Empirical Assessment of UML Static Object Diagrams

Marco Torchiano
Dipartimento di Automatica e Informatica
Politecnico di Torino
Italy
marco.torchiano@polito.it

Abstract

The essential UML logic diagrams are the class diagrams: they represent the classes of objects that make up a program. Object diagrams are used as the basis to show scenarios of dynamic evolution of the software. The use of object diagrams to depict static structure is very rare.

The purpose of this study is to investigate whether the use of static object diagrams can improve the comprehension of software systems.

We conducted a study with 17 graduate students during a software engineering course. The students were asked to answer questions about a software system. The system was described either with a class diagram or with both a class diagram and an object diagram. The student asked multiple choice questions on four different systems.

This study revealed that there is a statistically significant difference in the comprehension achievement for two of the systems. The effect of the presence of object diagrams can be classified of medium size.

These results allow us to formulate new research questions that will guide our future work in this area.

1. Introduction

The object-oriented paradigm has gained a wide range of applications among the software engineering and developer community. The main factor is the diffusion of UML [5], a sort of “lingua franca” used to speak about software.

UML consists of several notations that cover both the logical and physical structure of software. In particular, focusing on the logical notations, UML can describe both static and dynamic features. It provides models at two different levels of abstractions: classes and objects. At class-level we can think of a program as made up of classes and their associations: actually programmers, to define the behavior of a program, write classes that refer to each other. Though, if we observe the program during its execution (at run-time) we see it as made up of objects linked to each other and interacting; this is the object-level perspective.

Currently the prevailing perspective in the object-oriented community is the class-based. That is classes models are used more frequently than object models. If we look at the UML books [5] they describe in great detail the notations to draw class based diagrams while object diagrams are considered of secondary importance, they are used only to provide examples of the dynamic evolution of the system.

The purpose of this study was to conduct a preliminary investigation on the benefits deriving from the use of static object diagrams for the comprehension of software. Our goal is to collect first evidence of some benefic effect of the static object diagrams and to help focus more precise research questions.

The class-centered approach gives good results when applied strictly to software development, but there are many applications, ranging from data modeling, to process modeling to enterprise modeling, which demand for a different approach.

Those applications produce models that consist of a large number of objects: instance models are then required. UML can model both static and dynamic features of software products. It provides notations for two different levels of abstractions: class level and instance level. Table 1 summarizes this classification.

Table 1. Classification of UML diagrams.

	Static	**Dynamic**
Class	Class	Statecharts, Activity
Object	Object	Sequence, Collaboration

Notations at the class level of abstraction are used to describe features that apply to all instances.

Notations at the object level of abstractions are used to provide examples of how the features described at class level actually apply in special cases; in short they provide examples.

Considering the static diagrams, the essential difference among class and object diagrams is the software life-cycle phase addressed: the former depict the compile-time structure of the software, the latter depict the run-time structure of software.

Since its standardization by OMG, UML has been adopted by practitioners as a sort of "*lingua franca*" to describe software artifacts. Under this perspective UML is used, often in an informal way, to communicate among designers. Also UML is used ubiquitously in research papers to describe object-oriented software.

In addition, the object diagrams are very useful in several cases, especially when UML is used for analysis purpose and therefore it does not describes directly software artifacts. E.g. [1] presents an approach for business processes and workflow modeling based on hierarchical instance models. Hierarchical instance models can be describes by means of annotated object diagrams.

Typical examples of instance models are software architectural models. The architecture of a software system in its simplest form is made up instances of components and connectors linked together [8].

The notations that are part of UML have been object of several empirical studies. Some investigations focus on the use of diagrammatic representations in software engineering. In [6] an evaluation is presented of the suitability of different UML dynamic modeling diagrams for different modeling domains. In [3] there is a discussion about the best way to decorate diagrams to convey additional information on the relationships among the elements.

2. Objective of the study

We focus on the use of UML diagrams as communication tools. Therefore we are not interested in the formal underpinnings of the notations or in the precise use of them. We are more interested in the use of UML to describe partial and incomplete models.

When talking and reasoning about programs it is very useful to have object diagrams as a reference notation. This need emerged during object oriented programming lectures. In particular when helping the students to bridge the gap that separates the classes they write and the objects that actually form the program at run time.

From our perspective UML is mainly a communication tool; it is not used in a precise way. Often diagrams a sketched by hand on a blackboard or on transparencies to clarify some point or to answer questions from the students.

Therefore we investigate how the comprehension of a software system can be improved using UML diagrams. In particular the objective is to help the subject understand how an object oriented program works at run-time.

Let us consider an example of a program structure that represents the train timetables and keeps track of the actual times of passage at the stations.

The class structure of such program is represented in Figure 1. It contains the class TrainPath that is characterized by a code; it represents the path of train. This class is related to class Station through the association Statrs_at. The class Station has also a recursive association that is used to define the sequence of stations that constitute a path. The actual timings of the trains are represented by the classes Train and Passage. The former represents a concrete train that is traveling in a given date; the latter represents the actual passages of trains at stations and records the time of such passages.

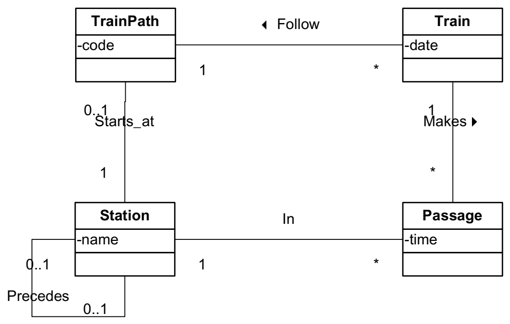

Figure 1: Class diagrams of Trains example.

The run-time configuration of objects and links corresponding to the above class diagram can be quite complex as we can see in Figure 2.

A typical question it is possible to ask about this system is: "starting from a TrainPath object, to reach the third Station object, how many objects are visited (excluding the train path and the 3rd station themselves)?"

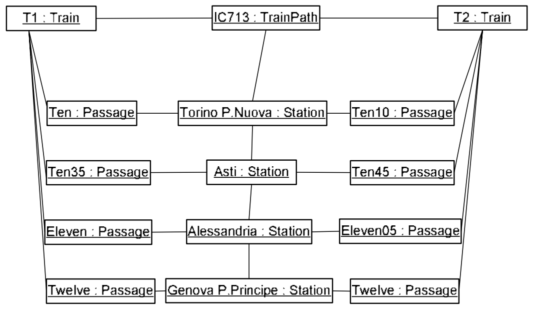

Figure 2: Object diagram for Trains example.

To answer the first two questions is sufficient to look at the cardinality of the associations in the class diagrams, or to observe the example provided by the object diagram.

To answer the third question it is much easier to reason on the object diagram than on the class diagram: to reach the third Station object you have to move to the first station, then to the second, and finally to the third one. Two intermediate objects are visited in this process.

To answer the fourth question it is easy to trace the operations on the example provided by the object diagram. For instance, starting from the Train object T1, first we have to navigate to the corresponding TrainPath object IC713, so we can access the ordered list of stations. Then we need to navigate through the Station objects; for each Station object we have to print its name and then navigate to all linked Passage objects, pick the one that is linked to the Train object T1 (the one we started from) and print the time of passage.

In this case we have to be careful in selecting the appropriate Passage object; the fact that there can be several Passage objects is evident from the object diagram. The same information is encoded into the cardinality of the associations "In" but it is less evident.

On the basis of the previous observations, we apply the GQM approach [2] to precisely formulate the high level goal of the experiment, that is:

> To analyze *the static object diagrams of UML* in order to *evaluate* with respect to *the comprehension of the software system* from the point of view of *the software maintainer* in the context of *a software engineering classroom.*

The central high level hypothesis of this study is that the presence of object diagrams improves the comprehension of the software system under study.

3. Method

The experiment controls a single factor (or independent variable): OD that indicates whether the object diagrams are present in the description of a software system.

For each task we measure the score (SC). It consists of the number of correct answers. There are four questions per task; therefore the score can range from 0 to 4.

The questions aim at assessing the level of comprehension of the system that the subjects acquired.

The purpose of this study is to identify initial evidence and confirm the line of research. Therefore we formulate only one simple hypothesis.

H1) The scores achieved in presence of the treatment (object diagrams) is higher than in absence of it. We can formulate the null and alternative hypotheses as follows:

$H1_0$: $\mu(SC+) \leq \mu(SC-)$

$H1_a$: $\mu(SC+) > \mu(SC-)$

Being SC+ the population that performed a task with object diagrams and SC- the population that did it without them.

We believe the presence of additional information (i.e. that conveyed by the object diagrams) naturally improves the performance. Therefore we formulate a unidirectional hypothesis.

The subjects are the 17 graduate students in their first year of a master in computer science. The experiment was conducted in the context of a software engineering course.

The course contents cover the classic SE topics: development process, analysis, design, UML, test.

In particular the experiment tasks were designed to serve as a check-up exercise on the UML knowledge acquired by the students.

We took special care in designing the tasks to make them useful exercises in the context of the course. The exercises were filled in anonymous form, serving mainly as self-evaluation tools.

The experiment was conducted as a regular class exercise; the students were allowed to have limited communication with each other because so they were used to.

The experiment was carried on using a paper-based questionnaire. The questionnaire contained

- four exercises (the tasks) consisting of
  - A short textual description of the system
  - A class diagram
  - Optionally an object diagram
  - Four questions aimed at assessing the comprehension of the software
- three post-experiment questions aimed at measuring the perceived usefulness of object diagrams

We decided to measure the comprehension of the software system by asking questions on the system.
The first task (FS) describes a file system . The elements of the file system can be folders, files, and links. Folders can contain other elements, while links refer to other elements in the file system.

The second task (R) represents simplified geographic information. It describes maps made up of cities connected by means of roads. Each Road starts and ends in a city. Roads are characterized by a length.

The third task (T) models trains and their paths; it is described in detail in section 2.

Finally the forth task (C) describes a catalogue. A catalogue of a category of items (e.g. cars) describes items (e.g. cars models) based on a set of features (e.g. number of doors) that can have a set of possible values.

For all of the four tasks, the questions are similar to those presented in section 2. They have been reformulated as multiple choice questions; each question has four possible answers, only one being the correct one.

We adopted a factorial design, two groups, one treatment, and four tasks. Each group had half of the tasks with the treatment (presence of object diagram) and half of the tasks without.

The group 0 had tasks FS+, R-, T+, and C-, while the group 1 had tasks: T-, C+, FS-, R+. The treatment presence being indicated by a "+" and its absence by a "-"

4. Results

Among the 17 subjects that participated in the experiment all delivered the questionnaire filled in at the end of the experiment. Although the size of the population is small, we have a 100% response rate.

The score achieved by the subjects in the experiment are summarized textually in Table 2. Here we report the means of the scores.

We can observe that for the first three tasks the means of the scores achieve in presence of object diagrams (group 0 for task FS, group 1 for task R, group 0 for task T) are higher.

The scores of the four task range from 1 to 4. No subject scores a zero in any task. Only one subject scored the maximum (4) on all four tasks.

Table 2. Summary of results.

Group	Subj.	FS	R	T	C
0	9	3,00	3,00	3,44	2,67
1	8	2,38	3,13	3,00	2,50
SD		1.10	0.75	0.83	0.94
	17	**2,71**	**3,06**	**3,24**	**2,59**

We want our experiment to have a meaningful statistical power. We require the statistical power of the tests (1-β) to be not lower than 75%. Given a sample size of 17 subjects and the standard deviations presented in Table 2, we must accept an α-level of 17%; this result is computed for the t-test, and it is conservative for more powerful tests.

We can now proceed in the analysis of the data to reject the null hypothesis $H1_0$. Since the conditions for applying the t-test do not hold, we need to choose a non-parametric test, in particular the Mann-Whitney test. We test directional hypotheses and therefore we use one-tail significance values. The results of the statistic tests are summarized in Table 3.

For the task File System the median of the score with object diagrams is 4, without object diagrams is 2. This difference is statistically significant with a p value of 0.14.

For the task Roads both median are equal to 3, but the 85% confidence interval ranges from 0 to 1. There is no statistically significant difference.

For the task Trains the medians with and without object diagrams are 4 and 3 respectively. This difference is statistically significant with a p value of 0.16.

Finally, for the task Catalogue both medians are equal to 3. There is no statistically significant difference.

Performing the analysis for each task separately we were able to identify differences in only two cases of which only one statistically significant.

To enhance the capability to detect the capability to reveal the effects of the treatment we perform a further analysis. The main limitation in the experiment is the sample size, thus we attempt to reduce it. We consider the four tasks as if they were four repetition of the same task; as a result we have a single task with four times the original subjects. The correctness of this approach was verified using the Kruskal-Wallis test that revealed no significant difference between the four tasks.

In three cases the tests revealed a significant difference in the effects of the two treatments. To evaluate the practical implications we report also the standardized effect size (Cohen's d) [4]. In the three statistically significant cases we observe a small (all tasks) or medium sized effect (tasks File System and Trains).

Table 3. Statistical tests results.

	Median			
Task	**SC+**	**SC-**	**P value**	**d**
FS	4	2	**0.1416**	**0.57**
R	3	3	0.3778	0.17
T	4	3	**0.1622**	**0.53**
C	3	3	0.4045	-0.18
all	3	3	**0.0974**	**0.28**

4.1. Threats to validity

The main threat to the **conclusion validity** of this study is represented by its low statistical power: we have too few data points therefore we are forced to accept high alpha levels to be able to reveal any effect. We accept this risk being a preliminary study and plan to replicate the experiment with more subjects. We used non-parametric tests; therefore we avoid the risk of violating the assumptions of statistical tests

The main threat to **internal validity** consists of the social threats: the subjects were used to communicate with each other during class exercises, they were asked to reduce the communication, and nevertheless they did communicate. This factor could have reduced the difference among the groups, thus in absence of this threat the differences revealed by the experiment can but be emphasized. The threat of maturation of the subjects during the experiment is addressed presenting the tasks in different order to the two groups. We have no threat deriving from the selection of the subjects since all students in the class participated in the experiment. Since we had a null drop out rate, the mortality of subjects is not a threat in our case.

For **construct validity**, we avoided the threat of a mono operation bias by providing the students with four different types of tasks that represent a significant range of software systems. We have no hypotheses guessing threat since the experiment was presented as a normal class exercise and the subjects were not informed of the hypotheses before the experiment. We had no evaluation apprehension because the exercise was performed in an anonymous way and was presented as a means for self evaluation.

As far as **external validity** is concerned, the experiment was performed with students with limited programming experience; we believe this is the main threat to generalizing the results to other contexts. Though, there is evidence that the improvements observed in experiments with students is similar to that observed in experiments with professionals [7]. Another issue in generalizing the results to industrial settings stems from the simplicity of the tasks. We acknowledge that we proposed simple problems: nevertheless they contain typical features of real-world problems.

5. Discussion

In general we can confirm a significant positive effect of relevant size of the object diagrams on the comprehension. Nevertheless the experiment revealed differences in the significance and magnitude of effect among the tasks. Therefore we need to analyze the tasks case by case, to formulate hypothesis on the causes of this difference.

For task File System we have a statistical significance and a medium sized effect. Clearly the object diagrams improved the comprehension of the system. The software consists of a tree-like structure of the objects. This is achieved by means of an indirect self-loop association. Actually the system contains an instance of the structural design pattern composite. Often the descriptions of patterns contain static object diagrams.

For task Roads the experiment didn't reveal any significant effect of object diagrams. This example is made up of two classes and two associations forming a loop that allows representing a graph. This system is very small (also compared to the other systems): it comprises just two classes.

For task Trains we have a statistical significance and a medium sized effect. Clearly the object diagrams improved the comprehension of the system. The software consists of a four classes that form a cycle and an additional self-loop association on class Station. This system presents a type-instance duality: TrainPath and Station describe a "type" while Train and Passage can be seen as "instances" of this type.

For task Catalogue we have no evidence of an improvement deriving from the use of object diagrams. Also this system contains cycles (but no self-loop association) and presents a typical type-instance duality.

The above observations are useful to formulate new hypotheses about when object diagrams are most useful. We could identify the following:

- in presence of structural design patterns
- in presence of direct or indirect self-loop associations that generate a graph like structure,
- in presence of a cycle made up of three or more classes and associations,
- in presence of a type-instance duality combined with self-loop associations.

It is also possible to identify counter-examples (where the use of static object diagrams does not provide a significant benefit):

- with very simple structures and a few classes.
- without self-loop associations.

6. Conclusions

The initial research question was whether object diagrams had impact on the comprehension of software system. We can conclude that in general the presence of object diagrams can medium size effect on the comprehension of software system with acceptable statistical significance. In particular the experiment revealed the effect only for certain types of systems as discussed in the previous section.

The practical result of this research is to foster an increasing attention towards the static object diagrams as a means of communication. In particular for system with certain features (e.g. self-loop associations).

From the analysis of the result a new, more specific, research question emerges: *for what type of systems the use of object diagrams improve the comprehension?*

This will be the subject of further investigations.

7. Acknowledgments

Thanks to Maurizio Morisio and Filippo Lanubile.

8. References

[1] R. Agarwal, G. Bruno, and M. Torchiano, "An Operational Approach to the Design of Workflow Systems" *Information and Software Technology*, 42 (8): 547-555, May 2000.
[2] V. Basili, G. Caldiera, and D. Rombach, "Goal question metric paradigm" in *Encyclopedia of Software Engineering*, vol. 1, J. J. Marciniak, Ed.: John Wiley & Sons, 1994.
[3] L. Bratthall and C. Wohlin, "Is it Possible to Decorate Graphical Software Desing and Architecture Models with Qualitative Information? - An Experiment" *IEEE Transactions on Software Engineering*, 28 (12): 1181-1193, December 2002.
[4] J. Cohen, *Statistical Power Analysis for the Behavioural Sciences*, 2nd ed. Hillsdale, NJ: Lawrence Earlbaum and Associates, 1988.
[5] M. Fowler, *UML Distilled*: Addison Wesley, 2000.
[6] M. C. Otero and J. J. Dolado, "Evaluation of the comprehension of the dynamic modeling in UML" *Information and Software Technology*, 46 (1): 35-53, January 2004.
[7] P. Runeson, "Using Students as Experiment Subjects – An Analysis on Graduate and Freshmen Student Data" Proc. of 7th International Conference on Empirical Assessment & Evaluation in Software Engineering (EASE'03), April 8-10, 2003
[8] M. Shaw and D. Garlan, *Software Architecture, Perspectives on an emerging Discipline*: Prentice-Hall, 1996.