REVISED EXPERIMENT IN EVOLUTION COMPLEXITY:

INSTRUCTIONS TO SUBJECTS

Technical report CSM-439, ISSN 1744-8050 (November 2005)
Department of Computer Science, University of Essex
Institut d'Informatique, Université de Mons-Hainaut

Tom Mens ([1]) and Amnon H. Eden ([2])

Abstract. We describe an experiment whose purpose is to establish our hypothesis regarding the relative "software flexibility" of particular design policies (in this particular case, programming styles) towards particular changes. We present a program with two implementation variants, one using a procedural programming style and the other one using an object-oriented programming style. We instruct the subjects to carry out a predefined set of changes in each implementation, and to test the changes they made. We measure the time each change required, in order to be able to compare the effect of the programming style used.

([1]) Service de Génie Logiciel, Institut d'Informatique, Université de Mons-Hainaut
Email: tom.mens@umh.ac.be
Postal address: Avenue du champ de Mars 6, 7000 Mons, Belgium
Phone: +32 65 37 3453, Fax: +32 65 37 3459

([2]) Department of Computer Science, University of Essex *and* Center For Inquiry
Email: eden@essex.ac.uk
Postal address: Colchester, Essex CO4 3SQ, United Kingdom
Phone: +44 (1206) 872677, Fax: +44 (1206) 872788

INSTRUCTIONS TO SUBJECTS – GROUP 1

Important !

Please read carefully this *Instructions sheet* from beginning to end before anything else.

The purpose of this experiment is to test the hypothesis put forth under the title "evolution complexity". There is no need for you to understand the theory; in fact, the integrity of the experiment is insured by ignoring the predictions we make.

Generally speaking, our hypothesis regards the time that certain evolution tasks will take to complete. To test this hypothesis, we ask you to carry out the tasks described in the Tasks tables (Table 1 and Table 4), to measure the time it took you to complete each using the Diary (Table 5), and to summarize the results in the Summary sheet (Table 6).

Before conducting this experiment, make sure that, along with this document, you are sitting next to a computer running a clear display of the digital clock, and that same computer has a working Java environment including a complete installation of Java, including the Java compiler and run-time environment.

Before you begin working on a particular task, please start a new entry in your Diary: write the task number under TASK and the current time under START TIME.

After completing a session, whether you decide to take a break or have finished a task, enter the current time in your Diary under END TIME. When you resume working, start a new Diary entry.

After completing all tasks, summarize the time it took you to carry out each task and write the total in the Summary sheet (Table 6).

Sample Diary

The following three entries in a Diary tell us you spent two 40-minute sessions on task no. 1 and one hour session on task no. 2:

TASK	START TIME	END TIME	LENGTH
1	1-Jan-05, 9:12	9:52	0:40
1	1-Jan-05, 12:20	13:00	0:40
2	2-Jan-05, 23:11	3-Jan-05, 1:11	1:00

Comments:

- Please carry out the tasks in the order specified.
- Note that every task that involves programming also requires some testing. **Note that the testing is part of the task**. Therefore, you should include the time it took you to test and correct each change in the respective task.
- Take as many breaks as necessary during the experiment.
- It makes no difference which computer you use or which Java environment you code your program with, as long as the same computer/environment are used for the duration of the experiment.
- Please do not concern yourself with "scoring well". This experiment is not meant to test your skills (although it may improve your skills). Take as long as it is necessary to complete each task.
- If you have any questions at any point, feel free to ask them, but please include the time it takes to correspond or discuss your questions in calculating the time it took to complete the task.

Thank you for participating in this experiment and good luck!

Table 1. Comprehension Task

(Task 1) Read and understand the implementation of the Clock problem using an object-oriented programming style:

(a) Read and understand the *State* design pattern (in Appendix). Please read the entire chapter from beginning to end, including the sample code, before engaging in other tasks.

(b) Read the description of the *Clock* problem given in Table 2, and read the sketch of the Java implementation given in Table 3.

(c) Understand the full Java implementation (Java source code) of the clock problem that has been provided to you at the start of the experiment.

(d) To ensure that you have understood the implementation, write a small test to ensure that "whenever button *b1* is pressed" the appropriate action is taken, for example by printing a message displaying the current time and the current state to the standard output.

Don't forget to keep track of the time it took you to carry out this comprehension task (Table 5 and Table 6).

Table 2. **The Clock problem**

Consider a digital clock with three display states: DisplayHour, DisplaySeconds, DisplayDate. The clock accepts input from button $b1$, which is used to change between states or to perform a specific action depending on the current state, as modelled in the following diagram:

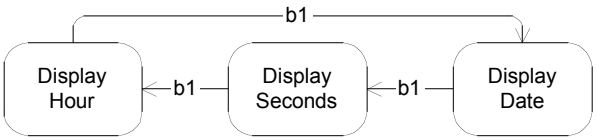


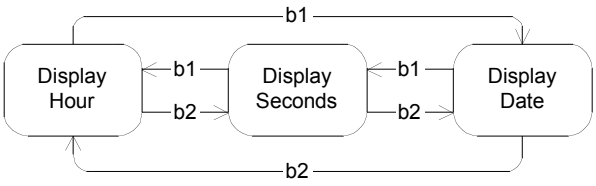
Table 3. **Sketch of the object-oriented implementation of the *Clock* problem**

Sketch of how to use the *State* design pattern to implement the solution to the *Clock* problem: Define a separate class for each state and use a context object to switch between the states. Specifically, write your program along the lines of the following object-oriented style (note: "style", this is not a complete program!):

```
class ClockContext {
  private int hours, minutes, seconds, dayInMonth, month;
  private ClockState currentState;
  public SwitchTo(ClockState NextState) {
    currentState = NextState;
  }
}
interface ClockState {
  void b1(ClockContext context); // button 1 pressed
}
class DisplayHour implements ClockState {
  public void b1(ClockContext context) { /* implement button 1 pressed */ }
}
class DisplaySecond implements ClockState {
  public void b1(ClockContext context) { /* implement button 1 pressed */ }
}
class DisplayDate implements ClockState {
  public void b1(ClockContext context) { /* implement button 1 pressed */ }
}
```

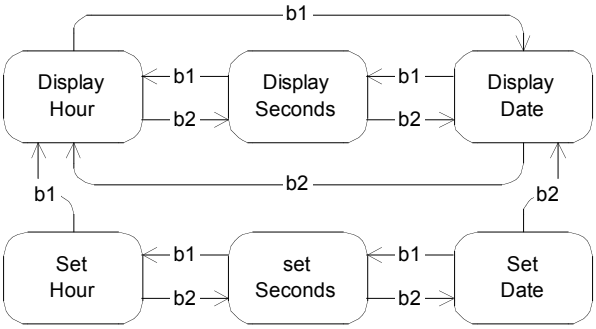
Table 4. **Tasks for evolving the *State* implementation**

(Task 2) Add a new button 2 to the *Clock* implementation: Add another button *b2* to the *Clock* implementation, so as to conform to the behaviour specified by the following illustration:



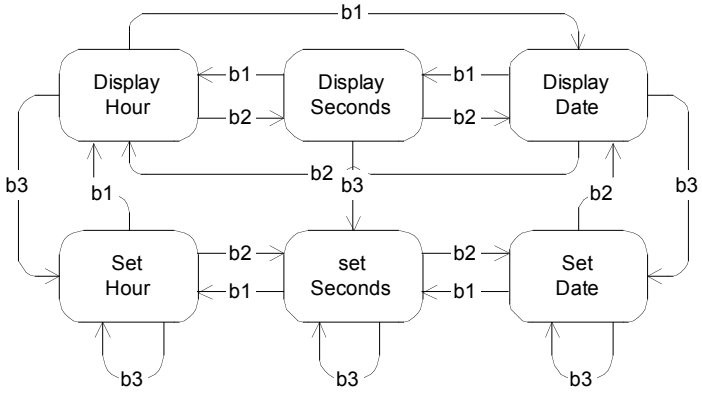
Upon completion of the change, write and execute a small test to ensure that "whenever button *b2* is pressed" the appropriate action is taken, for example by printing a message displaying the current time and the current state to the standard output. Also, execute the test for button *b1*, that has been implemented during (Task 1).

(Task 3) Add three new states to the *Clock* implementation: Add the states SetHour, SetSeconds and SetDate to the *Clock* implementation, so as to conform to the behaviour specified by the following illustration:



Upon completion of the change, write a small test to ensure that whenever "button *b2* is pressed" the appropriate action is taken, for example by printing a message displaying the current time and the current state to the standard ouput. Also, execute the same test for button *b1*, that has been implemented during (Task 1).

(Task 4) Add a new button 3 to the *Clock* implementation: Add a third button *b3* to the *Clock* implementation, so as to conform to the behaviour specified by the following illustration:



Note that in the SetHour (respectively SetSeconds and SetDate) state, pressing *b3* results in incrementing the minutes (respectively, seconds and days) with *1*.

Upon completion of the change, write a small test to ensure that whenever "button *b3* is pressed" the appropriate action is taken, for example by printing a message displaying the current time and the current state to the standard output. Also, execute the same tests for buttons *b1* and *b2*, that have been implemented in the previous tasks.

Table 5**. Diary**

Please use this table to record the time spent on each "session" of work, where "session" is a continuous time interval spent on a particular task. Each line in this table should represent exactly one continuous session. If you break up your task into more sessions than this table allows, please feel free to extend this table as necessary.

TASK	START TIME	END TIME	LENGTH

Table 6. **Summary sheet**

Your **last name**:

Your **first name**:

Programming languages you are familiar with:

Your experience in programming in these languages:

(Give a rating between -3 and +3. -3 = no practical experience, -2 = very bad, -1 = bad, 0 = acceptable , 1 = good, 2 = very good, 3 = excellent)

Language	Rating

Upon completion of all tasks, summarize the total duration of all tasks in this table. Add any comments you find relevant (increase table size if necessary.)

TASK	TOTAL LENGTH	COMMENTS
1		
2		
3		
4		

INSTRUCTIONS TO SUBJECTS – GROUP 2

Important !

Please read carefully this *Instructions sheet* from beginning to end before doing anything else.

The purpose of this experiment is to test the hypothesis put forth under the title "evolution complexity". There is no need for you to understand the theory; in fact, the integrity of the experiment is insured by ignoring the predictions we make.

Generally speaking, our hypothesis regards the time that certain evolution tasks will take to complete. To test this hypothesis, we ask you to carry out the tasks described in the Tasks tables (Table 1 and Table 4), to measure the time it took you to complete each using the Diary (Table 5), and to summarize the results in the Summary sheet (Table 6).

Before conducting this experiment, make sure that, along with this document, you are sitting next to a computer running a clear display of the digital clock, and that same computer has a working Java environment including a complete installation of Java, including the Java compiler and run-time environment.

Before you begin working on a particular task, please start a new entry in your Diary: write the task number under TASK and the current time under START TIME.

After completing a session, whether you decide to take a break or have finished a task, enter the current time in your Diary under END TIME. When you resume working, start a new Diary entry.

After completing all tasks, summarize the time it took you to carry out each task and write the total in the Summary sheet (Table 6).

Sample Diary

The following three entries in a Diary tell us you spent two 40-minute sessions on task no. 1 and one hour session on task no. 2:

TASK	START TIME	END TIME	LENGTH
1	1-Jan-05, 9:12	9:52	0:40
1	1-Jan-05, 12:20	13:00	0:40
2	2-Jan-05, 23:11	3-Jan-05, 1:11	1:00

Comments:

- Please carry out the tasks in the order specified.
- Note that **every** task involves programming, but also requires some testing. **Note that the testing is an essential part of the task**. Therefore, you should include the time it took you to test and correct each change in the respective task.
- Take as many breaks as necessary during the experiment.
- It makes no difference which computer you use or which Java environment you code your program with, as long as the same computer/environment are used for the duration of the experiment.
- Please do not concern yourself with "scoring well". This experiment is not meant to test your skills (although it may improve your skills). Take as long as it is necessary to complete each task.
- If you have any questions at any point, feel free to ask them, but please include the time it takes to correspond or discuss your questions in calculating the time it took to complete the task.

Thank you for participating in this experiment and good luck!

Table 7. **Comprehension Task**

(Task 1) Read and understand the implementation of the Clock problem using a procedural style:
(a) Read the description of the *Clock* problem given in Table 2, and read the sketch of the Java implementation given in Table 9.

(b) Understand the full Java implementation (Java source code) of the clock problem that has been provided to you at the start of the experiment.

(c) To ensure that you have understood the implementation, write and execute a small test to ensure that "whenever button *b1* is pressed" the appropriate action is taken, for example by printing a message displaying the current time and the current state to the standard output.

Don't forget to keep track of the time it took you to carry out this comprehension task (Table 5 and Table 6).

Table 8. The Clock problem

Consider a digital clock with three display states: DisplayHour, DisplaySeconds, DisplayDate. The clock accepts input from button *b1*, which is used to change between states or to perform a specific action depending on the current state, as modelled in the following diagram:

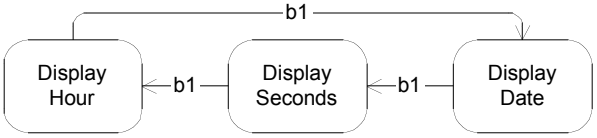


Table 9. **Sketch of the procedural implementation of the *Clock* problem**

Sketch of how to implement a procedural solution to the Clock problem: Define a Java class containing an enumeration of all the possible states, along the lines of the following procedural style (note: "style", this is not a complete program!):

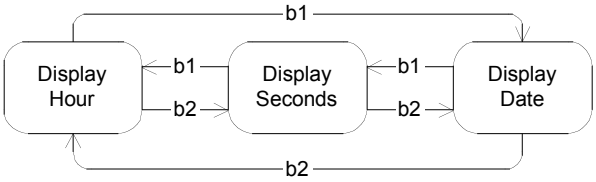
```
class ProcClock {
  enum states = {DisplayHour, DisplaySecond, DisplayDate, SetHour, SetDate};

  private states currentState;

  public void b1() { // button 1 pressed
    switch (currentState) {
      case DisplayHour: /*...*/;
      case DisplaySecond: /*...*/;
      case DisplayDate: /*...*/;
    }
```

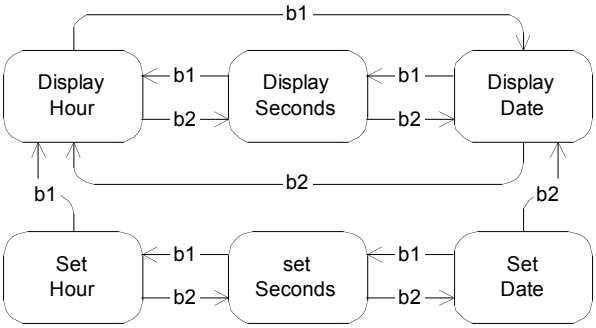
Table 10. **Tasks for evolving the *Clock* implementation**

(Task 2) Add a new button 2 to the *Clock* implementation: Add another button *b2* to the *Clock* implementation, so as to conform to the behaviour specified by the following illustration:



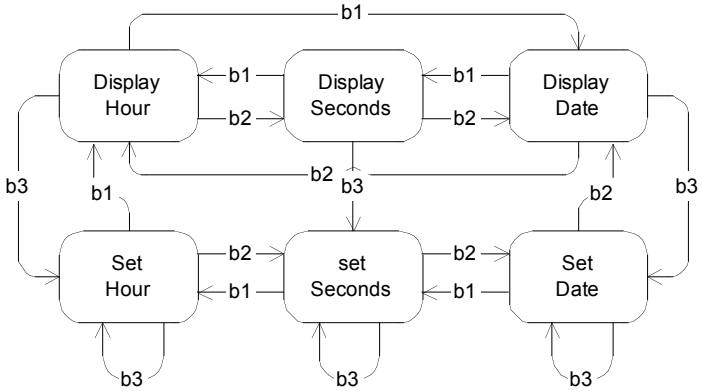
Upon completion of the change, write and execute a small test to ensure that "whenever button *b2* is pressed" the appropriate action is taken, for example by printing a message displaying the current time and the current state to the standard output. Also, execute the test for button *b1*, that has been implemented during (Task 1).

(Task 3) Add three new states to the *Clock* implementation: Add the states SetHour, SetSeconds and SetDate to the *Clock* implementation, so as to conform to the behaviour specified by the following illustration:



Upon completion of the change, write a small test to ensure that whenever "button *b2* is pressed" the appropriate action is taken, for example by printing a message displaying the current time and the current state to the standard ouput. Also, execute the same test for button *b1*, that has been implemented during (Task 1).

(Task 4) Add a new button 3 to the *Clock* implementation: Add a third button *b3* to the *Clock* implementation, so as to conform to the behaviour specified by the following illustration:



Note that in the SetHour (respectively SetSeconds and SetDate) state, pressing *b3* results in incrementing the minutes (respectively, seconds and days) with *1*.

Upon completion of the change, write a small test to ensure that whenever "button *b3* is pressed" the appropriate action is taken, for example by printing a message displaying the current time and the current state to the standard output. Also, execute the same tests for buttons *b1* and *b2*, that have been implemented in the previous tasks.

Table 11. **Diary**

Please use this table to record the time spent on each "session" of work, where "session" is a continuous time interval spent on a particular task. Each line in this table should represent exactly one continuous session. If you break up your task into more sessions than this table allows, please feel free to extend this table as necessary.

TASK	START TIME	END TIME	LENGTH

Table 12. **Summary sheet**

Your **last name**:

Your **first name**:

Programming languages you are familiar with:

Your experience in programming in these languages:

(Give a rating between -3 and +3. -3 = no practical experience, -2 = very bad, -1 = bad, 0 = acceptable , 1 = good, 2 = very good, 3 = excellent)

Language	Rating

Upon completion of all tasks, summarize the total duration of all tasks in this table. Add any comments you find relevant (increase table size if necessary.)

TASK	TOTAL LENGTH	COMMENTS
1		
2		
3		
4		

Additional comments (optional):