6

Evaluation

In this chapter we look at evaluation methods that have been used to assess various properties of software visualizations. We discuss both quantitative and qualitative methods, with a focus on the latter. We then take a closer look at the question of how one can evaluate the learning effectiveness of an algorithm animation and, finally, discuss some evaluation results that have been published on the learning effectiveness of algorithm visualizations.

6.1 Claims About Visualization Techniques

Section 2.1 presented several general reasons for using visualization techniques from a cognitive-psychological point of view. But this does not imply that every software visualization is per se a helpful tool. For each particular visualization technique, and indeed for each use of it for a particular task, we have to assess its usefulness. To this end, one has to make explicit what "useful" means. The primary goal of visualization is to convey information. It should convey this information in an understandable, effective, easy-to-remember way. These properties of the visualization are often compared to text-based representations of the same information.

Typical problems with evaluations of visualization techniques are the use of toy data sets, and the generation of visual artifacts that suggest nonexistent relations. But, above all, many evaluations are biased because they have been done by the developers of the visualization.

6.2 Quantitative Evaluation

Quantitative methods measure properties of the visualization or of the algorithm, or properties of the human observer interacting with the visualization. Typical questions about a visualization or algorithm are "How much information fits on the screen?" and "How fast does the algorithm compute the

visualization?" Typical questions about the human observer include "How fast does the observer react?", "How much of the information does the observer remember?", and, in particular, in educational settings', "How many tasks were performed correctly?"

Quantitative evaluation requires a statistical analysis of the results of a controlled experiment. An experiment can be described by a set of factors or variables. Factors that are controlled by the person who designs the experiment are called independent, whereas recorded values, such as answers to questions, are called dependent factors. Finally, there are what are called covariate factors, such as age or previous knowledge, which can have some influence on the outcome of the experiment and should be recorded as well.

After the experiment, the resulting data set can be used for descriptive and inferential statistical analyses. Typical statistical measures to describe the central tendency of the data set are its median and mean, and typical measures to describe the dispersion of the data set are its standard deviation and variance.

Inferential statistics tries to assess to what extent observed properties of the data set might have happened by chance. One of the most common methods of inferential statistics is to have two test groups and to compare their performance using a T-test.[1] A T-test determines whether the difference between the means of the two groups is statistically significant. Intuitively, a result is statistically significant if the probability that it is wrong is below a given value, sometimes called the α level. An α level of 5% has become widely accepted.

Some authors argue that in software engineering research it would be better to simply provide the error probability and leave the decision of whether it is "significant" to the reader [PPV00]. On the one hand, for safety-critical applications, an extremely low α level would be important. On the other hand, in situations where no better information is available, even an α level of 40% may still help to make a decision.

6.3 Qualitative Evaluation

Qualitative methods gather data about the individual experience of human observers with the visualization. This experience is verbalized in the form of introspective reports. Ericsson and Simon found that *think-aloud* reports during the task are more reliable than recall protocols [ES80].

When it comes to the human perception of and interaction with a visualization, qualitative evaluation methods are of the outmost importance for various reasons [McC93], including the fact that they require fewer test persons and cover more aspects of the visualization. In contrast, to achieve significant results, quantitative evaluations require that a sufficient number of participants perform a controlled experiment

[1] For more than two groups, a common method is ANOVA (analysis of variance).

Qualitative methods are in particular useful for formative evaluation. Here the goal is to improve a program under development.

6.3.1 Evaluation Based on Gestalt Theory

Sun and Wong have developed a set of 14 criteria for the evaluation of UML layouts [SW05]. On the basis of these criteria, they evaluated two commercial UML tools (Borland Together and Rational Rose). Their criteria were based mostly on gestalt-theoretical principles (see Sect. 2.1.4). These principles can be broadly characterize as guiding either the perceptual organization or the perceptual segregation of objects. The former is related to the question of which objects belong together, whereas the latter is related to the question of how a figure separated from the background.

The gestalt-theoretical principles that are important for perceptual organization are simplicity (the use of a suggestive figure), similarity, proximity, familiarity (meaningfulness), and connectedness. For perceptual segregation, symmetry, orientation, and contour play a crucial role.

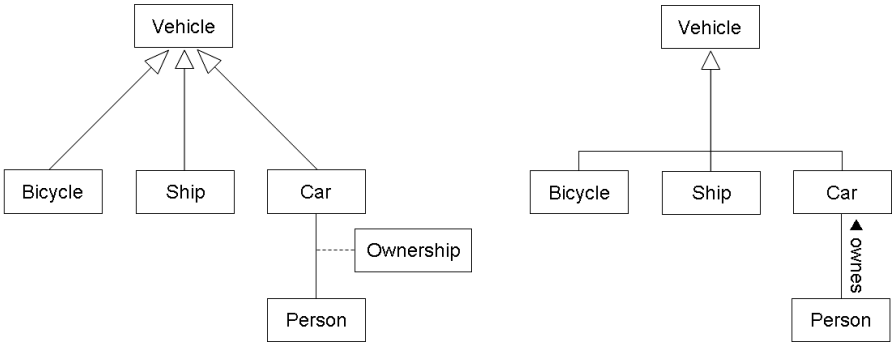

Fig. 6.1. UML diagrams: separate vs. joined inheritance arcs and association classes vs. labeled association links

For the perceptual organization of UML layouts, Sun and Wong suggested, for example, the following three criteria (see Figure 6.1):

- Inheritance arcs should be joined rather than separated (simplicity).
- An association name should be set beside the line, rather than in another association class linked to the line.
- The layout should only show selective information, i.e. information that suits the purpose.

For the perceptual segregation of UML layouts, they suggested, for example, the criterion that overlapping should be avoided, because overlapping will destroy the contours of objects and they will become difficult to recognize (contour and connectedness).

6.3.2 Task-Oriented Evaluation

We all know about task-oriented analysis from computer magazines: we all know task-oriented analysis: software products from different vendors are compared in large tables by listing features and other properties such as prices and licenses, but, more importantly, tasks which are typical of the application. For each product, the table indicates what tasks it supports and possibly even how well it supports each task.

Task-oriented analysis is very popular, because it does not require too much effort, while it actually provides an answer to a key question from a user's point of view: Does the system solve the user's problem?

Pacione et al. performed a task-oriented evaluation of five dynamic visualization tools [PRW03], ranging from visual debugging to UML tools that generate sequence diagrams. For their analysis, they came up with nine large-scale and six small-scale tasks which are typical of software comprehension. The former set of tasks included, for example, finding design patterns that have been applied in the subject system, and or reconstructing the high-level structure of the subject system. The small-scale tasks included finding the collaborations of a small set of objects, and how the state of a specific object changes. Pacione et al. concluded that none of the tools in their study supported all tasks at all levels and that combining static and dynamic information might lead to better results (see also Sect. 4.3).

The cognitive walkthrough is a task-oriented technique for evaluating the usability of a system. The technique is named in analogy to code walkthrough in software engineering. When performing a cognitive walkthrough one keeps track of the user's thought processes, decision making, and memory load at each step when the task is performed [WRLP94, PLRW92].

6.3.3 The Cognitive-Dimensions Framework

The cognitive-dimensions framework [GP96, BG03] is a qualitative and often formative evaluation method to assess (interactive) tools for storing, manipulating, and displaying information, or, as the authors of the framework put it, they are "a set of discussion tools for use by designers and people evaluating designs". Six types of generic tasks are distinguished:

Incrementation is the task of adding information to a system or document.
Transcription is the task of transforming information from one representation to another.
Modification refers to the changing of information or its representation.
Exploratory design is the task of sketching new representations without knowing the solution in advance.
Searching is the task of looking for a known piece of information.
Exploratory understanding is the task of discovering the overall structure of the information.

Table 6.1. Cognitive-dimensions profiles of four generic tasks

	Viscosity	Hidden dependencies	Premature commitment	Visibility
Exploratory design	Harmful	Acceptable	Harmful	Important
Modification	Harmful	Harmful	Harmful	Important
Incrementation	Acceptable	Acceptable	Harmful	Useful, nonvital
Transcription	Acceptable	Acceptable	Harmful	Useful, nonvital

Evaluation using the framework works as follows. First, one has to identify what generic tasks the user will perform with the system. Then, for each task, one has to assess how it fits each cognitive dimension. This leads to an observed profile for each task, which can be compared with an ideal profile listed in the cognitive dimensions tutorial [BG]. Table 6.1 shows the profiles of four generic tasks based on four of the 14 cognitive dimensions listed in Table 6.2.

Table 6.2. List of cognitive dimensions

Cognitive dimension	Description
Viscosity	Resistance to change
Visibility	Ability to view components easily
Premature commitment	Constraints on the order of doing things
Hidden dependencies	Important links between entities are not visible
Role expressiveness	The purpose of the entity is readily inferred
Error-proneness	The notation invites mistakes and the system gives little protection
Abstraction	Types and availability of abstraction mechanisms
Secondary notation	Extra information in means of other than formal syntax
Closeness of mapping	Closeness of representation to domain
Consistency	Similar semantics are expressed in similar syntactic forms
Diffuseness	Verbosity of language
Hard mental operations	High demand on cognitive resources
Provisionality	Degree of commitment to actions or marks
Progressive evaluation	Work-to-date can be checked at any time

As an example, consider the Creole plug-in that analyzes a Java program and produces various kinds of diagrams such as those shown in Sect. 3.4.4. With respect to most of the cognitive dimensions, Creole is well designed. Considering visibility, for example, boxes can be expanded by simply clicking on a box to show its components. On the other hand, the expressiveness of the graphical representation is very restricted: classes, packages, methods, and attributes are all represented by boxes and the semantics of a box can be inferred from the context. Furthermore, the viscosity of Creole is high.

When the source code changes, for example when a class is renamed, the class hierarchy has to be recomputed, and a new layout of the diagram is computed that can be quite different from the previous one.

For each of the dimensions, the cognitive dimensions tutorial [BG] provides a definition; one or more thumbnail examples; a more detailed explanation; the cognitive relevance of the dimension, in particular with respect to the generic tasks; cost implications; more detailed examples; workarounds; remedies; and trade-offs.

6.4 Educational Evaluation

Learning scenarios such as those described in Sect. 4.4.8 have been developed with the goal of facilitating understanding and learning of an algorithm. But the question arises of whether they really achieve this goal. In other words, the learning effectiveness has to be evaluated. To this end, user studies have to be performed.

We shall illustrate the various aspects of the design of a study by examples taken from an evaluation that we did several years ago for an interactive animation of a generation algorithm for finite automata which we briefly called GANIFA [GAN].

The screen dump in Fig. 6.2 shows how the generation process of a lexical analyzer is visualized by GANIFA. This example shows the conversion of a regular expression $(a|b)^*$ into an appropriate nondeterministic finite-state automaton ($RE \rightarrow NFA$). The generator has been integrated into an applet for visualizing the generation and computation of finite automata, which has been used in an electronic textbook on the theory of finite automata [Gan00].

Design of a Study

For a study of learning effectiveness, two key decisions have to be made: what kind of experiment will be performed, and who will be the test persons?

Test Persons Obviously, the test persons should belong to the target group of the learning scenario, for example, one can hardly evaluate a textbook for children with adult test persons. On the other hand, a sufficient and, in particular, representative number of members of the target group are often not available as test persons. Therefore many studies on algorithm animation have been done with psychology or computer science students and as a result are biased to some extent. After the test persons have been chosen, one has to split them into two ore more groups. These groups then use alternative means to learn about an algorithm. The simplest approach to forming these groups is to assign test persons randomly to groups. In the evaluation of GANIFA we had about 118 first-year computer science students, split randomly into four groups. Later it turned out that in some groups there were considerably more students with very good high-school grades in mathematics.

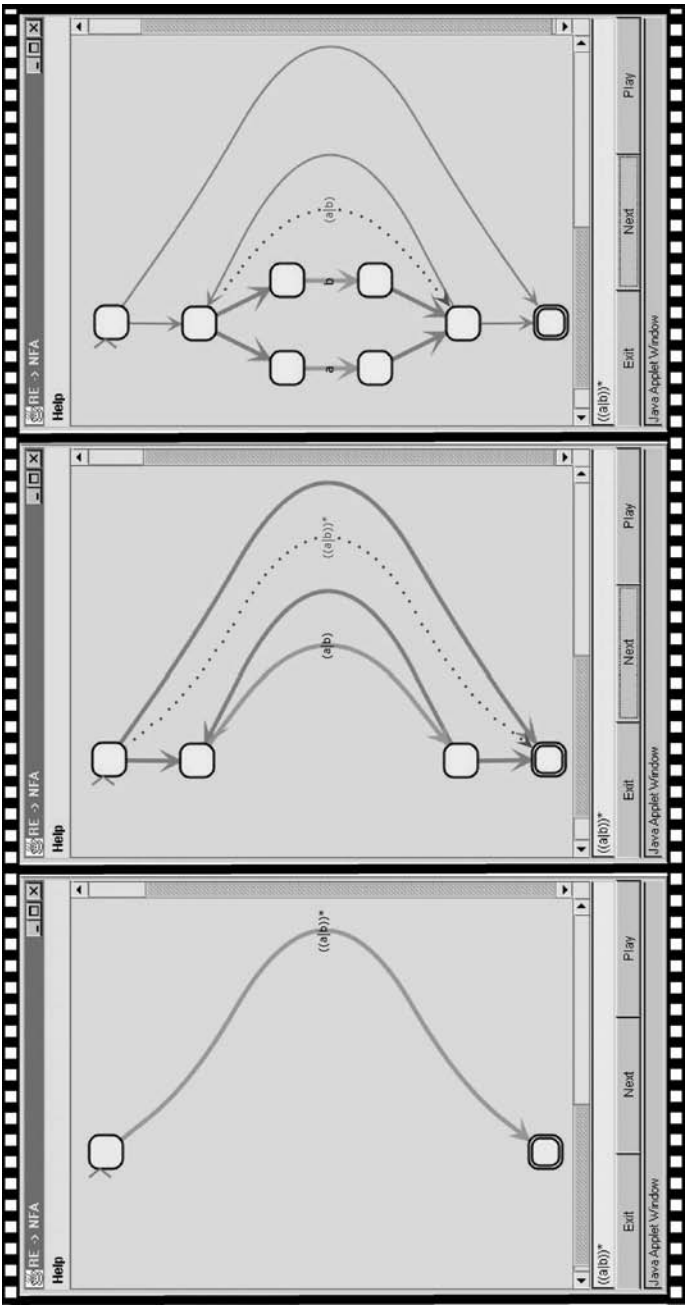

Fig. 6.2. GANIFA: animation of the generation of a finite automaton

If one has only a small number of test persons, one can try to prevent such unexpected effects by performing a pre-test first and taking the results of the test into account when forming the groups: for example, one may use the same numbers of people with good and bad pre-test results in each group.

Factors Now one can perform the actual experiment. In general terms, the goal of such an experiment is to find a relation between independent variables, i.e. the controlled factors, and dependent variables, i.e. the measured factors. To this end, the experiment is performed with several different values of the independent variables for each test group, and the changes in the dependent variables are recorded. Assuming n independent variables with m different values for each of these variables, one would need $n * m$ test groups with a sufficient number of persons per group to allow statistically significant results. So, as a matter of fact, most studies have concentrated on one or two independent variables. For example, learning efficiency is typically investigated by comparing different learning scenarios, i.e. the learning scenario is the independent variable.

In the GANIFA evaluation [Ker04, DK01], the independent variable investigated was the learning medium. We chose four media: a learning-software package with fixed animations, a learning-software package with generated animations depending on the students' inputs, a textbook, and a conventional lecture.

Measurement As we are interested in learning efficiency, we need appropriate measures. One approach is to measure performance during the experiment, i.e. to capture the students' behavior with traces or video recording. Instead or in addition, one can do post-tests, i.e. after the experiments the students have to answer a set of questions. With these questions, one tries to measure the students' conceptual or declarative knowledge, for example their understanding of the abstract properties of an algorithm, and their procedural knowledge or skills, for example their understanding of the procedural, step-by-step behavior of an algorithm. With a pre-test, one can see what the students knew beforehand, and in combination with the post-test results, one can measure the improvement or increase of knowledge.

For the GANIFA evaluation, we performed both pre- and post-tests to see whether the students' knowledge and skills had improved. In these tests, we had knowledge questions about conceptual and procedural understanding, transfer questions asking the students to transfer/apply knowledge in a different context, and open questions, where students could make comments and suggestions. Open questions are very important for identifying factors and problems that one was not aware of when designing the study. Below are some example questions from the GANIFA evaluation:

Pre-test

Do you know what finite automata are?
Which words belong to the language defined by the regular expression $(\mathtt{ab})^*$?

Post-test

Knowledge question
Which words belong to the language defined by the regular expression $\mathtt{ab}^*\mathtt{a}$?

Transfer question
We add the notation r^+ to our regular-expression language, where r is a regular expression, such that $L(r^+) = \{w^n | w \in L(r), n \geq 1\}$. For example, $L(\mathtt{a}^+) = \{\mathtt{a}, \mathtt{aa}, \mathtt{aaa}, \dots\}$. Give a construction rule for a transition diagram of an NFA.

Open question:
What properties of the animation helped you to better understand the generation algorithm?

Learning Theories While it might be sufficient for a designer of an algorithm animation to see that it helps learning, cognitive scientists have the more ambitious goal of trying to explain why. To this end, they try to apply existing learning theories or create new ones which address issues such as the correspondence of the graphical metaphors and the mental model, the use of both hemispheres of the brain, and the active construction of subjective knowledge by the test person.

6.5 Some Interesting Empirical Results

In this section, we discuss several empirical studies related to various fields of software visualization.

Algorithm animation In a metastudy performed by Hundhausen et al., more than 40% of the 24 studies considered did not find significant results [HDS02]. Several studies found that electronic learning material (multimedia or hypermedia) with algorithm animations outperformed lectures. The comparisons with textbooks are less clear. But the most important result of the metastudy was that the form of the learning exercise is actually more important than the quality of the visualizations used:

> Thus, according to our analysis, how students use AV [algorithm visualization] technology, rather than what students see, appears to have the greatest impact on educational effectiveness.

In particular, algorithm visualizations were efficient in scenarios were students had to actively solve prediction and programming exercises.

Static Program Visualization and Visual Programming There have also been many empirical studies about the use of diagrams for programming. For LabView, a widely used visual programming language, Green and Petre [GP92] found, in a quantitative study, that in most cases the diagrams were more difficult to understand than a textual representation, for both occasional and experienced users. Similar experimental results had been already reported about flowcharts in 1977 [SMMH77], but about ten years later, in a replicated experiment [Sca89] using time as an additional dependent variable, it turned out that flowcharts outperformed source code for algorithm comprehension.

Visual debugging Jones et al. evaluated the effectiveness of the Tarantula tool [JHS02, Tar] not with test persons, but by applying the system to different versions of the same program with different faults and assessing how many of the faulty and nonfaulty statements were colored reddish. In their experiments, they applied 1000 test cases to 60 different versions of a program with 9500 lines of code. Of these, 20 versions contained a single (known) fault, 10 versions two faults, 10 versions three faults, 10 versions four faults, and 10 version five faults. They found that fewer than 20% of the nonfaulty statements were colored in the reddest 20% of colors used. While the majority of faulty statements were colored in a reddish color, they also found that with an increasing number of faults, the number of reddish-colored faulty lines decreased.

Class Diagrams Irani and Ware have performed several experiments to compare UML diagrams and geon diagrams (see Sect. 3.4.2 for more details on geon diagrams). In one experiment [IW00], they first showed a UML or geon diagram for 15 seconds to the test persons, and then they presented them with diagrams of substructures and asked whether these occurred in the original diagram. All diagrams were shown on the computer screen and the test persons had just to press the "Y" or "N" key. In this experiment the average identification time was 4.3 seconds for geon diagrams, but 7.1 seconds for UML diagrams. But, even more importantly, the error rate was only 13% for geon diagrams, compared with 26% for UML diagrams. Thus the identification of substructures was much faster and more accurate with geon diagrams.

In another experiment [IW04], the test persons were given a problem description and four UML or geon diagrams. The task was to decide which one of the diagrams was created for that problem. In this experiment the average error rate was 15% for geon diagrams and 36% for UML diagrams.

Graph Drawing For more than a decade, Helen Purchase has performed empirical studies related to the aesthetics of graph layouts. Whereas in earlier work she found that reducing the number of edge crossings was the most important aesthetic consideration [PCJ96, Pur97], in recent work continuity also turned out to be an important factor [WPCM02]. Here, continuity means the

sum of the angular deviations of the incoming and outgoing edges for each node on a path. If the edges on a path form a straight line, the path has a continuity of 0 degrees. In Fig. 6.3 the path from node A to C has a continuity of 0 degrees, whereas the path from A to E has a continuity of 180 degrees.

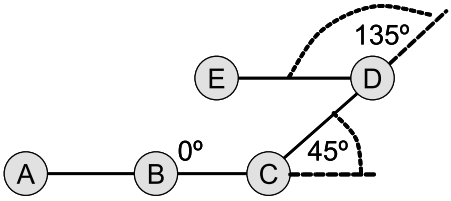

Fig. 6.3. Angular deviations along a path

In one of her experiments, test persons were asked to identify the shortest path between two highlighted nodes. For each graph shown, the values of several variables were recorded: the response time *rt*, the shortest path length *spl* in terms of the number of edges, the continuity *con* of the shortest path in degrees, the number of edges that cross the shortest path *cr*, and the number of branches *br* from the intermediate nodes on the shortest path. Some other properties, such as the total number of edge crossings, that turned out to be not significant using an α level of 5% were also recorded. By stepwise multiple-regression analysis the following relation was obtained between these variables with a regression correlation coefficient $R^2 = 0.784$:

$$rt = 0.414\, spl + 0.406\, con + 0.317\, cr + 0.172\, br \tag{6.1}$$

This indicates that the response time increases with the length of the shortest path, but also with its continuity and the number of crossings on this path.

6.6 Summary

There have been quite a lot of studies related to algorithm visualization systems in education. This is unfortunately not the case for other areas of software visualization. The lack of empirical studies is a shortcoming no only of software visualization research, but also of software engineering and computer science in general [Tic98]. In a survey of 400 research articles [TLPH95], Tichy et al. found that more than half of the software engineering papers were not experimentally validated.

As quantitative evaluations that involve human test persons are very time-consuming, at least qualitative evaluations should be performed during the design of visualization tools or posthoc.

Only a few software visualization systems exist that support the development of large software systems with the large, distributed teams of programmers that are common in the software industry today. The evaluation of industrial software visualization is full of open questions for empirical studies. To what extent has software visualization been applied effectively in industry? Does it keep its promises of increased productivity and decreased development and maintenance costs?

Exercises

Exercise 1: Design a controlled experiment to evaluate whether a tool such as those discussed in Sect. 3.4.4 can help a user to understand the source code of a given software system better than by reading only the textual representation of the source code itself. What are the independent, dependent, and covariate variables in your experiment?

Exercise 2: In Exercise 2 of Chap. 4 you had to implement a visualization of an algorithm for the scheduling of independent tasks. Now imagine that you had to evaluate the learning effectiveness of your visualization. To this end, you would have to design a pre- and a post-test. Suggest at least a pre-test question, a knowledge question, a transfer question, and an open question for such a questionnaire.

Exercise 3: Use the cognitive-dimensions framework to evaluate a software visualization system of your choice, preferably one that you have access to. For example, you could evaluate a tool that analyzes a Java program and produces a UML class diagram such as those discussed in Sect. 3.4.4. Do all dimensions apply to the system that you have chosen?