Using an Automatic Collection Method to Identify Patterns during Design Activity

Jonatan Hernandez[1], Hironori Washizaki[2], and Yoshiaki Fukazawa[3]

[1] Waseda University, Tokyo, Japan,
jhernandez@asagi.waseda.jp,
[2] washizaki@waseda.jp,
[3] fukazawa@waseda.jp

Abstract. Although design is an extremely important activity in software development, it is subjective because it depends on the designers' knowledge and skills. Every designer has her or his own strategies to solve design problems. Herein we model the design process as an ordered sequence of logical actions of "Create", "Delete", and "Modify" applied to the elements of a UML class diagram, and propose an automatic approach to collect information about the design process to elucidate design strategies. The strategies considered are top-down, bottom-up, breadth-first, depth-first, and opportunistic. By mining the ordered sequences of actions for frequent patterns and analyzing the position and distribution of the actions in the sequence, we obtained two types of relationships in the design process: micro-patterns and macro-patterns. Then we evaluated our approach with two case studies. The first one, which occurred over a short time frame with seven subjects, identified the strategies used, while the second, which involved three subjects over a long period, revealed that there is not a universal strategy, but a combination of strategies.

1 Introduction:

"All software is designed" [1]. Design is a fundamental step in the software development process, but it is subjective because the abilities of the designers greatly influence the final product. Hence, explicitly identifying successful strategies and patterns will not only increase the understanding of the design process, but will also help improve the skills of software designers.

1.1 Design Strategies

Here the term *strategic knowledge* [2] represents the strategies or approaches applied by the designers, such as top-down or bottom-up. These strategies can be identified using the order in which the elements are created, deleted, and modified. The main elements of a UML class diagram are entities and their relationships. These elements are classified at a different levels of abstraction; classes and relationships are in the higher level of abstraction because they are general,

while methods and attributes are in the lower level of abstraction because they are more specific. The elements and their abstraction levels in this paper are listed below:

- High level of abstraction
  1. **Classes and Interfaces** Entities
  2. **Relationships** Relationships among the entities such as dependency, generalization, and aggregation
- Low level of abstraction
  3. **Attributes and Operations** Entity details

UML specifications contain several types of diagrams (e.g., Activity Diagrams, Sequence Diagrams, Collaboration Diagrams, etc.). Every diagram has its own specific elements (e.g., Actors, Activities, etc.) [3]. Initially we chose the UML class diagram and its main elements as the starting point. The strategies considered are [4], [5]:

Bottom-up Operations and attributes are defined before classes.
Top-down Classes are defined before methods.
Breadth-first Operations are defined before being refined.
Depth-first Operations are defined in the class and immediately refined before creating the next method.
Opportunistic Frequent changes between different levels of abstraction.

It is important to note that there are some constraints when using a UML tool. For example, a bottom-up approach is more difficult because methods or attributes cannot be created without first creating a class.

1.2 Challenges

There are challenges when analyzing design activities. One is data collection. Previous research used verbal protocols to collect data about the design process in which the participants were asked to verbalize their thinking processes [4] [6] [7]. These protocols require an extensive analysis of the recorded sessions, which makes data collection time-consuming. Moreover, most of the elements created during an activity very short lived, and evidence is non-existent or quickly discarded (e.g., notes or talks with colleagues and co-workers) [1]. A method for fast, automated data collection should help establish an explicit record of the creative process.

Another is how to evaluate design. Design is difficult to evaluate in terms of correct or incorrect. As stated in [4], "design problem solutions are more or less 'acceptable' or 'satisfying', they are not either 'correct' or 'incorrect' ". Thus, each design problem has multiple solutions. A method that provides details about the process step-by-step should help realize a better evaluation design evaluation.

2 Proposal: Our Approach

We model the design process as an ordered sequence of actions. Each action is an operation over an element of the UML diagram. The operations are "Create", "Delete", and "Modify", and are applied on entities such as classes and interfaces as well as relationships like generalizations and associations. Figure 1 shows an example of our model of this process.

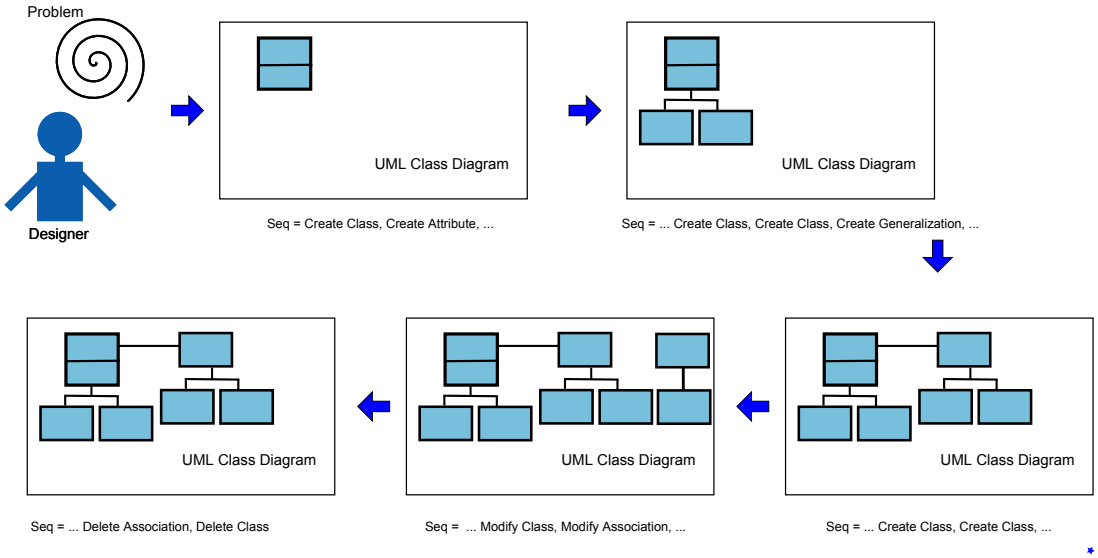

Fig. 1. Representation of the design process as a sequence of actions.

Every diagram creation process is represented as an ordered sequence S of actions a and elements e such as $\{a_1, e_1, a_2, e_2, ..., a_n, e_n\}$, where $a \in \{Create, Delete, Modify\}$ and $e \in \{class, attribute, operation, \ldots\}$. Using this step-by-step process representation of how the design elements are created, we can analyze the importance of elements, the order of creation, and the number of changes to the elements. Two tools were used to collect the sequences of these actions: a collection tool that we developed for ArgoUML [8] by using JAspect [9], and a collection tool provided by Sparx Systems Japan for Enterprise Architect [10]. Two patterns emerged: macro- and micro-patterns. Macro-patterns indicate the relationships between actions and sequences, including the position, distribution, and the total number of actions in a sequence. Examples of macro-patterns include the location of the Create operation, and the ratio between the total numbers of Create actions and Delete actions.

In contrast, micro-patterns represent the relationships between sequence actions, such as the frequent subsequences of actions. To find these relationships, we used the apriori algorithm to search for frequent subsequences and performed

a typical term and pattern-counting search [11]. An example micro-pattern is a frequent subsequence like an action of Create Class followed by Modify Class. Using the apriori algorithm we obtain rules, which are implications of the form $X \Rightarrow Y$ where $X, Y \subseteq a, e$ and $X \cap Y = \emptyset$ [12]. In other words a rule represents how likely an element X will be followed by an element Y in one subsequence. To create the subsequences we divided the main sequence into smaller subsequences of size 2, 3, 4, ..., up to $max.lenght$. These subsequences are repeated at least two times in the main sequence. Thus the $max.length$ is the length in which no more subsequences that repeated at least two times are found.

Two metrics are used to evaluate the rules: support [13] and confidence [14]. Only rules that meet a minimum value for both metrics are considered. Support is defined as the number of sequences in which the subsequence is present. For example, if a subsequence is found in six of seven transactions, its support is 6/7 or 0.86. Confidence is defined as $support(X \cup Y)/support(X)$, and indicates the percentage that a rule is true for all transactions containing both elements X and Y. For example, a support of 0.50 for the rule {Create Class} $\Rightarrow$ {Rename Class} indicates that the rule is true in half of the cases when a transaction contains both Create Class and Rename Class. Finally we show one example a diagram evolution. Figures 2, 3, and 4 show a diagram for minute 9, 30, and 50, respectively. The sequence began with defining many classes, but relationships, attributes, and methods, were added over time.

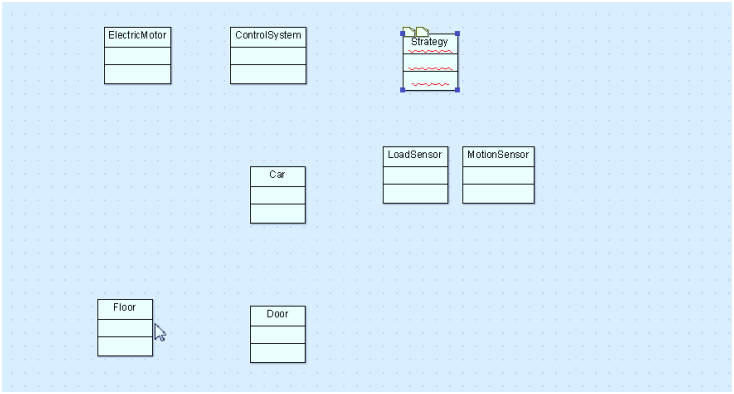

Fig. 2. Minute 9 of E2

3 Case Study 1: Short Time-frame Exercise

This case study involved a small design exercise, which lasted approximately one hour. The subjects (seven total) were asked to use the open source UML tool ArgoUML [8] to analyze the parts and control elements necessary for an

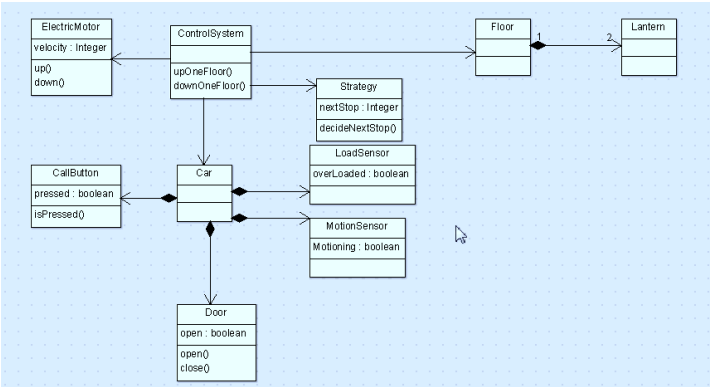

Fig. 3. Minute 30 of E2

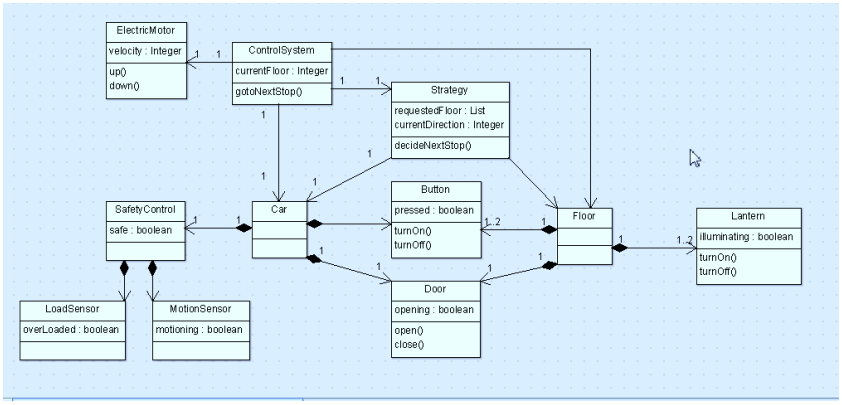

Fig. 4. Minute 50 of E2

elevator to work properly. The objective was to create a UML class diagram with elements and relationships appropriate to model an elevator and its control system. The subjects were volunteers from the Department of Computer Science and Engineering of Waseda University. All the diagrams can be reviewed online at [15].

The elements considered were *class*, *interface*, *attribute*, *operation*, *generalization*, and *association* from the UML standard (Table 1). In this ArgoUML case study, the "Modify" action was renaming of the elements.

The experiments and subjects are referenced as E1...E7 and S1...S7, respectively. Table 2 indicates the academic year of the subjects.

3.1 Macro-patterns

The first macro-pattern was the total number of Delete actions, which were the least frequently performed actions. The percentage of the total is ranged from

	Create	Delete	Modify
Class	CC	DC	MC
Interface	CI	DI	MI
Attribute	CA	DA	MA
Operation	CO	DO	MO
Generalization	CG	DG	MG
Association	CS	DS	MS

Table 1. Abbreviations for the actions collected from the users by our tool in the short time-frame exercise.

Experiment	Subject	Year
E1,E3,E4,E5, E6	S1,S3,S4,S5,S6	Undergraduate students
E2,E7	S2,S7	Graduate students

Table 2. Subjects listed by academic year

4% (E5) to 23% in (E1). Figure 5 shows the distribution for all the experiments. S1, who used UML diagrams for the first time in this case study, had the highest percentage of Delete actions.

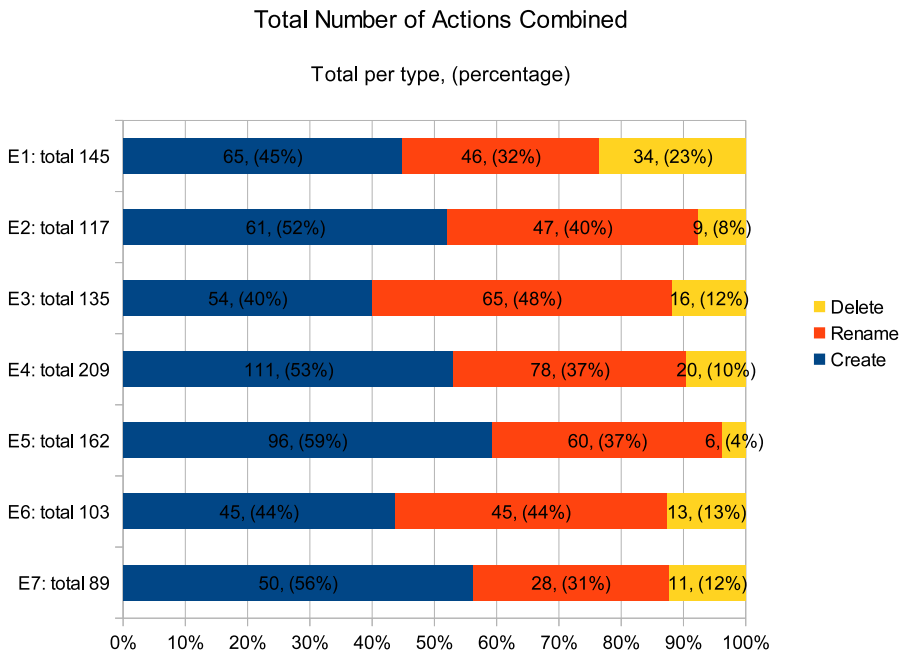

Fig. 5. Percentage of the Total Number of Actions

	Ratios		
	(Create/Modify) Class	(Create/Modify) Attribute	(Create/Modify) Operation
E1	0.62	0.7	0.57
E2	0.93	0.88	1
E3	0.44	0.82	0.94
E4	1.12	0.87	0.96
E5	1.46	1.06	1
E6	0.8	0.78	0.83
E7	0.75	1	2

Table 3. Ration of Create to Modify actions for class, attribute, and operation by subject

The second macro-pattern was the ratio between Create and Modify actions. Action targets were classes, attributes, and operations, which were the most used elements in all the diagrams. Table 3 shows their relationships where a value closer to one denotes fewer changes, while a value less than one indicates that the final name of the classes were altered several times (i.e., the number of modifications is higher than the number of created classes).

Closely related to the previous macro-pattern was the ratio between the total number of classes and the Create Class action (Table 4). Similar values indicated fewer changes in the classes during the exercise, and are associated with fewer errors.

Ratio	E1	E2	E3	E4	E5	E6	E7
Total Classes/CC	0.72	0.85	0.64	0.28	0.4	0.75	0.92

Table 4. Ratio of the total number of classes to Create actions by subject

The third macro-pattern was related to the distribution of actions (Fig. 6). Figure 7 shows the sequences used in E1, E2, E3, and E7. In E2, 9 of the 11 final classes of the diagram were created in the first 10 minutes of the experiment. In E3, all 7 of the final classes were created in the first 10 minutes. In E7, 9 out of the 11 classes were created at the beginning. In contrast, classes in E1 were created and removed throughout the experiment.

In E2 and E7 not only were classes created at the beginning, but they were created sequentially. Classes were created one after another in a long sequence of creation of classes. In E2 nine classes were created in a row, while in E7 seven classes were created in a row. Thus, this design used the same level of abstraction and the same type of elements.

3.2 Micro-patterns

The highest transition rates occurred when executing an action on an element and when continuing with more actions on the same type of element (Table

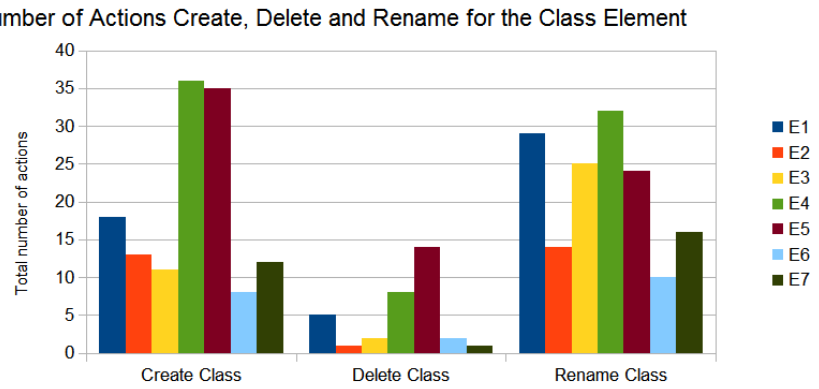

Fig. 6. Number of Create actions

5). The highest rates were for class, attribute and operation, which are the basic elements of a UML class diagram. This micro-pattern of executing actions on elements with the same level of abstraction was related to a breadth-first approach. Two more interesting transitions also resulted from a relationship, such an association or a generalization to a class. Consequently, design occurred at a high level of abstraction.

After a Delete or Modify action, the action with the next highest probability was Create (Table 6), indicating that a modification action created a new element, especially after a Modify action.

Transition		Probability	Transition		Probability
Class	→ Class	0.70	Generalization	→ Class	0.22
Attribute	→ Attribute	0.71	Association	→ Class	0.31
Operation	→ Operation	0.74			
Interface	→ Interface	0.45			
Association	→ Association	0.50			
Generalization	→ Generalization	0.48			

Table 5. Transition rates of elements

There was the second micro-pattern in the transition rates of the actions (Table 6). After the Delete action, there was a 0.39 probability of continuing with a similar action. Similarly, after a Create action, there was about 0.33 probability of repeating the same operation. The most interesting rates were from Create to Modify and vice versa. After creating an element, it seemed natural to modify it, whereas after an element was modified, creating a new element seems logical.

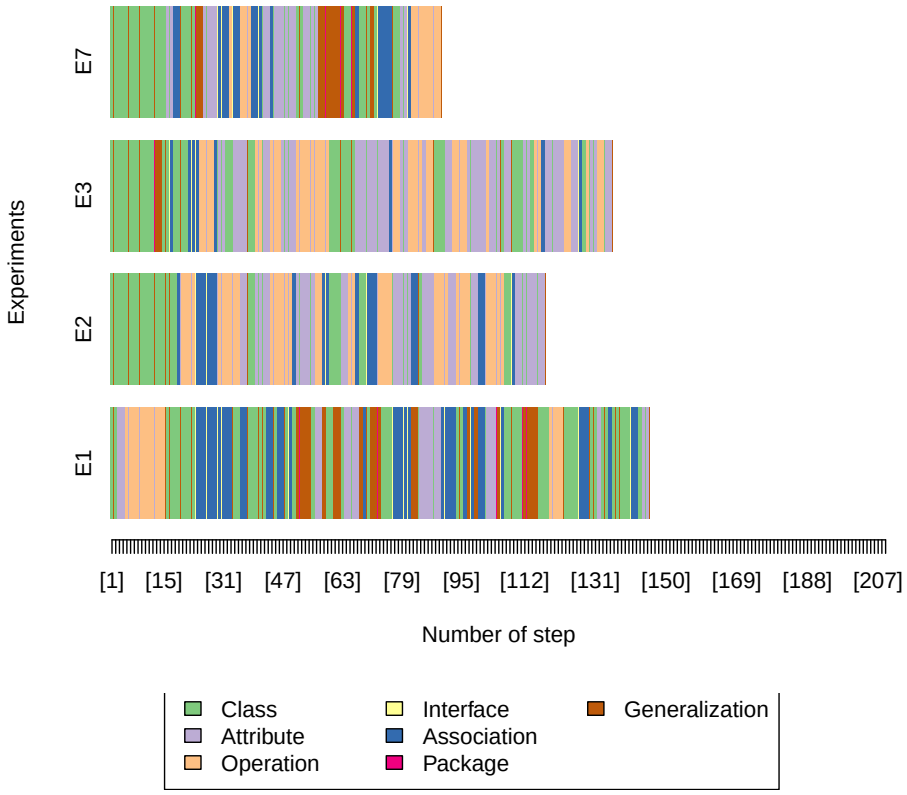

Fig. 7. Comparison of E1, E2, E3, and E7

The rules for Create and Modify actions had a high confidence (Table 7). These rules were related to designing elements at a same, high level of abstraction (i.e., design at the level of classes, which is a high level of abstraction) or designing at the same, more detailed level of attributes and operations.

3.3 Relationship between Strategies and Patterns

In this first case study, several students used a top-down, breadth-first approach. In particular, in the sequences of E2 and E7 (Fig. 7) the subjects were graduate students with more experience in design methods and tools, and began their class diagrams by defining the majority of the classes, and then they refined their design by adding more detail using methods and relationships. This top-down approach, which is demonstrated in Figs. 2 and 3, was commonly used

	→ Create	→ Delete	→ Modify
Create →	0.33	0.07	0.60
Delete →	0.48	0.39	0.13
Modify →	0.72	0.09	0.19

Table 6. Transition rates of actions

Rule			Support	Confidence
Modify Operation	→	Create Operation	0.18	0.77
Create Operation	→	Modify Operation	0.18	0.75
Modify Attribute	→	Create Attribute	0.18	0.73
Create Attribute	→	Modify Attribute	0.18	0.74
Modify Class	→	Create Class	0.30	0.79
Create Class	→	Modify Class	0.30	0.80

Table 7. Rules for the actions and elements

by designers, especially those who had experience in Object Oriented Programming [5]. The top-down approach depended on both designer's experience and difficulty of the problem. The top-down is the most common approach when the problem is known or the designer is experienced [16]. A common micro-pattern was creating and modifying elements on the same level of abstraction (Table 5), which is a breadth-first approach.

A second strategy used the opportunistic approach, where elements and relationships of the diagram were created on different levels of abstraction. E1 in Figure 7 had the highest variation in the abstraction levels.

4 Case Study 2: Long Time-frame Exercise

In this second case study, we collected the logs of students in a Software Engineering class. The students used the UML tool Enterprise Architect [10] for their designs. Data was collected for three months. The objective of the class was for the students to learn the basic principles of Software Engineering by acquiring basic design techniques. Although several domains were presented in the class, herein the domain is a hotel reservation system. Table 8 lists the elements considered in this case study. For Enterprise Architect, the "Modify" action was to rename and to change the properties of an element.

This exercise involved three undergraduate students enrolled in a Software Engineering course and were using UML for the first time. The experiments and subjects are referenced as E8, E9, and E10 and S8, S9, and S10, respectively.

4.1 Macro-patterns

The first macro-pattern was the total number of actions (Create, Delete, Modify) (Fig. 8). The most common action was Modify, which occurred 71.86 % of the time.

	Create	Delete	Modify
Association	CCn	DCn	MCn
Attribute	CA	DA	MA
Method	CM	DM	MM
Class	CC	DC	MO
Object	CO	DO	MO

Table 8. Abbreviations for the Actions collected from the users in the long time-frame exercise.

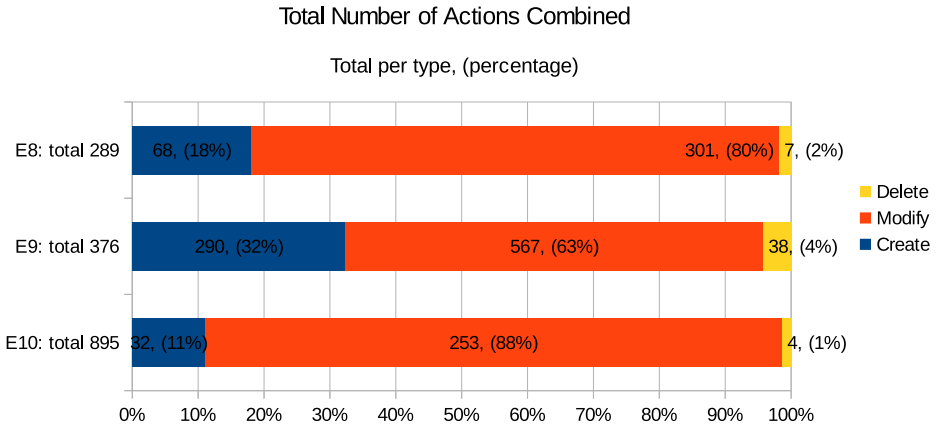

Fig. 8. Total Number of Create, Delete, and Modify Actions

Modifying existing elements was the most common operation, and the element most used was association. A second macro-pattern found was long sequences of Modify actions on the association elements. The longest sequence was 104 modifications of existing associations. Figure 9 shows some very long chains of actions on association elements.

4.2 Micro-patterns

Tables 9 and 10 show the transition rates of the elements and actions. The action with the highest probability was Modify, which was usually executed after any other action. Because examples were given during the class, the students used these examples as the basis for their own designs, which involved more modifications than creation of new elements. The diagrams [15] showed that of all of the students created similar designs.

Table 11 shows the rules for the elements of Enterprise Architect.

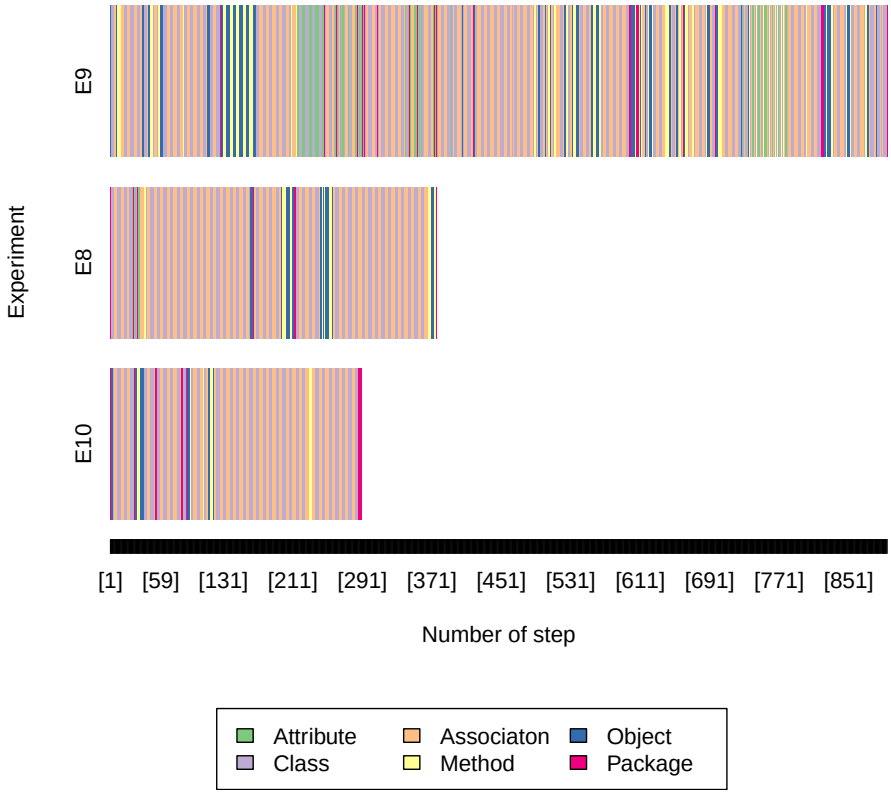

Fig. 9. Sequences for elements

4.3 Relationship between Strategies and Patterns

Although we expected for a top-down, breadth-first approach similar to the first case study, the subjects in the second case study used a combination of strategies. The most common pattern was the modification of the association elements, such as generalization, aggregation, and dependency. 72.88% of all actions in a sequence targeted and association element, and 71.86% of the steps the action was Modify. We believe this result was due to the numerous in-class examples, which were the basis of each subjects' design. The relationships of the elements are high-level abstraction elements. Consequently, the subjects designed at a high level of abstraction instead of using operations or attributes to realize a detailed design.

	→ Create	→ Delete	→ Modify
Create →	0.01	0.02	0.97
Delete →	0.27	0.24	0.49
Modify →	0.33	0.03	0.64

Table 9. Transition rates of actions

Transition	Probability
Attribute→Class	1.0
Method→Class	1.0
Association→Association	0.90
Object→Object	0.75
Class→Class	0.48
Package→Package	0.41

Table 10. Transition rates of elements

5 Comparing Results from the Case Studies

In the first (short time-frame) case study, the top-down, bread-first approach was clearly used, whereas the second (long time-frame) case study, one specific strategy was not employed. The difference seemed to be the length of the experiment. Moreover, the second case study involved students learning about object-oriented design for the first time, which explained why the level of abstraction were frequently changed in a more opportunistic approach.

6 Related Work

In [2], three teams' abilities to solve a design problem were compared by watching videos of the developers, and classifying the statements made during the design to identify the strategies that each team used. Herein our classification is based on the sequence of actions that each developer employed to create a class diagram.

In [7] the design activities of three designers were compared through the verbalization of different tasks. The strategy level of the design activity was analyzed while creating a program at two levels: global-control strategy and very particular strategies, (i.e. "reuse, consideration of the user of the system and simulation"). These strategies are at a different level of abstraction that those used in our study.

In [17] designer's experience is important because it affects his behavior. For example, simulation and note taking only occur when a designer has sufficient domain knowledge. If the designer already has a plan based on his experience, he will use that plan. In our case study, we also observed different strategies in each of the case studies, but we considered time to be the most important factor because the strategies differ in the short time-frame and long time-frame case studies.

In [18] is a study where two teams of two people each designed a traffic simulator, the strategies are divided by two factors: the approach to explore

Rule	Support	Confidence
Create Association→ Modify Association	0.56	0.97

Table 11. Rules for actions and elements

the possible designs: breadth-driven or deep-driven, and the problem solving strategy: problem-driven or solution-driven. The strategies in this study were obtained by observing the discussion of the teams using the recorded session in a video, and classifying the statements during the design session. The approach is different, at a higher level, and with the need of direct observation and analysis of the design process.

In [19] forty professionals participated in a software design task and their strategies were analyzed with a verbal protocol. In this study the differences between high and moderate performers was evident only when analysing what it is called "task-irrelevant verbalizations", which shows the difficulties of analysing design activity with verbal protocols.

In [20] three design sessions, each consisting of a pair of designers working on a design problem were analyzed. The analysis in this paper was made by analysing the sessions, evaluating the discussion and ideas talk by the designers during the sessions. The analysis focused on dividing the sessions into cycles and describing the subjects in each session. This approach requires much analysis that can only be made by a researcher.

7 Conclusions and Future Work

We found macro- and micro-pattern relationships using a quick, non-intrusive method to collect the information about a design process. These patterns were used to identify the strategies employed by designers. The information collected using our process was very specific because it is composed of a set of basic actions that offer a simple but complete view of the design process. We identified two strategies. One was a top-down, breadth-first strategy where classes were initially defined and then the design details were considered. The other was an opportunistic approach where the elements of the diagram were created at different levels of abstraction.

We conducted our experiment on two different groups of students. All the subjects were students, therefore at this moment we cannot generalized our results to other groups of people. In the future we plan to experiment with different groups of subjects, such as professionals.

In the future, we plan to analyze the relationship between metrics and patterns. The strategies used might be related to the quality of the UML diagram, such as quality metrics of the design. We intend to use the elucidated patterns to help the designers inexperience designers to improve their designs.

References

1. A. Baker, A. van der Hoek, H. Ossher, and M. Petre, "Guest editors' introduction: Studying professional software design," *IEEE Software*, vol. 29, no. 1, pp. 28–33, Feb. 2012.
2. V. Popovic and B. Kraal, "Expertise in software design: Novice and expert models," *Proceedings of Studying Professional Software Design*, 2010.
3. P.-A. Muller, *Instant Uml*. Wrox Press Ltd., 1997.
4. W. Visser and J. Hoc, "Expert software design strategies," in *Psychology of Programming*. Academic Press, 1990, pp. 235–249.
5. F. Détienne, "Design strategies and knowledge in object-oriented programming: effects of experience," *Human–Computer Interaction*, vol. 10, no. 2-3, pp. 129–169, 1995.
6. K. Dorst and J. Dijkhuis, "Comparing paradigms for describing design activity," *Design Studies*, vol. 16, no. 2, pp. 261–274, 1995.
7. W. Visser, "Designers' activities examined at three levels: organization, strategies and problem-solving processes," *Knowledge-Based Systems*, vol. 5, no. 1, pp. 92–104, 1992.
8. "Argouml." [Online]. Available: http://argouml.tigris.org/
9. "Aspectj," december 2013. [Online]. Available: http://www.eclipse.org/aspectj/
10. "Enterprise architect," december 2013. [Online]. Available: http://www.sparxsystems.com
11. "Text analysis utilities," june 2013. [Online]. Available: http://cran.r-project.org/web/packages/tau/tau.pdf
12. M. Hahsler and K. Hornik, "Building on the arules infrastructure for analyzing transaction data with r," in *Advances in Data Analysis, Proceedings of the 30th Annual Conference of the Gesellschaft fur Klassifikation e.V., Freie Universitat Berlin, March 810, 2006, Studies in Classification, Data Analysis, and Knowledge Organization*. Springer-Verlag, 2006, pp. 449–456.
13. R. Agrawal and R. Srikant, "Mining sequential patterns," in *Data Engineering, 1995. Proceedings of the Eleventh International Conference on*. IEEE, 1995, pp. 3–14.
14. M. J. Zaki, "Spade: An efficient algorithm for mining frequent sequences," *Machine learning*, vol. 42, no. 1-2, pp. 31–60, 2001.
15. "Diagrams," january 2014. [Online]. Available: http://www.fuka.info.waseda.ac.jp/~jonatan/ref/RD1.html
16. R. S. Rist, "Knowledge creation and retrieval in program design: A comparison of novice and intermediate student programmers," *Human-Computer Interaction*, vol. 6, no. 1, pp. 1–46, 1991.
17. B. Adelson and E. Soloway, "The role of domain experience in software design," *Software Engineering, IEEE Transactions on*, no. 11, pp. 1351–1360, 1985.
18. A. Tang and H. van Vliet, "Design strategy and software design effectiveness," *Software, IEEE*, vol. 29, no. 1, pp. 51–55, 2012.
19. S. Sonnentag, "Expertise in professional software design: A process study." *Journal of Applied Psychology*, vol. 83, no. 5, p. 703, 1998.
20. A. Baker and A. van der Hoek, "Ideas, subjects, and cycles as lenses for understanding the software design process," *Design Studies*, vol. 31, no. 6, pp. 590–613, 2010.