Automatic time measurement for UML modeling activities

Ira Diethelm[1], Leif Geiger[2], Christian Schneider[2], Albert Zündorf[2]

[1]Gaußschule
Löwenwall 18a
38100 Braunschweig

[2]Fachgebiet Softwareengineering
Universität Kassel
Wilhelmshöher Allee 73
34121 Kassel

(ira.diethelm | leif.geiger | christian.schneider | albert.zuendorf)@uni-kassel.de

www.se.e-technik.uni-kassel.de

Abstract:

In order to improve learning and teaching processes in computer science, we need to analyze current processes qualitatively and quantitatively. Such an analysis may finally allow to evaluate empirically the effects of new approaches and of changes to the process in comparison to previous processes. First of all, the learning effects may be measured using usual examination schemes. However, for deeper insights, the measurement of process results should be correlated to measurements during process execution. This paper outlines the current state-of-the-art in automatic time measurement in CASE tools and what may be achieved in the near future. This is done with respect to empirical studies for learning and teaching processes.

1 Introduction

In his PhD Thesis, Carsten Schulte has proposed detailed protocols of a proband's activities during UML modeling as a means for empirical studies in the area of didactics of computer science, cf. [Schu03]. Carsten Schulte used

- video protocols of the class room at all,
- video protocols of the screen contents of each proband's computer during exercises,
- audio tapes protocolling probands' discussions, and
- internal command protocols of the employed CASE tool.

After the lessons, Carsten Schulte and his team had to evaluate all these protocols manually. For the CASE tool usage, they did a replay of each session and in a raster of a

minute, they categorized the probands' activities according to the topic under work and according to the kind of activity performed (e.g. coding, bug fixing, discussion, ...). As one sees easily, such an evaluation of session protocols is extremely tedious. In order to facilitate empirical studies based on such protocols, this paper explores the possibilities and restrictions of automatic session protocol evaluation via CASE tools.

2 Automatic time measurement

In principle, it is pretty simple to add automatic protocol features to a certain CASE tool. Most CASE tools actually protocol all user interaction already e.g. for undo/redo or for recovery functionality. Thus, all that needs to be done is adding time stamps and logging of all the operations.

Note, this kind of automatic time measurement has a systematic fault, since it just measures editing activities. The times when the proband does not edit but he is thinking about a certain problem or he is discussing a topic with some team mate or he is just out for a break or he is in a meeting or he is interrupted by a phone call or he finds a brilliant solution to a problem during sleep at night, all these non-editing activities are not covered. Covering more of these non-editing activities needs other observation techniques that probably are able to enhance the measurement. However, we have no idea, how to automate the evaluation. Luckily, some recent empirical studies give hints, that in UML projects the amount of editing activities seems to be closely related to the amount of non-editing activities. If this holds, the automatic time measurement could be a reasonable indicator of the overall effort for UML based modeling. Accordingly, this paper assumes that automatic time measurement is a valid means for empirical studies on modeling activities.

Note, a simple protocol of time stamped user commands provides only limited information for empirical studies on modelling activities. To provide substantial value, the protocol must allow to retrieve detailed information of the part of a UML model that is changed, how it is changed, and in the optimal case why it is changed. Ideally, the changes are related to specific kinds of tasks and it is later on possible to relate them to specific elements of the edited UML specification. We will show how such information may be obtained and how this information may be exploited at the example of the Fujaba CASE tool.

The Fujaba CASE tool has a plug-in called CoObRA that adds undo/redo, recovery, and versioning functionality to it (see [Schn03]). CoObRA is an acronym for Common Object Replication frAmework. Basically, Fujaba employs a dedicated object structure, the so-called meta-model to represent the UML diagrams edited by a user. During editing e.g. a class diagram the user adds new objects to the internal meta-model, removes objects from the meta-model or changes the values of certain attributes of certain meta-model objects. All these operations are protocolled at this level of detail together with detailed timing information.

Note, in Fujaba layout information is stored as part of the logical meta-model and thus the corresponding operations are also covered by the CoObRA protocol.

Since certain operations results in a large number of changes to the internal object structure

of the Fujaba CASE tool, our protocol groups all changes into so called user commands. This raises the level of abstraction for the analysis while it still maintains exact information about the modified data.

3 Exploiting the change protocols

3.1 Session replay

Provided with detailed undo/redo information it is for example possible to revert a whole user session and to re-execute it step by step, in slow motion, fast-forward, or even in backward mode. This could be exploited for manual analysis of user sessions as in [Schu03].

3.2 Relating changes to specification elements

For further analysis of change protocols, it is mandatory, to identify the edited specification parts. This might be a certain fraction of a class diagram or e.g. an activity diagram modeling the behavior of a certain method. Note, sometimes the protocol data might not properly identify the edited parts of the specification, e.g. if internal statistical data is targeted. However, due to our experiences, in almost all cases it is very simple to identify the edited diagram element and to relate it to a certain part of the overall specification. This could be done on a coarse grain level e.g. per method body of on a fine grain level e.g. down to the different parts of a sequence or of an activity diagram.

3.3 Relating changes to project phases

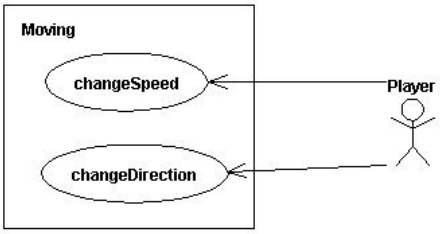
If the user follows a certain process, it is even possible to relate our protocol data to different kinds of activities like requirements definition, analysis, design, implementation, or maintenance. In our courses, we use Fujaba with the so-called Fujaba Process ([DGZ04a, GSZ03]). Fujaba supports this process by providing a document centered view of a project handbook that guides the user through the development process. In this view, the user starts in a dedicated chapter of the project handbook by editing use cases, cf. Figure 1. For each use case, Fujaba automatically adds a pre-formatted section for a textual use case description. This is then filled, manually. (The example is taken from a guided tour created for one of our courses at University of Kassel. It deals with a very simple rule in a board game called Mississippy Queen, where one travels a changing river with a steamer).

4. Analysis Phase

In the following subchapter we place the top level package diagram for use cases.

4.1. Description of Moving

<to be filled>



4.1.1. Description of usecase changeSpeed

Scenario changeSpeed1

Start situation: A new player gets the turn.

Invocation: The game asks the player to change the current speed of his steamer.

Step 1: The player increases the speed three times.

Step 2: The player decreases the speed one time

Step 3: The player stops changing the speed and the coal is decreased by one

Result situation: Speed has increased by two and coal has decreased by one.

Figure 1: Requirements definition in the Fujaba Process

A special user command turns such a textual use case description into the outline of an UML interaction diagram allowing to elaborate the use case description. In our case these are so-called story boards, a combination of UML activity and UML collaboration diagrams, cf. Figure 2.

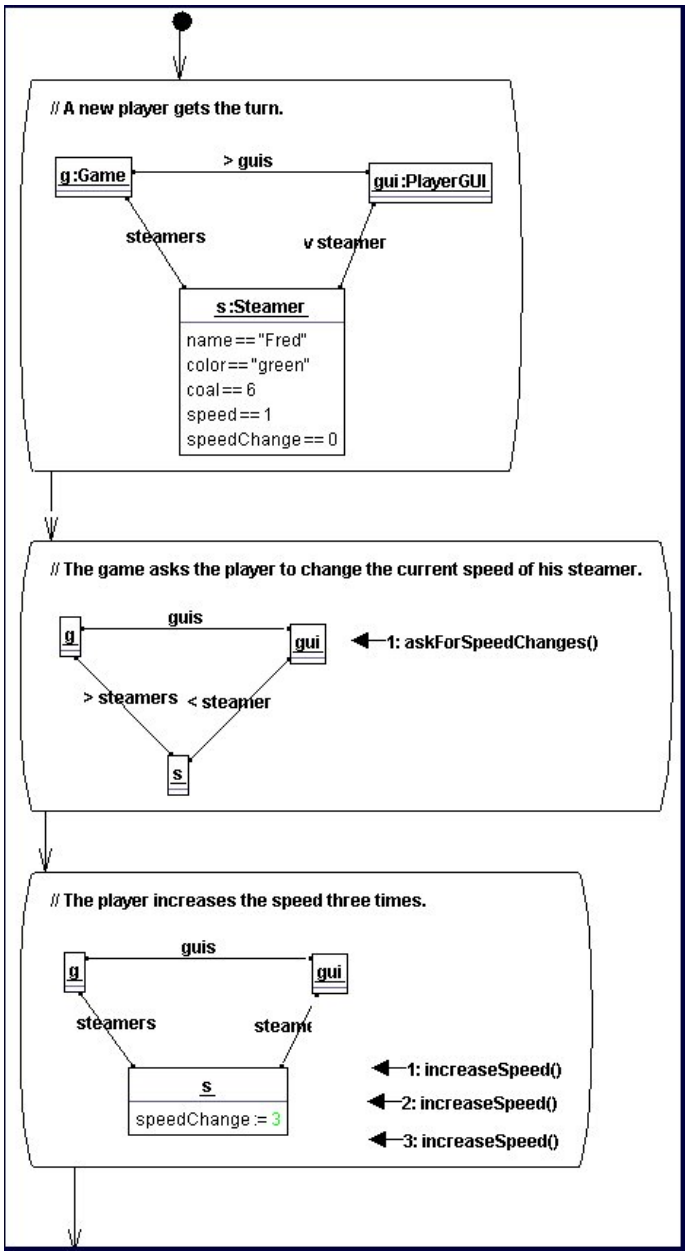


Figure 2: Analysis with story boards

From such a story board, specific user commands derive a class diagram and a test specification. The class diagram already includes declarations for all methods employed in the scenarios, cf. Figure 3.

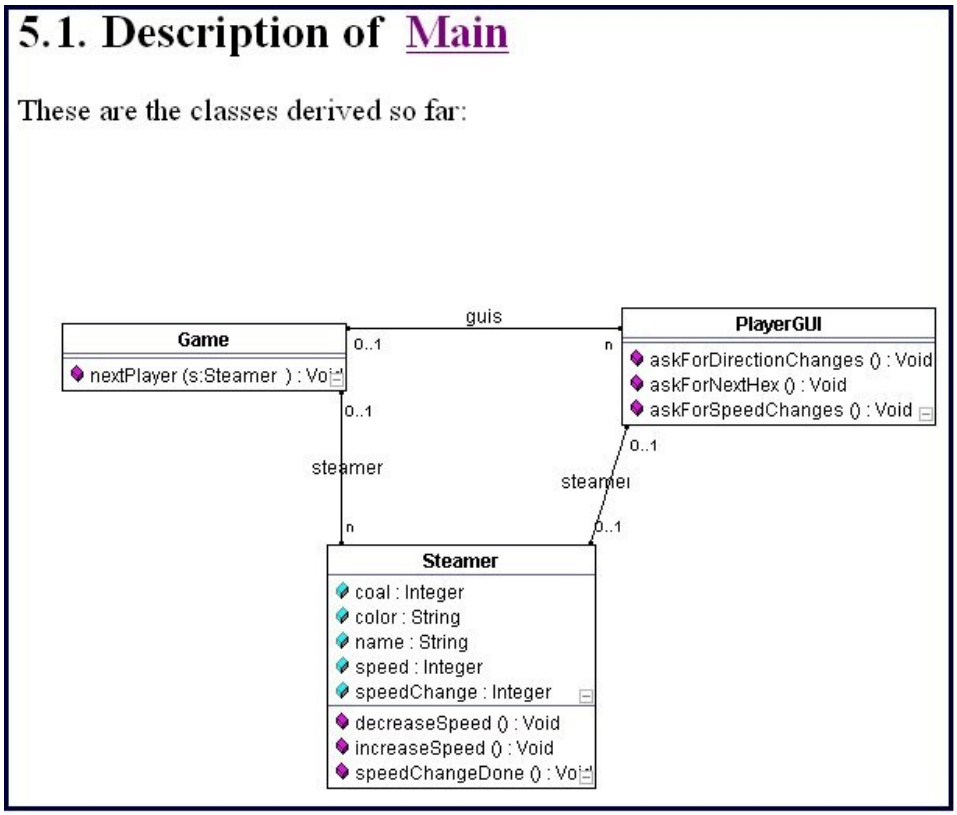


Figure 3: Class diagram derived from the story board

Now, the user has to implement the method bodies appropriately, cf. Figure 4. Once the functionality is provided, the user generates code from his specification, compiles it and tests it against the test derived from his story boards.

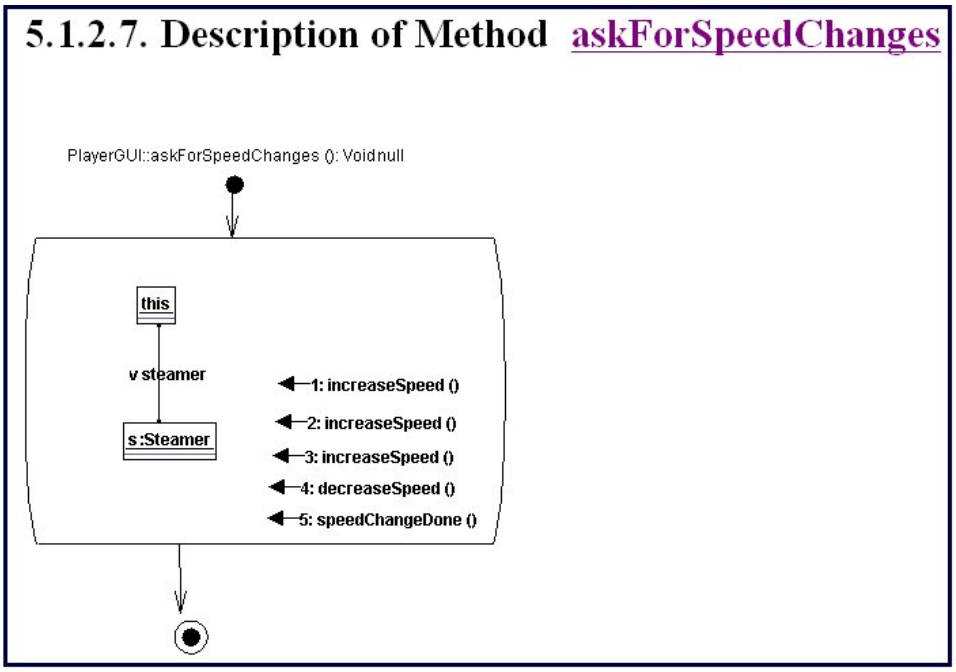


Figure 4: Method body specification

Following this process allows to relate editing activities to project phases: editing a use case or a textual use case description is considered as a requirements activity. Editing a story board belongs to the object oriented analysis phase. And editing a method body means implementation effort. Finally, any activity after successful compilation and after running the first test may be considered as a testing, bug-fixing, and maintenance activity.

Note, the Fujaba process is an use case driven and an iterative process. This means, the developer realizes one use case after the other. Thus, after implementing the methods employed in a certain use case, all testing and bug fixing activities are related to the same use case until the developer starts editing another use case.

3.4 Relating changes to tasks

As already discussed, the Fujaba Process may be used in an iterative way and by project teams. In this case, different team members may work at different use cases. Each team member may work on just one use case at a time. As outlined above, in Fujaba it is still possible to relate most editing activities to specific use cases. In the case of editing a textual use case scenario, this is trivial. The same holds for a story board, since a story board always elaborates a certain use case. In case of method bodies the situation is not always clear. However, in the Fujaba Process each use case describes a certain system

functionality that is realized by a dedicated method. Editing this method is clearly related to the corresponding use case. Similarly, the story board elaborating a given use case may employ some additional methods. If these methods have not yet been used in other use-cases they may be related to the current use case. Finally, we derive automatic tests from each story board. Bug-fixing activities caused by failure of such a test may also be related to the corresponding use case. Thus, in most cases we are able to relate editing activities to dedicated steps in the realization of a dedicated use case, even in an iterative, multi user process.

Note, due to our detailed protocol data it might be possible to analyze which parts of a specification are modified in response to a failed test. If only a method body is edited, it was an implementation problem. If the class diagram is changed, it was a design problem. If even a story board or a use case scenario needs to be adapted, it is an analysis or requirements problem, respectively. This might be related to the overall effort for fixing the bug. This may allow to study the question, whether in iterative processes the assumption still holds that bugs in early phases cost a magnitude more than bugs in later phases.

3.5 Summing up editing times

Until now, we have just related editing activities to specific tasks in the modelling process. In addition, we have some experiences in summing up the times for these editing activities. As already mentioned, Fujaba's internal change protocol provides time stamps for all editing activities. Usually, these time stamps show phases of intensive editing where subsequent editing steps have very short time distances (some seconds) followed by certain gaps, where no editing activity is recorded (for several minutes). As discussed in the introduction, the tool is not able to guess what is going on during these gaps. The user may be thinking or working on the problem with pencil and paper or discussing it with his team mates. Or the user may just take a break. As discussed, we just measure the editing activities and hope that they resemble the over all modelling effort, closely.

Based on the time stamps of our activity protocol, there are multiple ways to sum up the time spent on the different tasks on a project. A simple scheme might e.g. assume a minimal time required for a single editing activity e.g. 10 seconds and a maximum time between two editing activities that is not considered as a break, e.g. one minute. Accordingly, if we record only a single editing command, we add the 10 seconds to the time spent on the corresponding task. Second, if we do not record editing for more than e.g. one minute, we assume that the user takes a break. In that case we might add the time span from the first activity after the previous break until the last activity before the new break plus 10 seconds to the corresponding task. If the task changes during a sequence of activities e.g. between step a and b, we might cut the interval in the middle between step a and b.

Using such an approach, it might be possible to measure the time spent on a specific task with some realistic precision. This precision might be adjusted by empirical studies collecting more precise time data with alternative (manual) means.

If such an automatic time measurement delivers reliable data, there are multiple applica-

tions of such an approach in the area of software engineering and project management. For example, statistical data collected in this way from several projects might be used as a basis for effort estimations and project planning for new projects. Similarly, time data collected during project execution might be used for project tracking, cf. Figure 5. Each time, a certain task is completed, the tool might relate the measured time spent on that task with the time estimated for that task. Such a comparison may allow to indicate phases of good progress as well as delays for certain tasks that may need management intervention.

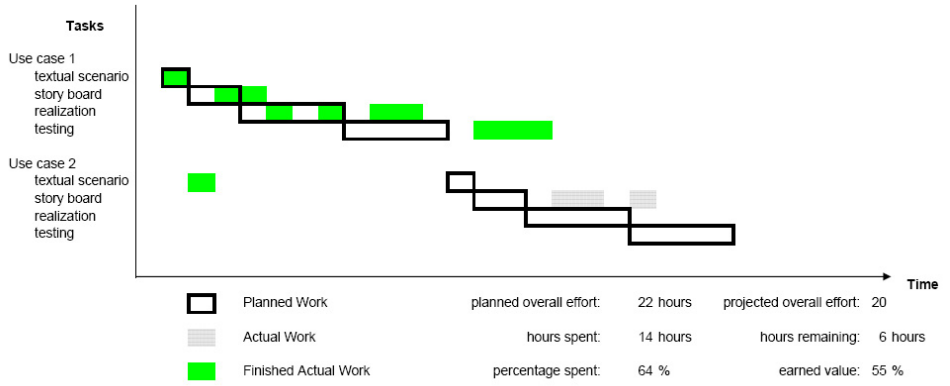


Figure 5: Possible exploitation of automatical time measurement

The example in Figure 5 outlines some possible project plan view based on such an automatic time measurement. The hollow bars indicate planned efforts for two use cases. These effort estimations could be derived from earlier projects. The green bars show actually spent time related to tasks. Note the gaps in these bars that might be caused by phases of thinking and discussions or e.g. by lunch and coffee breaks. The grey bars indicate time spent on task that are not yet completed. In this example, the estimated over all effort is 22 hours. The measurement of the actually spent time sums up to 14 hours so far. Thus, in this example already 64% of the project budget are consumed. Summing up the percentage of completed tasks results in only 55%. However, only 11 hours (half of the budget) have been spent in order to complete 55% of the tasks, thus in this example the project seems to be slightly ahead of schedule. Such situations could be reflected in an adjusted projection of the required overall effort and of a projection of the time required for completion.

Such an exploitation of the automatic time measurement is especially of interest for the area of software engineering. However, we are confident, that similar analysis mechanisms could also provide valuable input for more general empirical studies on modelling activities.

3.6 Editing patterns

In our work with the Fujaba environment in class rooms and by supervising student projects we frequently observe typical editing patterns. For example, during editing a story board the students frequently detect the necessity of an additional object within their story board while editing some later activity. Adding an object to the story board in this phase requires to go back to the first activity, to add the object there and then to copy this change forward to the next activities step by step until the student reaches his former point of editing, again.

Such editing patterns are very interesting from an analysis point of view since they indicate situations or points in time when the student has discovered a misconception in his solution. This might e.g. give hints for insufficient group discussions on the scenario.

Fujaba provides some functionality for pattern detection within static source code. Currently, this functionality is extended towards analysis of program execution traces. Similar techniques might be usable for the analysis of editing patterns, too.

4 Summary

This paper outlines the automatic protocol features of the Fujaba CASE tool. This automatic protocol features enable us to replay user sessions and to relate editing activities to different project phases and to dedicated use cases. This is supported for iterative projects with multiple developers working on a common project in parallel. On this basis, numerous other analysis mechanisms may be realized.

We propose to use these automatic protocol evaluation features of Fujaba to automate protocol evaluation in empirical studies like the one of [Schu03] .

References

[DGMZ02] I. Diethelm, L. Geiger, T. Maier, A. Zündorf: Turning Collaboration Diagram Strips into Storycharts; Workshop on Scenarios and state machines: models, algorithms, and tools, ICSE 2002, Orlando, Florida, USA, 2002.

[DGZ02] I. Diethelm, L. Geiger, A. Zündorf: UML im Unterricht: Systematische objektorientierte Problemlösung mit Hilfe von Szenarien am Beispiel der Türme von Hanoi; in Forschungsbeiträge zur ”Didaktik der Informatik“ - Theorie, Praxis und Evaluation, GI-Lecture Notes, pp. 33-42, 2002.

[DGZ04a] I. Diethelm, L. Geiger, A. Zündorf: Systematic Story Driven Modeling, a case study; Workshop on Scenarios and state machines: models, algorithms, and tools, ICSE 2004, Edinburgh, Scottland, 2004.

[DGSZ04b] I. Diethelm, L. Geiger, C. Schneider, A. Zündorf: Measurement of modeling abilities; Concepts of Empirical Research and Standardisation of Measurment in the Area of Didactics of Informatics (CERSMADI), Dagstuhl, Germany, 2004.

[Fu02] Fujaba Homepage, Universität Paderborn, http://www.fujaba.de/.

[GSZ03] L. Geiger, C. Schneider, A. Zündorf: Integrated, Document Centered Modeling in Fujaba; 1st International Fujaba Days, Kassel, Germany, 2003.

[Hu00] P. Hubwieser: Didaktik der Informatik - Grundlagen, Konzepte, Beispiele; Springer Verlag, Berlin, 2000.

[Hu98] Watts S. Humphrey: Introduction to the Personal Software Process; Addison-Wesley, Amsterdam, 1998.

[KNNZ00] H. Köhler, U. Nickel, J. Niere, A. Zündorf: Integrating UML Diagrams for Production Control Systems; in Proc. of ICSE 2000 - The 22nd International Conference on Software Engineering, June 4-11th, Limerick, Ireland, acm press, pp. 241-251, 2000.

[life02] life^3-Homepage, Universität Paderborn, http://life.uni-paderborn.de/.

[Schn03] C. Schneider: CASE Tool Unterstützung für die Delta-basierte Replikation und Versionierung komplexer Objektstrukturen; Diploma Thesis, Corolo Wilhelmina zu Braunschweig, Braunschweig, Germany, 2003.

[Schu03] C. Schulte: Lehr- Lernprozesse im Informatik-Anfangsunterricht; PhD Thesis, University of Paderborn, 2003.

[SN02] C. Schulte, J. Niere: Thinking in Object Structures: Teaching Modelling in Secondary Schools; in Sixth Workshop on Pedagogies and Tools for Learning Object Oriented Concepts, ECOOP, Malaga, Spanien, 2002.

[Zü01] A. Zündorf: Rigorous Object Oriented Software Development; Habilitation Thesis, University of Paderborn, 2001.