Experiments on Object-Oriented Analysis and Design

Erik Arisholm

Series of experiments forming one quasi-experiment

• A series of five controlled experiments (can be considered as one quasi-experiment) where the subjects consisting of
  • 295 junior/intermediate/senior Java consultants from Norway, Sweden and the UK, and
  • 273 undergraduate/graduate students from Norway and Canada
• performed maintenance tasks on two alternative designs of the same Java system
• to assess the effects of (combinations of)
  • control style (centralized vs delegated),
  • maintenance task order (easy vs difficult task first),
  • documentation (UML versus no UML), and
  • development process (pair programming vs individual)
• on software maintainability (change effort and correctness)
[simula research laboratory]

The individual experiments

Exp 1: The Original Experiment with students and pen-and-paper tasks (Fall 1999)
- Effect of Centralized (bad) vs Delegated (good) Control Style
- 36 undergraduate students and 12 graduate students

Exp 2: The Control-style experiment with professional Java developers and Java tools (Fall 2001-Spring 2002)
- Effect of Centralized vs Delegated Control Style for Categories of Developer
- 99 professionals, 59 students

Exp 3: UML experiments (Spring 2003 - Fall 2004)
- Effect of UML (vs No UML) for the Delegated Control Style
- 20 students from UiO (Spring 2003) + 78 students from Carleton Univ., Canada (Fall 2004)

Exp 4: Task Order Experiment (Fall 2001-Spring 2005)
- Effect of Task Order and Centralized vs Delegated Control Style
- Difficult task first: 66 students from NTNU (Spring 2005)

Exp 5: Pair programming experiment (Fall 2003-Spring 2005):
- Effect of Pair Programming (vs individual programming) and Delegated vs Centralized Control Style
- 196 professional programmers in Norway, Sweden, and UK (98 pairs)
- 99 individuals (from Exp 2)

A quasi-experiment of increasing scope

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Control Style</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>UML</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Task Order</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Pair Programming</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Coffee-machine</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>Duration (minutes)</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>Correctness (%)</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>Centralized (CC)</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>Delegated (DC)</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>No UML</td>
<td></td>
<td>x</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Some UML</td>
<td></td>
<td>x</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Complete UML</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Easy First</td>
<td></td>
<td>x</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Difficult first</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Java Pen &amp; Paper</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Java IDE</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>UML Tool</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Individual</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Pair Programming</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BSc-students</td>
<td>36</td>
<td>27</td>
<td>98</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MSc-students</td>
<td>12</td>
<td>32</td>
<td>66</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Juniors</td>
<td>31</td>
<td>50</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Intermediates</td>
<td>32</td>
<td>70</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Seniors</td>
<td>36</td>
<td>76</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Centralized vs Delegated Control Style

- **The Delegated Control Style:**
  - Rebecca Wirfs-Brock: A delegated control style ideally has clusters of well defined responsibilities distributed among a number of objects. To me, a delegated control architecture feels like object design at its best...
  - Alistair Cockburn: [The delegated coffee-machine design] is, I am happy to see, robust with respect to change, and it is a much more reasonable "model of the world."

- **The Centralized Control Style:**
  - Rebecca Wirfs-Brock: A centralized control style is characterized by single points of control interacting with many simple objects. To me, centralized control feels like a "procedural solution" cloaked in objects...
  - Alistair Cockburn: Any oversight in the "mainframe" object (even a typo!) [in the centralized coffee-machine design] means potential damage to many modules, with endless testing and unpredictable bugs.

### The treatments

<table>
<thead>
<tr>
<th></th>
<th>CC</th>
<th>DC</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>CoffeeMachine</strong></td>
<td>Initiates the machine; knows how the machine is put together; handles input.</td>
<td>Initiates the machine; knows how the machine is put together; handles input.</td>
</tr>
<tr>
<td><strong>CashBox</strong></td>
<td>Knows amount of money put in; gives change; answers whether a given amount of credit is available.</td>
<td>Knows amount of money put in; gives change; answers whether a given amount of credit is available.</td>
</tr>
<tr>
<td><strong>FrontPanel</strong></td>
<td>Knows selection; knows price of selections, and materials needed for each; coordinates payment; knows what products are available; knows how each product is made; knows how to talk to the dispensers.</td>
<td>Knows selection; coordinates payment; delegates drink making to the Product.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>Product</strong></th>
<th>Knows its recipe and price.</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>ProductRegister</strong></td>
<td>Knows what products are available.</td>
</tr>
<tr>
<td><strong>Recipe</strong></td>
<td>Knows the ingredients of a given product; tells dispensers to dispense ingredients in sequence.</td>
</tr>
<tr>
<td><strong>Dispensers</strong></td>
<td>Controls dispensing; tracks amount it has left.</td>
</tr>
<tr>
<td><strong>DispenserRegister</strong></td>
<td>Knows what dispensers are available.</td>
</tr>
<tr>
<td><strong>Ingredient</strong></td>
<td>Knows its name only.</td>
</tr>
<tr>
<td><strong>Output</strong></td>
<td>Knows how to display text to the user.</td>
</tr>
<tr>
<td><strong>Input</strong></td>
<td>Knows how to receive command-line input from the user.</td>
</tr>
<tr>
<td><strong>Main</strong></td>
<td>Initializes the program</td>
</tr>
</tbody>
</table>
Assignment of subjects (Exp 2):
Randomized Block Design

<table>
<thead>
<tr>
<th></th>
<th>CC</th>
<th>DC</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>Undergraduate</td>
<td>13</td>
<td>14</td>
<td>27</td>
</tr>
<tr>
<td>Graduate</td>
<td>15</td>
<td>17</td>
<td>32</td>
</tr>
<tr>
<td>Junior</td>
<td>16</td>
<td>15</td>
<td>31</td>
</tr>
<tr>
<td>Intermediate</td>
<td>17</td>
<td>15</td>
<td>32</td>
</tr>
<tr>
<td>Senior</td>
<td>17</td>
<td>19</td>
<td>36</td>
</tr>
<tr>
<td>Total</td>
<td>78</td>
<td>80</td>
<td>158</td>
</tr>
</tbody>
</table>

Experiment design (Exp 2)

- Experience Questionnaire: Resolves most technical problems. Subjects get familiar with the experiment process.
- Training Task: Common task to compare the programming skills of the subjects.
- Task 1: Pretest Task
  - Randomized block assignment: Students: "undergraduate", "graduate".
  - Professionals: "junior*company", "intermediate*company", "senior*company".
- Subject solves 4 tasks on either the delegated or the centralized design. Each task builds on the solution given to the previous task.
Results

The effect of control style depends mainly on the experience of the developers!

Discussion point (Exp 2)

- What are the main threats to validity of this study?
- Is it possible to extend the design to address some of the threats?
Experiment 4: The Effect of Task Order on the Maintainability of Object-Oriented Software (*)

- Research questions
  - RQ1: Does the order in which you perform maintenance tasks affect maintainability?
  - RQ2: Does the effect of task order depend on how the system is structured (control style)?


Another way to look at RQ1 and RQ2
Hypotheses

- **H0₁ – The Effect of Task Order on Duration.** The time taken to perform change tasks is equal for the easy-first and hard-first task order.

- **H0₂ – The Moderating Effect of Design on Duration.** The difference in the time taken to perform change tasks for easy-first and hard-first task order does not depend on design.

- **H0₃ – The Effect of Task Order on Correctness.** The correctness of the maintained programs is equal for easy-first and hard-first task order.

- **H0₄ – The Moderating Effect of Design on Correctness.** The difference in the correctness of the maintained programs for easy-first and hard-first task order does not depend on design.

Experiment design
Dealing with non-equivalent groups (to compare results across experiments)
Necessary to have a common pre-test to adjust for skill differences between groups (using ANCOVA)*

Post-test (Y) = β₀ + β₁*x + β₂*z

Z = treatment z=1
O = treatment z=0


<table>
<thead>
<tr>
<th>Model</th>
<th>Response</th>
<th>Distrib.</th>
<th>Link</th>
<th>Model Term</th>
<th>Primary use of model term</th>
</tr>
</thead>
<tbody>
<tr>
<td>(1)</td>
<td>Duration</td>
<td>Gamma</td>
<td>Log</td>
<td>Log(pre_dur)</td>
<td>Covariate to adjust for individual skill differences</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>TaskOrder</td>
<td>Test H₀₁ (Duration Main Effect)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Design</td>
<td>Models the effect of design on duration</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>TaskOrder x Design</td>
<td>Test H₀₂</td>
</tr>
<tr>
<td>(2)</td>
<td>Correctness</td>
<td>Binomial</td>
<td>Logit</td>
<td>Log(pre_dur)</td>
<td>Covariate to adjust for individual skill differences</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>TaskOrder</td>
<td>Test H₀₃ (Correctness Main Effect)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Design</td>
<td>Models the effect of design on correctness</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>TaskOrder x Design</td>
<td>Test H₀₄</td>
</tr>
</tbody>
</table>
Effects of Task Order and Control Style

Discussion points (Exp 4)

• What are the main threats to validity of this experiment?

• What are the practical implications of the results?
Exp. 3: Does complete UML documentation with use-case, sequence and class-diagrams help novice developers (BSc-students) to understand the delegated control style?

See also:

---

Task correctness with or without UML

---

Effort spent on coding and testing

Effort to understand, code and test tasks 2+3+4

Effort spent on coding and testing and updating the diagrams

Effort (incl. updating UML doc) for tasks 2+3+4
Discussion points (Exp 3)

• What are the main threats to validity of this experiment?

• How would you extend the experiment to address some of these threats?

The Pair Programming Experiment (Exp 5)

• Independent variable:
  – Pair programming vs. individual programming

• Moderator variables:
  – Developer category (junior, intermediate, senior)
  – Control-style (delegated and centralized)

• Dependent variables:
  – duration, effort and correctness

• Treatments:
  – Junior, intermediate and senior pairs and individuals were randomly assigned to one of two alternative (delegated or centralized) designs of a coffee-machine, and performed four (incremental) maintenance tasks on the system

• Subjects:
  – 295 junior, intermediate and senior professional Java consultants:
    • 99 individuals (conducted in 2001/2002)
    • 98 pairs (conducted in 2004/2005)
Research Question

What is the effect (regarding duration, effort and correctness) of pair programming (vs. individual programming) for various levels of programmer expertise and task complexity when performing change tasks?

Quasi-Experimental Design
(combines subjects from Exp 2 and Exp 5)

Presentation to Participants

Pre-Questionnaire
e.g., education, experience

Training Task and Task 1

Tasks 2-5 (Centralized Control Style)
- Individual
  - Junior ind (16)
  - Intermed. ind (17)
  - Senior ind (17)
- Pair
  - Junior pair (12)
  - Intermed. pair (25)
  - Senior pair (19)

Tasks 2-5 (Delegated Control Style)
- Individual
  - Junior ind (15)
  - Intermed. ind (15)
  - Senior ind (19)
- Pair
  - Junior pair (13)
  - Intermed. pair (15)
  - Senior pair (19)
Some of the preparations...

- Initial pilot experiment (presented at ISERN'02 in Japan)
- Design of the main experiment (presented at ISERN'03 in Italy)
- Quality assurance of experimental materials (three pilot experiments)
- Defining and evaluating questionnaires (pre-, post-, and personal characteristics)
  - More than 200 questions
  - Tested the questionnaires on 28 professional developers
- Translating all experimental materials to Swedish and English (from Norwegian)
- Development of automatic test-script and web-based review system
  - Two independent reviewers analyzed all task solutions manually (more than 1000 tasks)
- Collaboration with a psychologist, who was responsible for the administration and analysis of a "big five" personality test
- Recruiting 295 Java consultants from 27 consultancy companies in Norway, Sweden and UK!

Formal Hypotheses

- **H0<sub>1</sub>** Effect of Pair Programming on Duration: The duration to perform change tasks is equal for individuals and pairs.
- **H0<sub>2</sub>** Effect of Pair Programming on Change Effort: The effort spent to perform change tasks is equal for individuals and pairs.
- **H0<sub>3</sub>** Effect of Pair Programming on Quality: The quality of the maintained programs is equal for individuals and pairs.
- **H0<sub>4</sub>** Moderating Effect of Programmer Expertise on Duration: The difference in the duration to perform change tasks for pairs versus individuals does not depend on programmer expertise.
- **H0<sub>5</sub>** Moderating Effect of Programmer Expertise on Change Effort: The difference in the effort spent to perform change tasks for pairs versus individuals does not depend on programmer expertise.
- **H0<sub>6</sub>** Moderating Effect of Programmer Expertise on Quality: The difference in the quality of the maintained programs for pairs versus individuals does not depend on programmer expertise.
- **H0<sub>7</sub>** Moderating Effect of Task Complexity on Duration: The difference in the duration to perform change tasks for pairs versus individuals does not depend on task complexity.
- **H0<sub>8</sub>** Moderating Effect of Task Complexity on Change Effort: The difference in the effort spent to perform change tasks for individuals and pairs does not depend on task complexity.
- **H0<sub>9</sub>** Moderating Effect of Task Complexity on Quality: The difference in the quality of the maintained programs for pairs versus individuals does not depend on task complexity.
**ANCOVA model specifications**

<table>
<thead>
<tr>
<th>Model Response</th>
<th>Model Term</th>
<th>Primary use of model term</th>
</tr>
</thead>
<tbody>
<tr>
<td>Log(Duration)</td>
<td>Log(pre_dur)</td>
<td>Covariate to adjust for individual differences in programming skill</td>
</tr>
<tr>
<td></td>
<td>PP</td>
<td>Test H01 (Duration Main Effect)</td>
</tr>
<tr>
<td></td>
<td>CS</td>
<td>Assess the effect of control style on duration</td>
</tr>
<tr>
<td></td>
<td>Log(pre_dur)*PP</td>
<td>Test H04</td>
</tr>
<tr>
<td></td>
<td>Log(pre_dur)*CS</td>
<td>Test for ANCOVA assumption of homogeneity of slopes</td>
</tr>
<tr>
<td></td>
<td>PP*CS</td>
<td>Test H07</td>
</tr>
<tr>
<td></td>
<td>Log(pre_dur)<em>PP</em>CS</td>
<td>Test for ANCOVA assumption of homogeneity of slopes</td>
</tr>
<tr>
<td>Log(Effort)</td>
<td>Log(pre_dur)</td>
<td>Covariate to adjust for individual differences in programming skill</td>
</tr>
<tr>
<td></td>
<td>PP</td>
<td>Test H02 (Effort Main Effect)</td>
</tr>
<tr>
<td></td>
<td>CS</td>
<td>Assess the effect of control style on duration</td>
</tr>
<tr>
<td></td>
<td>Log(pre_dur)*PP</td>
<td>Test H05</td>
</tr>
<tr>
<td></td>
<td>Log(pre_dur)*CS</td>
<td>Test for ANCOVA assumption of homogeneity of slopes</td>
</tr>
<tr>
<td></td>
<td>PP*CS</td>
<td>Test H08</td>
</tr>
<tr>
<td></td>
<td>Log(pre_dur)<em>PP</em>CS</td>
<td>Test for ANCOVA assumption of homogeneity of slopes</td>
</tr>
<tr>
<td>Log(Correctness)</td>
<td>Log(pre_dur)</td>
<td>Covariate to adjust for individual differences in programming skill</td>
</tr>
<tr>
<td></td>
<td>PP</td>
<td>Test H03 (Correctness Main Effect)</td>
</tr>
<tr>
<td></td>
<td>CS</td>
<td>Assess the effect of control style on duration</td>
</tr>
<tr>
<td></td>
<td>Log(pre_dur)*PP</td>
<td>Test H06</td>
</tr>
<tr>
<td></td>
<td>Log(pre_dur)*CS</td>
<td>Test for ANCOVA assumption of homogeneity of slopes</td>
</tr>
<tr>
<td></td>
<td>PP*CS</td>
<td>Test H09</td>
</tr>
<tr>
<td></td>
<td>Log(pre_dur)<em>PP</em>CS</td>
<td>Test for ANCOVA assumption of homogeneity of slopes</td>
</tr>
</tbody>
</table>

**Results**

- **Total Effect of PP**
  - 84%
  - 7%
  - -8%
  - -40%
  - -20%
  - 0%
  - 20%
  - 40%
  - 60%
  - 80%
  - 100%
  - 120%
  - 140%
  - 160%

- **Moderating Effect of System Complexity on PP**
  - CC (easy)
  - DC (complex)

- **Effect of PP for Juniors**
  - 5%
  - 111%
  - 73%
  - -40%
  - -20%
  - 0%
  - 20%
  - 40%
  - 60%
  - 80%
  - 100%
  - 120%
  - 140%
  - 160%

- **Effect of PP for Intermediates**
  - 43%
  - 4%
  - -28%
  - -40%
  - -20%
  - 0%
  - 20%
  - 40%
  - 60%
  - 80%
  - 100%
  - 120%
  - 140%
  - 160%

- **Effect of PP for Seniors**
  - 55%
  - 5%
  - 111%
  - 73%
  - -40%
  - -20%
  - 0%
  - 20%
  - 40%
  - 60%
  - 80%
  - 100%
  - 120%
  - 140%
  - 160%
Summary of results

- Performing change tasks on a delegated control style requires, on average, more time and results in more defects than on a centralized control style, in particular for novices (undergraduate students and junior consultants)
  - Only seniors seem to have the necessary skills to benefit from the more "elegant" delegated control style.
  - Explanation: Unlike experts, novices perform a mental trace the code in order to understand it. This tracing effort is more difficult in a delegated control style.
  - Results are consistent when performing the most difficult task first (though the disadvantage of a delegated control style is smaller than for easy first).

- Two ways to decrease the cognitive complexity of the delegated control style (for novices in particular)
  - Extensive UML documentation
  - Pair programming