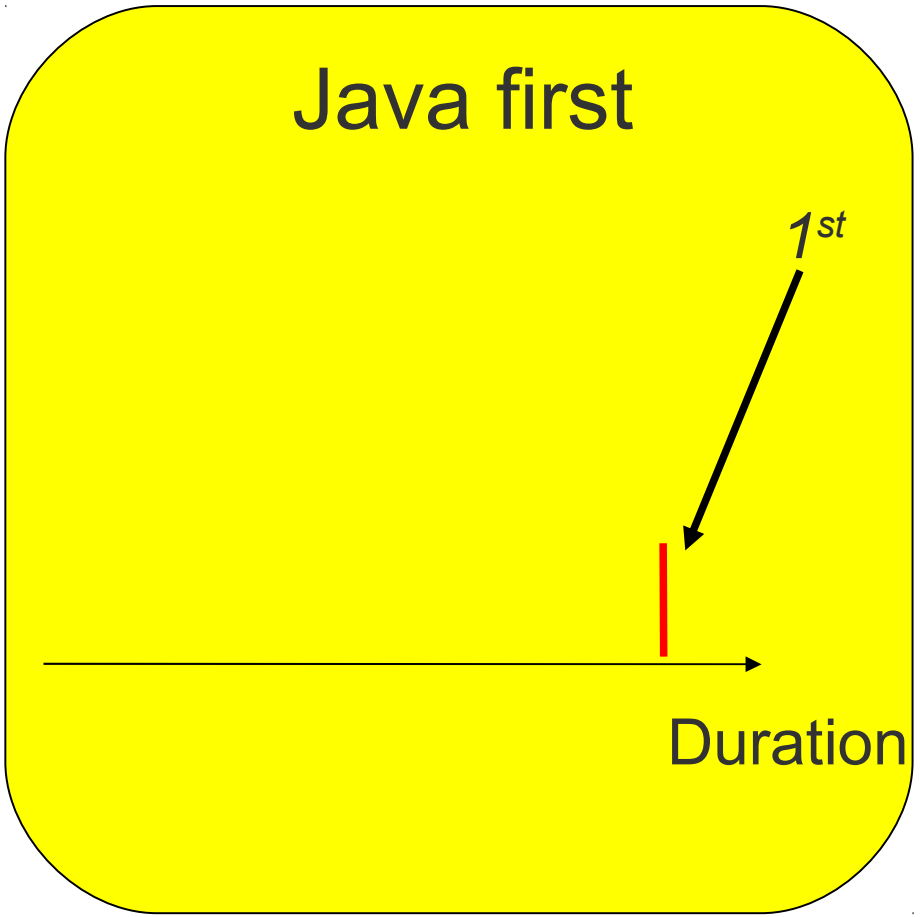


Within-subject measurement

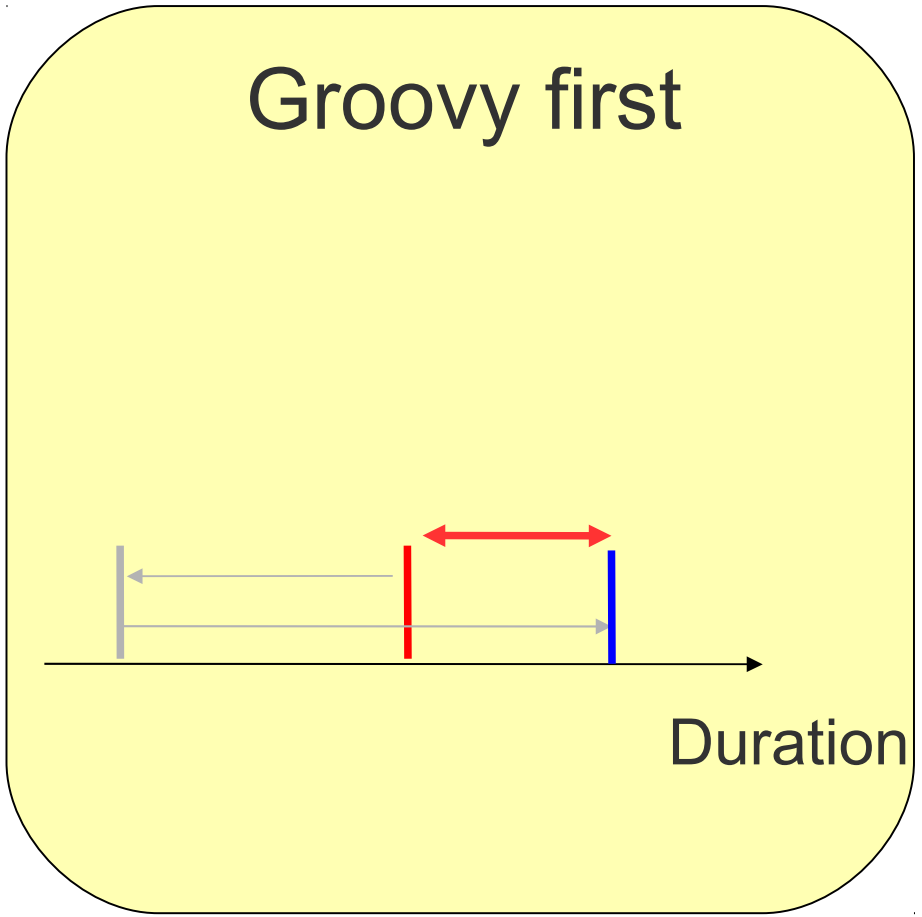


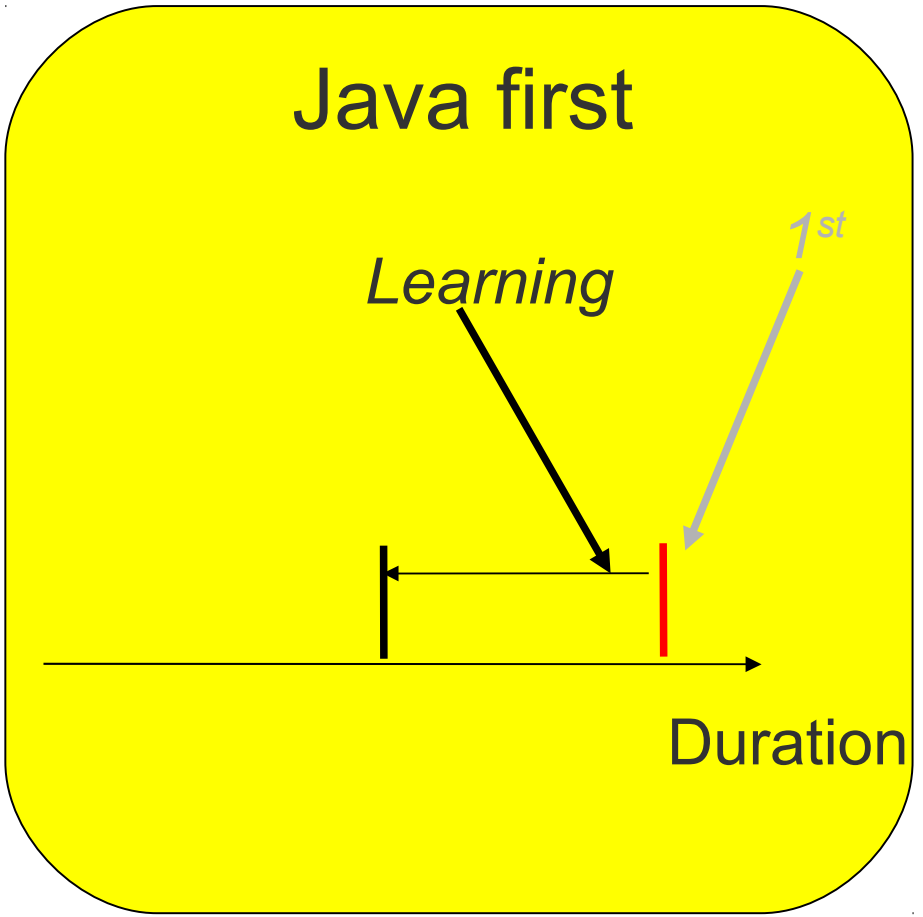
Within-subject measurement



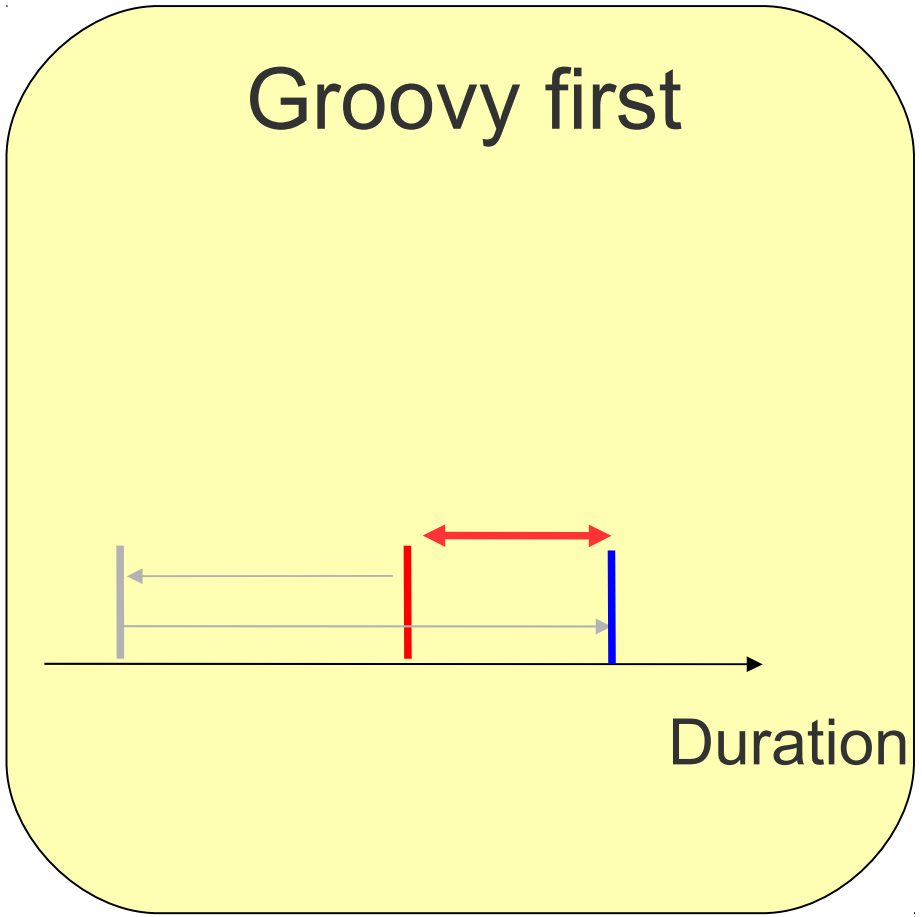


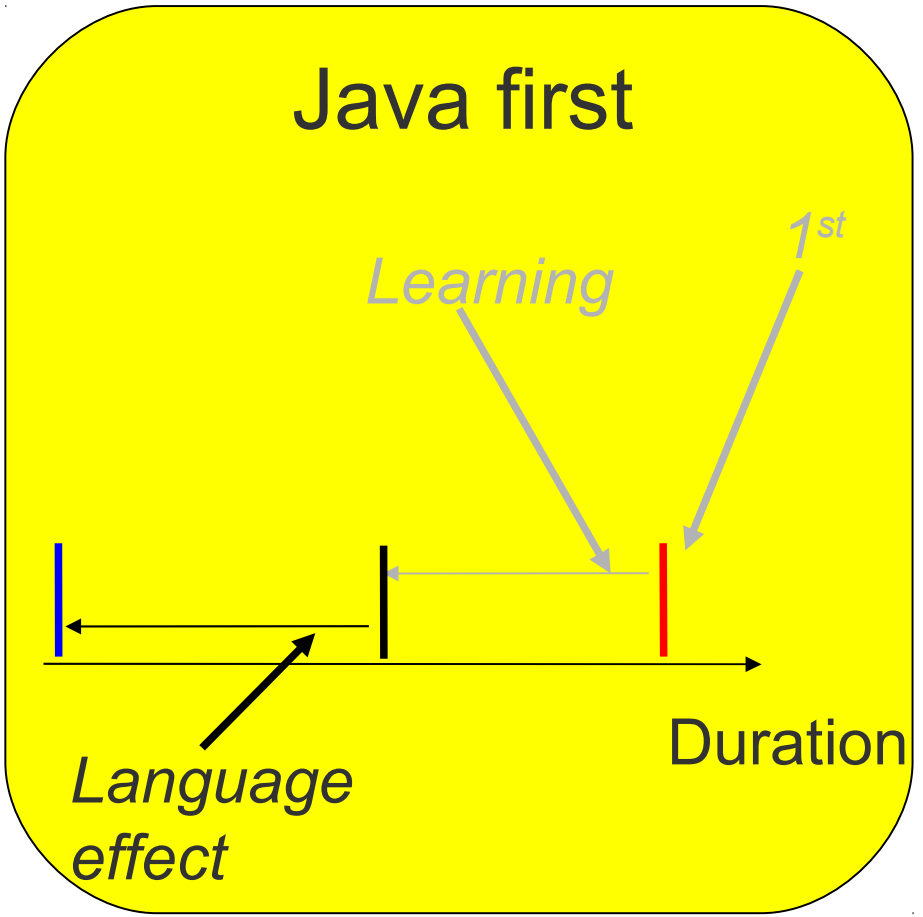
Within-subject measurement



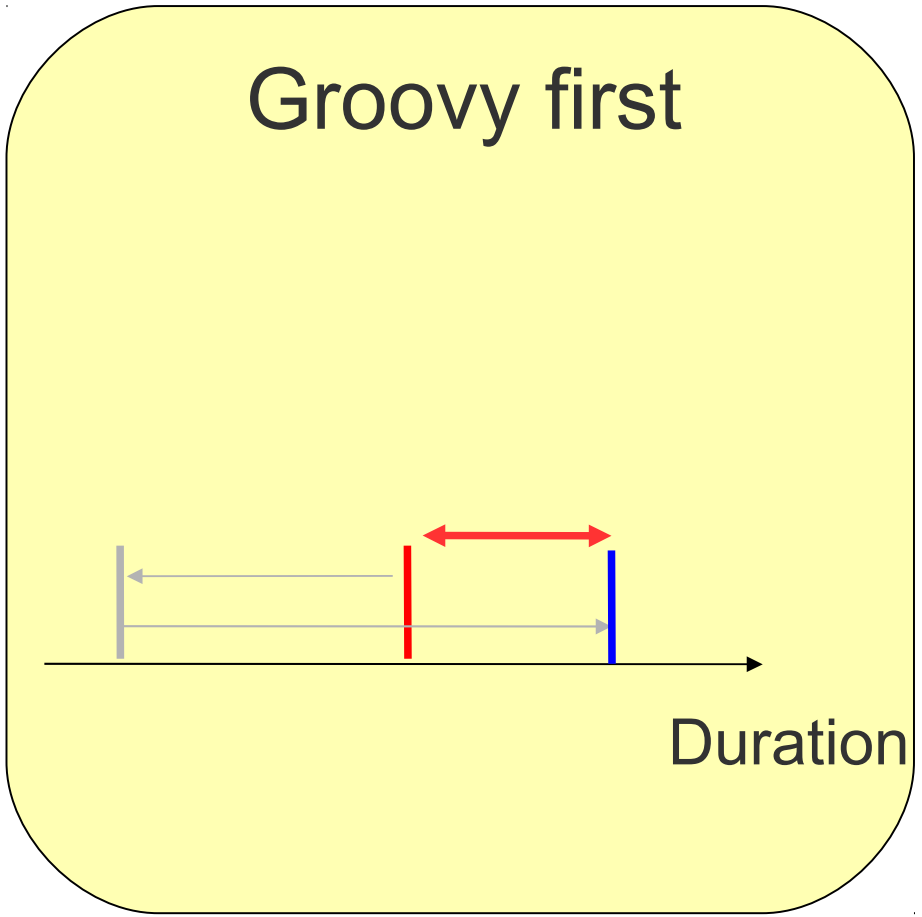


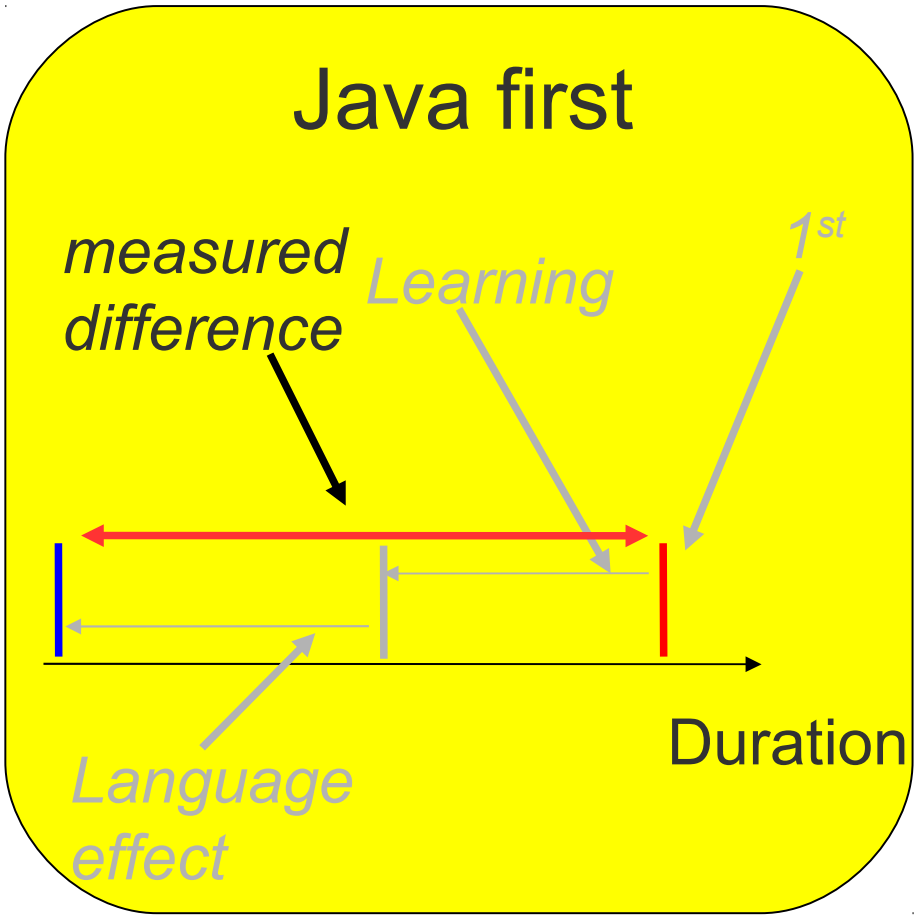
Within-subject measurement



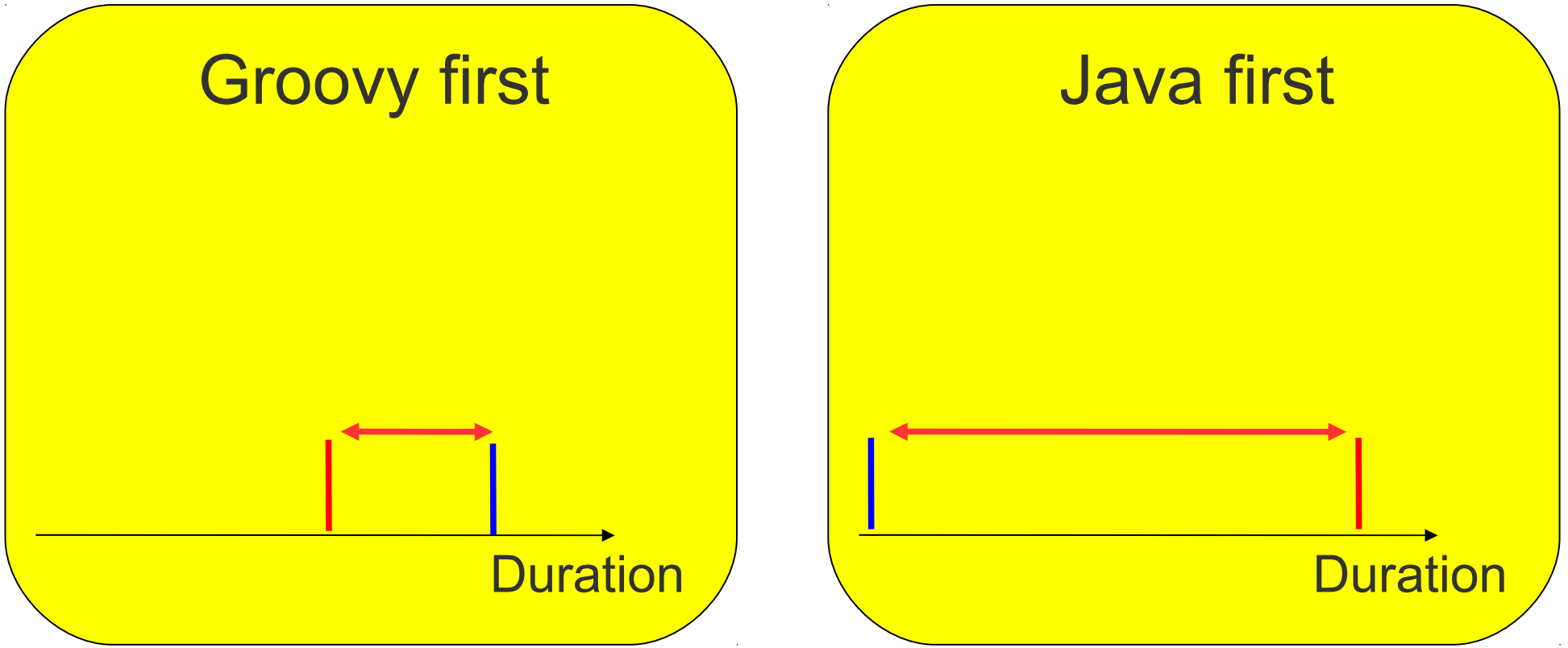


Within-subject measurement





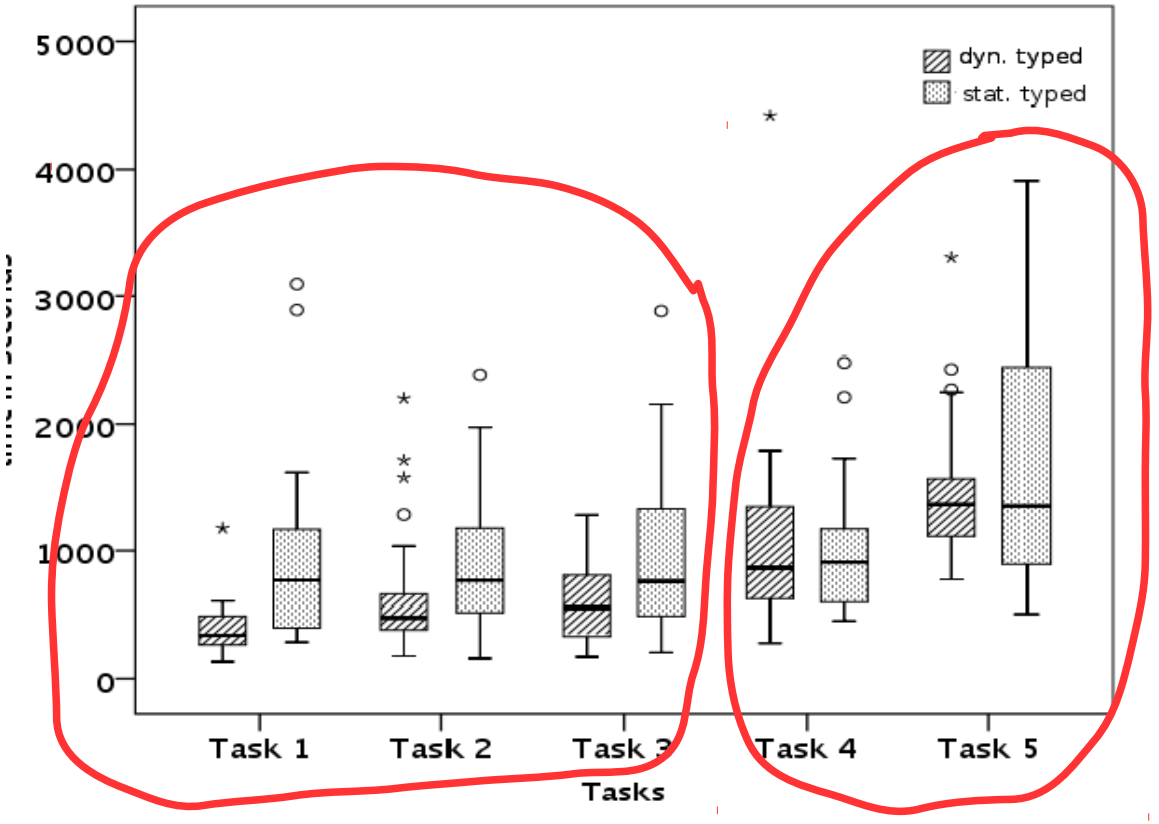
Within-subject measurement



- Small effect in Group „Groovy First“
- Large effect in Group „Java first“

Experiment Results

[Stuchlick, Hanenberg@DLS'11]



- Results/Interpretation
  - Type casts are not that important

Within-subject measurement

- Problem
  - If learning effect larger than language effect
    => no measured difference

- But...
  - Large effort put into task definition and pilot-tests
  - Learning effects rather minor problems

Task Definition

- What is the hypothesis?
  - Large number of techniques do not already provide one
- Motivation
  - „Find a programming task, where static type system (likely) have an effect“
  - Reduce confounding factors as much as possible
    - No IDE (!), Tasks quite small
  - Variability among subjects should be as less as possible

- Our „process“
  - Discussion, discussion, discussion, pure speculations
  - Very small pilot studies

Task Definition - Example

- Task
  - „Create for the dungeon game a new field, which contains a trap and put a new hero on it“

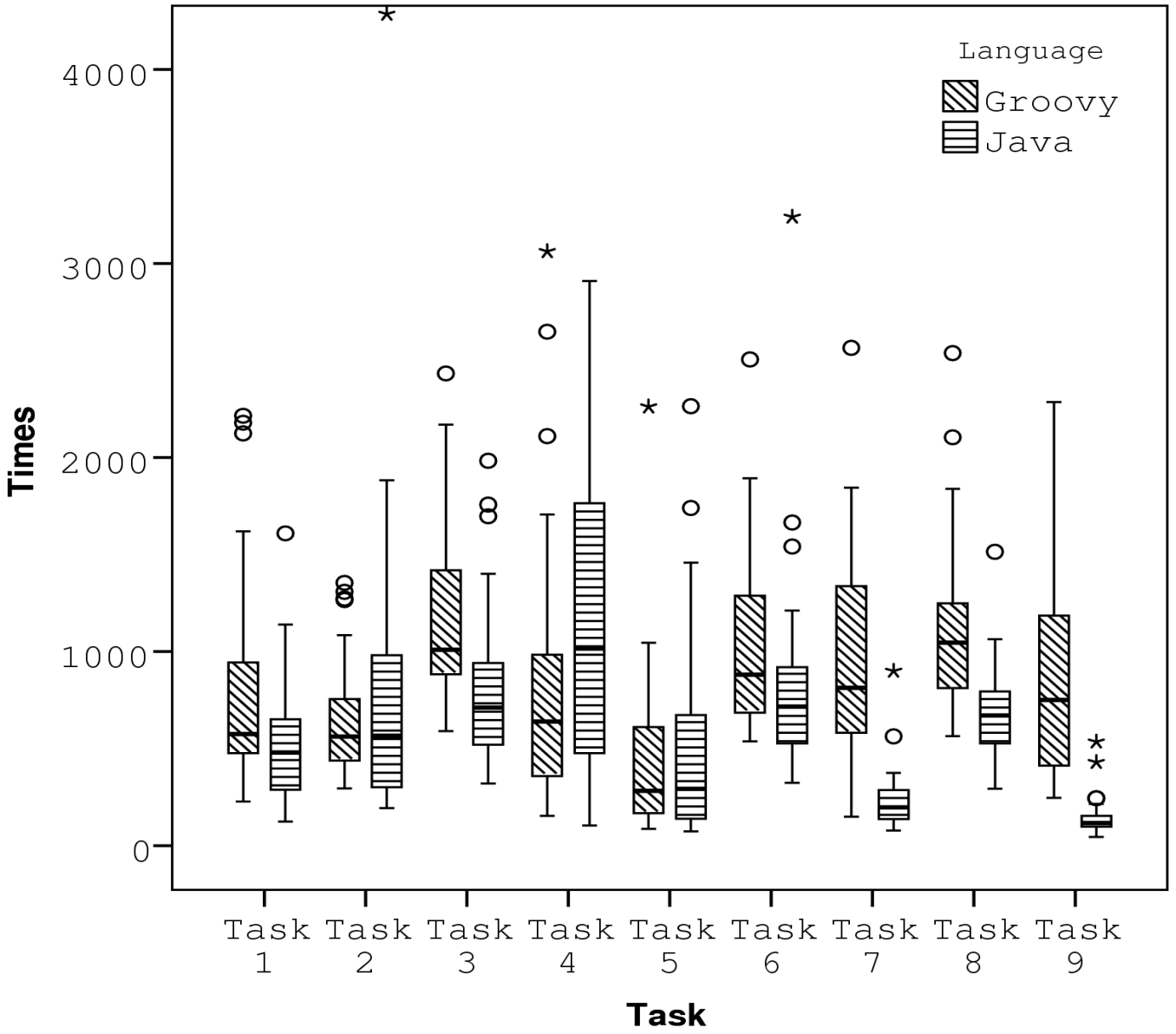
```
// Groovy solution
public def setUpLevelField(def x_position,  def y_position, def trapKind){
  def trap = new Trap(trapType);
  def trapField = new TrappedLevelField(x_position, y_position, trap);
  trapField.setItems(new GameList());
  trapField.setSubject(new Player(new Inventory(), new Body()));
  return trapField;
}
```

Example: Static Type System

- Background: 4 experiments, „mixed results“
- Idea: Static type systems help when using an undocumented API
- Experiment
  - Java / Groovy as PLs
  - 9 programming tasks (designing tasks took about 2 month)
    - 2 tasks: fix semantic error / 2 tasks: fix type error / 5 tasks: use API classes
  - 33 subjects (mainly students)
  - Within-subject design (2 groups)
- Result
  - Positive effect for 6/9 tasks
    - No effect on fixing semantic error
    - Positive effect on fixing type error
    - Mostly (4/5) positive effect on using API classes

Example: Static Type System

- Task 4,5: Semantic errors
- 1,2,3,6,8: New class usage
- 7, 10: Type errors



Example: Static Type System

- Potential problems
  - Artificially constructed API
    - parameter names do not reflect on type names (but on names chosen from the domain)
    - Is it repesentative?
  - Artificially constructed environment
  - Artificial programming tasks
  - Java type system
- Maybe we measured something else
  - „*Existence of type annotations in the code help....no matter whether they are statically type checked or not*“
- Maybe „in the wild“ positive effect of static type system „vanishs“
  - There is no generalizability

Discussion & Conclusion

- Controlled experiments as a research method
- Many, many problems
  - Missing experimentation in the past
  - Basics
  - Organizational issues
  - ...
- It is still worth to do experiments
  - Programming languages are (mostly) for humans

Problem(s) in Experimentation

- Between-Subject Design: Each subject measured once
  - Problem
    - Deviation among subjects potentially hides effect
    - Requires balancing between groups (for small groups)
  - Benefit
    - No learning effect , Lower costs than within-subject-design
- Within-Subject Design: Each subject measured twice
  - Problem
    - Obvious learning effects
  - Benefit
    - Indivivual deviation not that important

Controlled Experiments for the empirical evaluation of programming language constructs: type systems as an example

Stefan Hanenberg
University of Duisburg-Essen, Germany

London, UK, 27.02.2012