

Special Issue

How design notations affect the comprehension of Web applications



Filippo Ricca¹, Massimiliano Di Penta^{2,*},[†], Marco Torchiano³,
Paolo Tonella⁴ and Mariano Ceccato⁴

¹*Unita' CINI at DISI (Laboratorio Iniziativa Software FINMECCANICA/ELSAG S.p.A. - CINI), Genova, Italy*

²*RCOST - University of Sannio, Benevento, Italy*

³*Politecnico di Torino, Italy*

⁴*FBK-irst - Fondazione Bruno Kessler, Trento, Italy*

SUMMARY

Web application design requires the modeling of multiple, separate concerns, such as the navigational structure, the business logic and the data persistence. To this aim, several methodologies have been conceived. One of them, the Web Application Extension (WAE), extends the UML notation by means of stereotypes and tagged values intended to capture Web-specific concepts (e.g., the navigational structure). Although the WAE methodology is nowadays quite mature and ready for industrial adoption, the question whether it is able to actually facilitate the task of developers and maintainers has still to be empirically investigated. This paper reports and discusses the results from a controlled experiment on the benefits associated with the use of the WAE notation in the execution of comprehension tasks, carried out before maintenance. The WAE notation was compared against the use of pure unified modified language. Results indicate that the use of the WAE notation significantly improves the level of comprehension, although it does not increase the time needed to perform the comprehension task in a significant way. Copyright © 2007 John Wiley & Sons, Ltd.

Received 9 February 2007; Revised 17 July 2007; Accepted 31 July 2007

KEY WORDS: empirical studies; program comprehension; Web applications; design notations

1. INTRODUCTION

Web applications are complex software systems, involving several concerns (persistent data, business logic, navigation structure, user interface, security, session management, authentication, etc.)

*Correspondence to: Massimiliano Di Penta, RCOST - University of Sannio, Benevento, Italy.

[†]E-mail: dipenta@unisannio.it



and technologies (server-side scripts and languages, HTML, XML, client scripts, communication protocols, etc.). Hence, their design, implementation and evolution remains a challenging task.

Several approaches and notations have been proposed to facilitate the high-level representation of Web applications and to support their successive implementation, for example the Web Modelling Language—WebML [1], the UML-based Web Engineering—UWE [2], the Web Site Design Method—WSDM [3], the Object-oriented Hypermedia Design Model—OOHDM [4], the Web Application Extension WAE—[5] and the Ubiquitous Web Applications—UWA [6]. Many of these notations are extensions of the Unified Modeling Language—UML [7]. However, developing design models and keeping them consistent with the implementation is a costly and time-consuming activity. Often, the pressure for the delivery of initial or new versions of the applications forces developers to deal immediately with the implementation issues and details. As a result, models are often unavailable or inconsistent with the implementation. An option is reverse engineering them from the source code, but again this does not come at no cost. On the other hand, the availability of high-level models may simplify the understanding and evolution of such applications, hence justifying the costs involved in model recovery or construction from scratch. Bearing this in mind, it is necessary to investigate what are the benefits introduced by the availability of Web-specific design models, provided that such availability would require additional effort and expertise.

In this paper, we investigate how the availability of Web-specific design models influences the comprehension of Web applications in the context of a maintenance task. We designed and conducted a controlled experiment for the assessment of the effectiveness of a specific design notation which extends UML-Conallen's stereotypes, also known as WAE, Web Application Extension [5]. We considered the execution of maintenance tasks involving program understanding and modification, and compared the performance of developers using Conallen's diagrams with those using basic (standard) UML. The obtained results support the hypothesis that design models contribute significantly to simplify the comprehension of Web applications. However, such a support does not have a direct effect in terms of time needed to perform the comprehension task. When models are available, users spend more time on them and less time on the source code, so that the overall time required is not significantly different from that for the control groups. However, subjects using Web-specific models acquire a deeper knowledge about the system than those who use generic (non-stereotyped) UML diagrams only.

The paper is organized as follows: Section 2 provides a primer on how to model Web applications with basic UML and with WAE. Section 3 describes the design of the empirical study we conducted. Section 4 presents and discusses the results. Related works are commented in Section 5. Finally, Section 6 concludes the paper and outlines directions for future work.

2. WEB DESIGN NOTATIONS: UML and WAE

During maintenance tasks, models help designers and programmers in understanding the system by abstracting away some of the details and by identifying the artifacts interested in the maintenance. Web applications, like traditional software systems, are often represented by a set of models: use case models, design models, implementation models, deployment models, security models, etc. We focus here on design models that, in the case of Web applications, are



used to describe concerns proper of this kind of application, i.e., the navigational structure, the session-handling mechanism, etc. Also, during comprehension and maintenance activities, design models are used to identify the portion of a Web application that is interested by a change request. This section recalls how a Web application design can be realized using a basic UML notation, in particular class diagrams. Then, it describes how a UML extension for modeling Web applications, i.e., WAE, can be used to better capture concerns peculiar to Web applications.

UML [8] is the *de facto* standard modeling language for software-intensive systems. It can also be used to represent the design of a Web application. In a standard UML class diagram (i.e., one with no Web-specific stereotypes), Web pages are mapped to classes, while relationships between pages are mapped by means of the *use* relationship. Assuming a Web page is a class, in a standard UML class diagram, page scripts and variables are mapped to class operations and attributes, respectively. A problem arises when we consider that a Web page may contain a set of scripts executed on the server (e.g., CGI, Java Server Pages—JSP or Servlets) and a completely different set of scripts executed on the client (e.g., Javascript). When we look at a Web page class in a model, there is confusion on which operations are executed on the server and which are executed on the client. A simple mapping of Web pages to UML classes does not help designers and programmers in the comprehension of the system. Moreover, basic class diagrams do not permit the representation of hyperlinks between HTML pages, nor of how variables are submitted through a form to a server-side application.

The creators of UML realized that the language could not be sufficient to capture all the relevant aspects of a particular domain or architecture. For this reason, an extension mechanism was defined to allow designers extend the semantics of UML. In particular, the extension mechanism allows people to define *stereotypes* (adornments or icons having a well-defined semantics), *tagged values* (value pairs used to assign a value to an element of the model) and *constraints* (rules that define the well formedness of the model) that can be applied to the elements of the model. Jim Conallen, starting from similar considerations, came to the conclusion that UML should be extended to avoid ambiguity and to better represent the relationships among Web pages. WAE, the UML extension proposed by Conallen [5], is a notation for the design, construction and maintenance of Web applications. It models pages, hyperlinks and dynamic content on the client and server by using the UML's extension mechanisms. With respect to the basic UML, the most important novelty is that the server-side aspects of a Web page are modeled with one class, stereotyped as `<<Server Page>>`, and the client-side aspect with another class, stereotyped as `<<Client Page>>`. The two classes are related to each other with a directional relationship. Since a server page builds a client page (an HTML document that contains both content and presentation), this association is stereotyped as `<<Build>>`. Using these stereotypes makes it easier to model pages' scripts and relationships. The `<<Server Page>>` class operations are functions in the server-side scripts of a page, while the `<<Client Page>>` class operations are functions visible on the client side. By separating the server-side and client-side aspects of a page into different classes, it is possible to highlight the relationships between pages and other classes of the system [9]. Server pages are modeled with relationships to server-side resources—e.g., databases, Java components, etc.—while client pages are modeled with relationships to client-side resources—e.g., Java Applets, ActiveX, Javascript functions, etc. Further details about the WAE methodology and the related notation can be found in the book by Conallen [5].



3. EXPERIMENT DEFINITION, DESIGN AND SETTINGS

This section provides the experiment definition and context, and describes the experimental design and procedure following the guidelines by Wohlin *et al.* [10] and by Juristo and Moreno [11] on how to document and report empirical studies in software engineering.

3.1. Experiment definition

The *goal* of the study is to analyze the use of stereotyped UML diagrams (following the notation defined by Conallen in the WAE methodology [5]), with the *purpose* of evaluating their usefulness in Web application comprehension. The *context* is represented by open-source Java Web applications (the study objects), and by a classroom of software engineering undergraduate students (the subjects). The *quality focus* is to improve the maintainers' comprehension level and the time needed to perform the comprehension necessary for a maintenance task. The *perspective* is both of *Researchers*, evaluating how effective are the stereotyped class diagrams for supporting Web application maintenance, and of *Project managers*, evaluating the possibility of adopting the WAE methodology or at least its notation in the development of Web applications.

3.2. Context

Two Web applications were selected as objects for this study: *Claros*[‡] and *WfMS*[§] [12]. Both are small/medium size, freely available applications (see Table I) based on the Servlet/JSP technology. *Claros* is an on-line Web mail management application. As many other existing Web mail systems, it provides features such as composing and sending a message, downloading messages from a mail box, browsing, answering, forwarding and deleting messages. *WfMS* is a simple workflow management system that permits the definition of processes and their enactment. While *WfMS* is larger in terms of classes, *Claros* has a larger View (and thus a more complex navigational model). The View comprises 19 UML classes (38 in the Conallen's diagram) for *Claros* and 13 (24 in

Table I. Characteristics of the systems under study.

	Files	LOC
<i>Claros</i>		
Java	44	6288
JSP	34	1996
Total	78	8284
<i>WfMS</i>		
Java	85	2378
JSP	7	431
Total	92	2809

[‡]<http://www.claros.org>.

[§]<http://www.pearsoned.co.uk/HigherEducation/Booksby/BrugaliTorchiano/>.



Conallen's diagram) for WfMS. UML and Conallen diagrams of both *Claros* and *WfMS* views can be found in a longer technical report [13].

Subjects who participated in the study were Bachelor students attending the Laboratory of Software Engineering course (second year BSc) at the University of Trento, Italy. Thirty-five students were registered for this course. All of them took part in the experiment, in other words a convenience sampling was used for subjects' selection. Subjects had previously attended Programming and Software Engineering courses. They had sufficient knowledge of UML and Java to understand the models and source code of our experimental objects. According to their grades in previous courses, subjects were classified into *high-* and *low-*ability subjects. Such a classification is useful to block the effect of subjects' ability on results.

3.3. Hypotheses

We are interested in analyzing how stereotypes affect comprehension and maintainers' productivity during the comprehension task. Thus, we formulate two null hypotheses:

H_{c0} When doing a comprehension task the use of stereotyped class diagrams (vs non-stereotyped class diagrams) *does not significantly affect* the comprehension level.

H_{t0} When doing a comprehension task the use of stereotyped class diagrams (vs non-stereotyped class diagrams) *does not significantly affect* the maintainers' productivity in the comprehension task.

When each one of the null hypotheses is rejected with relatively high confidence (we set $\alpha=5\%$ in our case), it is possible to formulate alternative hypotheses:

H_{ca} When doing a comprehension task the use of stereotyped class diagrams (versus non-stereotyped class diagrams) *significantly affects* the comprehension level.

H_{ta} When doing a comprehension task the use of stereotyped class diagrams (versus non-stereotyped class diagrams) *significantly affects* the maintainers' productivity in the comprehension task.

3.4. Experiment main factor

In this experiment we use only one main factor, herein referred as *Method*, concerning the use of different notations for producing the class diagrams for a Web application. The treatment considered in this experiment is WAE, the design notation proposed by Conallen [5], herein referred just as '*Conallen*'. Since this notation extends UML through a set of stereotypes, the notation used for comparison (second treatment) is basic UML, with no Web-specific stereotype. The aim is to determine whether, during a maintenance task, stereotyped diagrams help in improving the comprehension level and maintainers' productivity in the comprehension task. In both cases subjects will also have access to the Web application source code. This is crucial for two main reasons:

1. In a realistic environment, maintainers start processing a change request by comprehending the (sub)system to be maintained through the analysis of high-level artifacts, but also of the source code.
2. To make the comparison fair, it is necessary to provide the same level of information to subjects working with stereotyped diagrams and with basic UML diagrams; in the second case (UML) maintainers would be able to find some pieces of information—e.g., that related



to the navigational structure—in the source code only, while in the first case (Conallen) the same information may appear in the stereotyped class diagrams.

3.5. Experimental procedure and design

Before performing the experiment, subjects were trained on Conallen's notation, as well as on all technologies used in the target applications (e.g., Servlets/JSP). During the laboratory session preceding the experiment, they had to practice with Servlet/JSP technology, by developing a simple Web application. To practice with Conallen's notation, they had to analyze the design of a simple Web application related to an electronic shop system. Finally, right before the experiment a briefing session aimed at providing them with detailed instructions about the tasks to be performed during the laboratory sessions. Subjects were asked to perform their tasks individually, and this was enforced by the presence of professors and teaching assistants in the laboratory. The experimentation consisted of two laboratory sessions (*Lab 1* and *Lab 2*), held during two subsequent weeks, and each lasting approximately up to two hours, although subjects were asked to use only the time needed to perform the requested task, and not necessarily the whole laboratory tasks.

The experimental design followed a counter-balanced scheme, in which each subject had to perform the experimental task with different treatments of the main factor (*Method*), i.e., Conallen and UML, on two different systems (Claros and WfMS). The design is the same as used in other comprehension-related experiments [14,15]. As shown in Table II, subjects were assigned to four groups composed of 6–8 subjects, each receiving the same treatment in each laboratory. The assignment was random; however, to permit blocking, it was ensured that each group contained roughly the same number of high- and low-ability subjects.

Each laboratory consisted of a comprehension task on the assigned object (Claros or WfMS) documented either by Conallen or by pure UML diagrams. The comprehension task was carried out by answering 12 open questions (the same number of questions as in [16]) on the assigned system. The questions, different for the two Web applications, concerned comprehension or impact analysis issues related to realistic maintenance scenarios for Web applications. To formulate them, we referred to some change requirements of real Web applications published on SourceForge[‡], a large repository of open-source projects. A sample of the questions for the WfMS application is shown in Table III. To answer the questions, subjects were provided with the following material:

- a brief, textual description of the software system purpose and of its main features;
- the system source code;
- (Conallen or pure UML) design diagrams;

Table II. Experimental design.

	Group 1	Group 2	Group 3	Group 4
Lab 1	Claros–Conallen	Claros-UML	WfMS–Conallen	WfMS-UML
Lab 2	WfMS-UML	WfMS–Conallen	Claros-UML	Claros–Conallen

[‡]<http://sourceforge.net/>.



Table III. Sample questions (five out of 12) for WfMS.

ID	Question
1	Suppose that you have to set the background color of each Web page using CSS (Cascading Style Sheets). Which classes/pages does this change impact?
2	Suppose that you have to substitute, in the entire application, the form-based communication mechanism between pages with another mechanism (i.e., Applet, ActiveX, etc.). Which classes/pages does this change impact?
3	Does the application conform to the Model-View-Controller (MVC) pattern? If yes, which class (or classes) implements the controller component?
4	The description of a process is made up of three main types of elements (activity, participant and transition) and stored in an XPDL file. Which are the process modeling classes (i.e., the classes used to represent the processes in memory)?
5	Which classes are initialized when the JSP container starts and are destroyed when it shuts down? These classes keep the long-lived information and are used by almost all Web pages.

- the installed system, so that they could interact with it;
- the list of questions to be answered;
- a form to provide the answers and to specify the time when they started the task and when they completed it; and
- a copy of the laboratory instructions.

The whole experimental package is available for replication purposes^{||}. Additionally, at the end of each laboratory session, we asked subjects to fill in a survey questionnaire, aimed at gaining insights about their behavior during the experiment and at better explaining the quantitative results. The questionnaire concerns the task and system complexity, the adequacy of the time allowed to complete the task and the usefulness of the provided diagrams. As shown in Table IV, it consists of seven common questions plus two questions (Q8 and Q9) to be answered by subjects only using Conallen's diagrams. Answers to Q1–Q5 and to Q8, Q9 are expressed on a Likert scale [17] from 1 (strongly agree) to 5 (strongly disagree). Answers to Q6, Q7 are based on a 5-point ordinal scale: {A, B, C, D, E}.

3.6. Variable selection

As described in Section 3.4, this experiment has only one factor, the *Method*, which indicates the notation used to describe the Web application (*UML* or *Conallen*). Besides the main factor, the analysis of results must account for possible confounding factors, in particular we identified the following:

1. *System*: Since the experiment involves two objects (systems), i.e., Claros and WfMS, it is necessary to analyze to what extent results are influenced by the characteristics of a particular system.

^{||}http://www.rcost.unisannio.it/mdipenta/Conallen-UML-Experimental_Package.zip.



Table IV. Post-experiment survey questionnaire.

ID	Question
Q1	I had enough time to perform the lab tasks (1–5).
Q2	The objectives of the lab were perfectly clear to me (1–5).
Q3	The questions were clear to me (1–5).
Q4	I experienced no difficulty in reading the diagrams (1–5).
Q5	I experienced no difficulty in reading the source code (1–5).
Q6	How much time (as a percentage) did you spend looking at class diagrams? (A. <20%; B. >=20% and <40%; C. >=40% and <60%; D. >=60% and <80%; E. >=80%)
Q7	How much time (as a percentage) did you spend for source code browsing? (A. <20%; B. >=20% and <40%; C. >=40% and <60%; D. >=60% and <80%; E. >=80%)
Q8	I understood the meaning of Conallen's stereotypes (1–5).
Q9	I found Conallens stereotyped diagrams useful (1–5). 1 = Strongly agree, 2 = Agree, 3 = Not certain, 4 = Disagree, 5 = Strongly disagree.

2. *Lab*: It is necessary to analyze how performances change across the two laboratories, i.e., *Lab 1* and *Lab 2*.
3. *Ability*: Finally, it is necessary to investigate to what extent results depend on the ability of each subject and whether subjects with high and low ability behave differently.

To measure the comprehension level and test the null hypothesis H_{C0} , we assessed the answers to the questionnaires using an information-retrieval approach. Since the answer to each question has to be expressed as a list of system elements, i.e., classes, JSPs and HTML pages, we denote: $A_{s,i}$ as the set of elements mentioned in the answer to question i by subject s ; and C_i as the correct set of elements expected for question i .

Based on the above definition, we computed precision and recall for each answer [18]. Precision measures the fraction of items in the answer that are correct:

$$Precision_{s,i} = \frac{|A_{s,i} \cap C_i|}{|A_{s,i}|}$$

Recall measures the fraction of expected items that are in the answer:

$$Recall_{s,i} = \frac{|A_{s,i} \cap C_i|}{|C_i|}$$

To obtain an overall measure of the comprehension level, other than considering precision and recall separately, it is possible to compute the *F-measure* [18], which is a standard metric defined as the harmonic mean of precision and recall:

$$F\text{-measure}_{s,i} = \frac{2 \cdot precision_{s,i} \cdot recall_{s,i}}{precision_{s,i} + recall_{s,i}}$$

To test the null hypothesis H_{t0} , it was necessary to measure the time spent by each subject to perform the experimental task. To this aim we used the starting and end times indicated by each subject. Although a finer-grained analysis of effort could have been done, this approach is reasonable and reliable enough since (i) forms were checked by teaching assistants to ensure the



correctness of the information provided and (ii) subjects did not interrupt their laboratory task with any break.

As far as the survey questionnaire is concerned, we can define an additional derived measure based on answers to questions **Q6** and **Q7**. Q6 (Q7) represents the percentage of time spent on diagrams (code). It can be interpreted as the probability of finding a specific subject looking at diagrams (code) in any moment during the comprehension task. Since the answers are on an ordinal scale, we encode these answers to an interval scale (from 0 to 1), assigning for each answer the middle point of the interval it represents. The encoded values of **Q6** and **Q7** represent the probability of looking at diagrams (P_{diag}) and the probability of looking at code (P_{code}), respectively. To obtain a measure of how much the presence of stereotyped diagrams affects how the available sources of information are used, we can compute the odds ratio of looking at diagrams vs code:

$$OR_{diag/code} = \frac{P_{diag}/(1 - P_{diag})}{P_{code}/(1 - P_{code})}$$

4. EXPERIMENTAL RESULTS

This section reports the analysis of results obtained in the experiments we performed. We split the analyses into three parts: first we analyze the effects of the main factor, then we focus on the co-factors and eventually we analyze the data collected through the survey questionnaire.

4.1. Analysis of main factor

A first insight can be obtained by looking at the descriptive statistics in Table V and at the boxplots in Figure 1 that compare the dependent variables (F -measure, precision, recall and time) between subjects using basic UML and those using Conallen's stereotypes. Figure 1(a) highlights the benefits obtained when Conallen's stereotypes are used in terms of increased F -measure, Precision and Recall. Mann–Whitney one-tailed tests support such an evidence (p -value=0.01 for F -measure, 0.036 for precision and 0.009 for recall). Instead, there is no significant difference in terms of time needed to perform the task (p -value=0.43), indicating that the use of Conallen's stereotypes does not directly influence subjects' productivity in the comprehension task (see Figure 1(b)).

Table V. Descriptive statistics.

Statistics	Method	Differences Median	Differences Mean	Differences Std. dev.
F -measure (%)	Conallen	73.08	66.62	15.51
	UML	57.18	57.91	14.87
Precision (%)	Conallen	76.04	70.54	14.65
	UML	63.89	63.99	15.26
Recall (%)	Conallen	71.92	66.72	17.21
	UML	57.71	57.73	15.20
Time (m)	Conallen	90.50	94.00	16.40
	UML	91.50	91.60	17.54

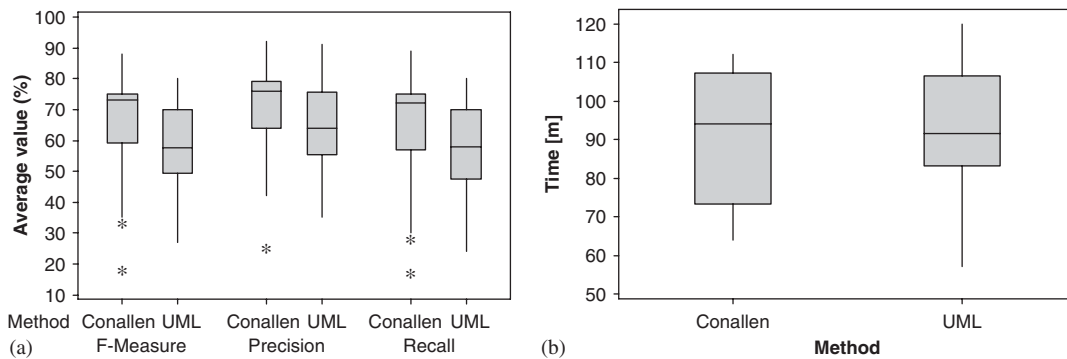


Figure 1. Boxplot of (a) *F*-measure, precision and recall and (b) time spent to perform the task.

Table VI. Wilcoxon paired test (significant value is shown in boldface).

Variable	Median	Mean	Std. dev.	<i>p</i> -Value
<i>F</i> -measure	5.65	8.38	18.36	0.045
Precision	9.68	6.72	19.06	0.066
Recall	5.03	8.35	19.11	0.053
Time	−6.50	−4.75	26.28	0.83

The chosen experiment design also allows us to perform a paired statistical test, the Wilcoxon test, for the 20 subjects who participated in both *Labs*. The test indicates (Table VI) that subjects' comprehension significantly improves in terms of *F*-measure when Conallen's stereotypes are available, and that a marginal improvement is also visible in terms of precision and recall separately. Nothing can be said regarding time: paired analysis indicates that, when using Conallen's stereotypes, subjects take more time, although such a difference is not significant.

Overall, the statistical tests performed allow us to reject the null hypothesis H_{C0} stated in Section 3, while it is not possible to reject the null hypothesis H_{T0} . In other words, results from this experiment show a significant improvement in the comprehension level when using diagrams with Conallen's stereotypes. However, the use of Conallen's stereotypes does not introduce any productivity improvement in the comprehension task; on the contrary, when looking at paired data, it appears that when using stereotypes subjects tend to use more time, although this difference is not significant and such an overhead is paid back by a significant improvement in the comprehension level.

The statistical significance alone does not tell anything about the practical impact of the treatment: it is important to measure the effect size of the main factor over the dependent variables, i.e., the magnitude of a main factor treatment effect on the dependent variables. We used the Coehn standardized difference between two groups [19], defined as the difference between means (M_1 and M_2) divided by the pooled standard deviation (σ) of both groups:

$$d = \frac{M_1 - M_2}{\sigma}$$



The effect size is considered small for $d=0.2$, medium for $d=0.5$ and large for $d=0.8$. Results showed, considering all subjects participating in the experiment, a *medium* effect size ($d=0.58$) of the *Method* on *F*-measure, and a *negligible* effect size ($d=0.077$) on time.

4.2. Analysis of co-factors

In the following, we will analyze the effect of other factors, namely *System*, *Lab* and *Ability*, over the obtained results. For the sake of brevity, we will show only the effect of these factors on the *F*-measure and time, since the effect on precision and recall is similar to that on *F*-measure. The two-way ANOVA analysis of *F*-measure by *Method* and *System*, by *Method* and *Lab* and by *Method* and *Ability* did not reveal any significant effect of these co-factors, nor any significant interaction with the *Method*. Figure 2 shows the interaction plots of the *F*-measure by *Method* and *System* and by *Method* and *Ability*. Parallel lines indicate the absence of interaction, converging lines a mild interaction, and intersecting lines the presence of interaction. Figure 2(a) indicates the presence of an interaction, yet not significant, between *Method* and *System*. In other words, the benefit introduced by the use of stereotypes is higher for Claros than for WfMS. This is because the Claros view is much more complex than the WfMS view, and stereotypes introduce benefits when comprehending that part of the class diagram. Figure 2(b) shows that there is a mild interaction between *Method* and *Ability*: *Low-ability* subjects benefit more from use of stereotypes than *High-ability* subjects, who are usually more capable of quickly finding the answer to the questions in the source code. ANOVA does not show any significant effect of the *Lab* factor on *F*-measure, nor any interaction with the *Method*.

When performing two-way ANOVA of time by *Method* and *System*, by *Method* and *Lab*, and by *Method* and *Ability*, we found a significant effect of only the *Lab* factor (Table VII). Figure 3 shows the interaction plots for the three factors. Figure 3(c) shows how the time is significantly reduced in *Lab 2* and that there is no interaction at all between *Method* and *Lab*. After working on a comprehension task in the first lab, subjects were better able to quickly analyze a system by

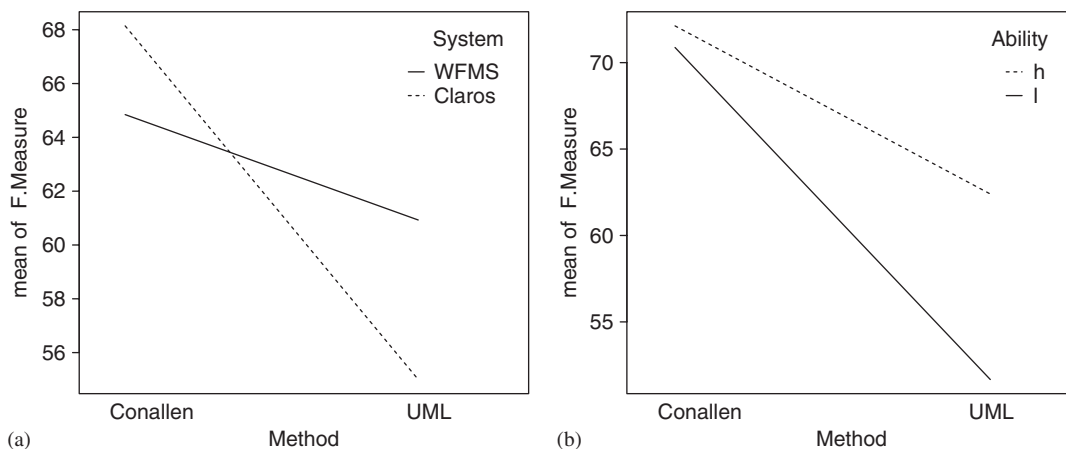


Figure 2. Interaction plots of *F*-measure (a) by method and system (b) by method and ability.



Table VII. Two-way ANOVA by Method and Lab for the time-dependent variable.

Variable	Df	Sum sq	Mean sq	F value	p-Value
Method	1	15.5	15.5	0.0895	0.7661
Lab	1	6377.4	6377.4	36.8724	1.69e-07
Method:Lab	1	0.1	0.1	0.0007	0.9789
Residuals	50	8647.9	173.0		

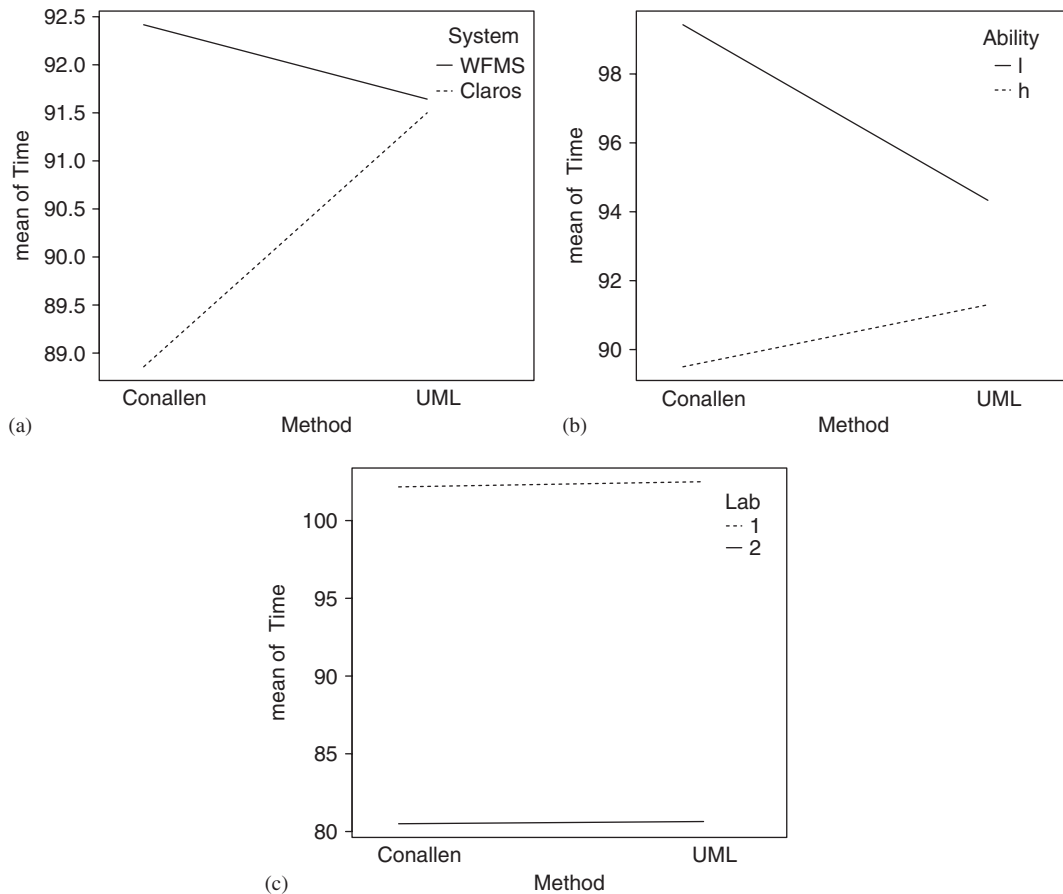


Figure 3. Interaction plots of time: (a) by Method and System; (b) by Method and Ability; and (c) by Method and Lab.

looking at diagrams and at the source code. However, while subjects were able to perform the task *quicker*, they were not able to perform it *better*, as indicated by the absence of a significant effect of the *Lab* on the *F*-measure.



Figure 3(a) shows the presence of a mild interaction between *Method* and *System*, having an effect on the time: while in the case of Claros, subjects were much more productive when using stereotypes, this was not the case for WfMS. Once again, the explanation can be that Claros has a complex view, needed to manage all the Web mail features, while the rest of the system is not particularly complex, i.e., it uses the Java application programming interface (API) to access the mail folder and to send e-mails. On the contrary, the WfMS view is quite simple, while the features it provides, proper of a workflow management system, are much more complex. Thus, for Claros the presence of a stereotyped class diagram helped subjects improve both productivity and comprehension. For WfMS stereotypes produced a negative, although not significant, effect, probably because subjects lost more time searching in the stereotypes some answers that could be found only in the source code.

As shown in Figures 2(b) and 3(b), *Method* and *Ability* interact in opposite directions with respect to *F*-measure and time. While stereotypes help *Low-ability* subjects to increase their understanding level, they also cause an increase in the time needed to perform the task. *High-ability* subjects also obtain benefits in terms of comprehension level when using stereotypes. However, (i) such a benefit is smaller and (ii) for *High-ability* subjects the use of stereotypes does not require additional time. This is confirmed by the effect sizes [19]: for *High-ability* subjects *Method* has a higher effect size on *F*-measure (2.13) than for *Low-ability* subjects (0.72). The effect size on time is in both cases small, i.e., -0.10 for *High-ability* subjects and 0.23 for *Low-ability* subjects, even though of opposite sign.

4.3. Analysis of survey questionnaires

Analysis of the survey questionnaires filled in by the subjects after each experiment can be useful to better understand the experimental results. Analyses are supported by descriptive statistics, boxplots (Figure 4) and results of Mann–Whitney test, for which the *p*-value is reported.

Overall, all subjects agreed that they had enough time to perform the task **Q1**, although this feeling was significantly stronger in *Lab 2* than in *Lab 1* (*p*-value 0.05). This confirms the quantitative results on the dependent variable time, which significantly decreased in *Lab 2*. The objectives were clear enough (**Q2**, median = 2), and became marginally clearer from *Lab 1* to *Lab 2* (*p*-value

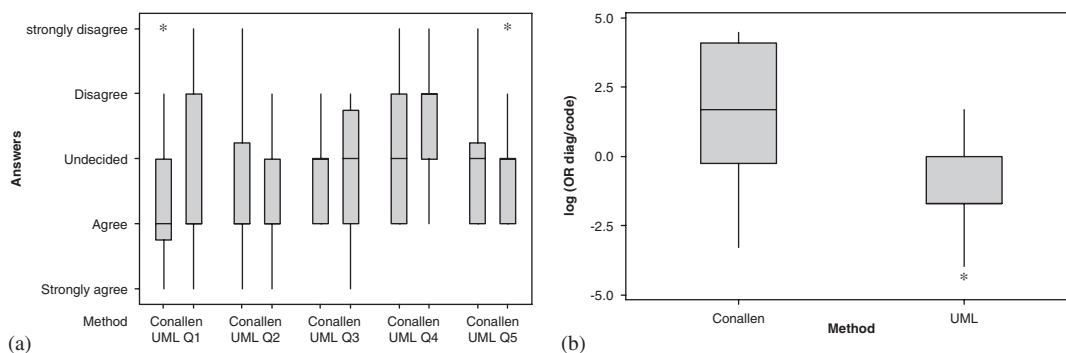


Figure 4. Survey questionnaires: (a) answers to questions Q1–Q5 and (b) odds ratio of looking diagrams vs code.



0.09). Similar agreement levels are obtained for the clarity of questions (**Q3**, median = 2) and, also in this case, there was a significant improvement between *Lab 1* and *Lab 2* (p -value 0.05). This supports the explanation provided in Section 4 about the improvement of time: after having already done a similar task, yet with a different *System* and with a different treatment for the *Method*, subjects were more confident and took less time. On comparing the results of questions **Q1–Q3** for different *Method* treatments, no significant difference can be found: the confidence on the task to be performed and the feeling that time suffices improves only across *Labs* and does not change between subjects using Conallen and those using UML. Also, other factors (*System*, *Ability*) have no significant effect on the answers.

Subjects felt an average difficulty (median = 3 in both cases) when understanding diagrams (**Q4**) and when understanding code (**Q5**). No significant difference is found across *Labs*, and for subjects having a different *Ability*. The latter indicates that, while subjects' *Ability* can have an influence on the results, as shown in Section 4, it happens that both *High-* and *Low-ability* subjects have a similar perception of the task difficulty. Instead, confirming the usefulness of Conallen's diagram, subjects had less difficulty in understanding the UML diagrams when Conallen's stereotypes were present (p -value = 0.03). By comparing data for different *Systems*, it is possible to note a marginal difference between the code of Claros and WfMS (the latter appears to be more complex, p -value 0.07). This supports what is discussed in Section 4: most of the Claros complexity is in the view (although subjects did not consider its diagrams more complex to be understood), while the WfMS source code is much more complex.

Subjects spent between 40 and 60% of their time looking at diagrams (**Q6** and **Q7**). Such a percentage did not change across *Labs*, while subjects working on WfMS spent a bit more time over diagrams. The odds ratios of looking diagrams vs code (Figure 4(b)) exhibit a significant difference (p -value $< 4 \times 10^{-5}$). In the UML groups the mean value is close to one (0.994), indicating a fair distribution of time among code and diagrams, while in the Conallen groups it is much higher (26.56), indicating that most of the laboratory time is spent on diagrams. This supports the quantitative results, revealing the usefulness of stereotypes: not only do subjects using them improve their comprehension level, but also they can better exploit design documentation to understand the system instead of directly looking at the source code.

Finally, subjects using Conallen's stereotypes agreed that they understood the notation well (**Q8**, median = 2) and found the stereotypes useful to support the task they performed (**Q9**, median = 2). No significant difference can be observed across *Labs* and for different *Systems*.

4.4. Discussion

The main result of this study is that Conallen's diagrams provide a significant support to Web application understanding. This can be explained by the presence of Web application features represented explicitly in Conallen's diagrams. These diagrams highlight concerns such as the navigational structure, the relationships between server pages and the pages they generate, the data submitted through a form to a server page, and the presence of mechanisms such as sessions. These concerns are necessary for maintainers to obtain a broader view on how the Web application is structured, which are the navigational paths, how information flows across pages, etc., before getting into source code details. All this is not described in basic UML diagrams; thus, subjects using them are forced to find in the source code information that, instead, Conallen's diagrams show explicitly. Moreover, even when source code browsing is necessary, Conallen's diagrams provide a support for guiding



source code understanding, by guiding the maintainer to identify where some details relevant to a particular maintenance task can be found. These findings, besides being supported by statistics on quantitative data, are confirmed by the answers to questions Q6 and Q7 of the survey: subjects using basic UML spent significantly more time on the code than those using Conallen. Thus, Conallen's explicit representation of Web-specific notions gave a relevant contribution to the comprehension of the considered Web applications.

Overall, the use of stereotypes did not introduce any significant improvement in terms of the time needed to perform the task: while Conallen's notation helped to better comprehend the Web applications, this does not mean that subjects were able to be more productive when carrying out the comprehension task. However, when looking at factors that could have influenced the time-dependent variable, it was found that the *Lab* factor had a significant effect on time. When performing the task of the second laboratory, subjects seem to have acquired some knowledge on how diagrams can be useful and can effectively support the completion of the assigned task. In other words, subjects became more familiar with the experimental settings and with the activities to be carried out. This is confirmed by some answers they provided to the survey questionnaire (e.g., Q1–Q3), where adequacy of the allowed time, clarity of objectives and clarity of questions increased from *Lab* 1 to *Lab* 2. Despite such an evident learning, the effect of using Conallen's notation on the comprehension level did not significantly change across laboratories. This can be explained as follows:

1. Subjects were already confident in Conallen's notation after the pre-experiment training laboratory.
2. The practice made during the first laboratory task was useful to gain better confidence with the experimental setting; however, it did not suffice to significantly increase confidence in Conallen's notation.

Other than the effect of *Lab* on time, the two-way ANOVA indicates a small, not significant difference between Claros and WfMS: stereotypes were more useful for Claros than for WfMS. This can be explained by the specific features of these two applications. Claros is larger and more complex; thus, diagrams are expected to be more beneficial for it. Moreover, the navigation structure prevails over the business logic for Claros, while the opposite is true for WfMS, so that Conallen's diagrams are expected to be more informative for Claros. Finally, subjects were more familiar with the application domain of Claros (Web mail) than WfMS (workflow management), so that they had to spend more time on the code to understand the business logic of WfMS (see also the answers to Q6 and Q7).

The subjects' *Ability* had a mild influence on both time and comprehension level. However, the effect observed was in two opposite directions, as discussed in Section 4. In other words, Low-ability subjects found more benefits in the use of Conallen's notation, although this caused an increase in time. When stereotypes are available, *Low-ability* subjects find a support guiding them to correctly answer the questions: while the use of such a support requires spending more time, since stereotyped diagrams contain more details, such an overhead is paid back by a higher comprehension level. For *High-ability* subjects, the benefits introduced by Conallen's stereotypes are smaller in terms of increased comprehension level. On the other hand, they did not cause an overhead in terms of time needed: on the contrary, they helped to save time, avoiding the need for subjects to look deeply at the source code.



4.5. Threats to validity

This section discusses the threats to validity that can affect the results of our experiment: *internal*, *construct*, *conclusion* and *external* validity threats [10].

Internal validity threats can be due to the learning effect experienced by subjects between *Labs*. This is mitigated thanks to the experimental design: subjects worked, over the two *Labs*, on different *Systems* and using two different levels of the main factor (UML vs Conallen). Nevertheless, there is still the risk that, during *Labs*, subjects might have learned how to comprehend the source code of a Java Web application and how to read UML diagrams. We tried to limit this effect by means of a preliminary training phase. Subjects were previously trained on the topics affecting the experiment. Moreover, the *Lab* factor has been accounted as a factor in the analysis of results. ANOVA showed no significant effect on the comprehension level due to *Lab*, although subjects spent significantly less time in *Lab 2*. This means that, despite an obvious maturation effect, such an effect is limited to the capability of subjects to perform the task faster, i.e., in *Lab 2* they know better how to understand a system by browsing the source code and the design documents. Threats due to the instrumentation were limited: the tasks and forms presented to subjects were clear enough, as they reported on the survey questionnaires, and the time sheets used to compute the effort were checked by teaching assistants. The effect of subjects' *Ability*, although not significant, was accounted by means of blocking and by using ANOVA analyses and interaction plots. Finally, although a few subjects participated in one *Lab* only (either the first or the second), we still had enough data (20 points) to make the application of paired test possible, and we complemented paired analyses with unpaired analyses over the whole data set.

Construct validity threats that may be present in this experiment, i.e., interactions between different treatments, were mitigated by a proper design that allowed to separate the analysis of the different factors and of their interactions. To avoid social threats due to evaluation apprehension, students were not evaluated on their performance in the *Lab*. Finally, subjects did not know the experimental hypotheses. We are aware that computing precision and recall on results of a questionnaire constitute a simplified view of program comprehension; however, identifying the artifacts affected by a maintenance request is a crucial comprehension step a maintainer has to perform.

Regarding *conclusion validity*, appropriate tests were performed to check whether null hypotheses could be statistically rejected or not. In particular, the chosen experiment design permitted the use of paired tests, although results of unpaired tests are available for a larger data set (see Section 4). In cases where differences were present but not significant, this was explicitly mentioned. Non-parametric tests were used in place of parametric tests where the conditions necessary to use parametric tests did not hold. Also, ANOVA results were confirmed by non-parametric tests (Friedman test). The measures chosen to evaluate the comprehension, i.e., precision, recall and *F*-measure, allowed us to evaluate the questionnaire answers in an objective manner, avoiding the requirement for subjective scores. The comprehension questionnaire covered different aspects of the system, so that the high number of correct answers indicates a good comprehension level. Survey questionnaires, mainly intended to get qualitative insights, were designed using standard formats and scales [17]. This allowed us to use statistical tests to analyze also differences in the feedbacks.

External validity threats are always present when experimenting with students. The selected subjects represent a population of undergraduate students specifically trained on Web development technologies and software engineering methods. Most of them are ready to become junior developers in industrial environments. Nevertheless, the question of whether more skilled subjects, such



as graduate students or professionals, would gain similar benefits from the use of Conallen's notation is worth further experimentation. The experimental objects were two real Web applications belonging to different domains. This makes the context quite realistic, although many commercial or institutional Web applications may be larger and different. However, given the time constraints of the experiment and the involved subjects (students), it was not feasible to consider larger examples. Also, further experiments are desirable, since systems from other domains or systems realized using other Web development technologies (e.g., Ajax, frameworks such as Structs, Ruby on Rails, etc.) could lead to different results.

5. RELATED WORK

This section discusses related literature concerning Web application design methodologies, and studies aimed at assessing the use of Web application-specific notations and the use of stereotypes in comprehension tasks.

5.1. Web application design methodologies

Several Web application modeling methods have been proposed for different purposes, such as design, architecture recovery, re-structuring and testing. Each method emphasizes some particular aspect of a Web application. Many works have been dedicated to the problem of Web application design: they include languages, models and graphical notations useful for the specification of Web applications.

Among the design methodologies [1,3,5,6,20,21] that have been proposed in support to Web development, those conceived for the specification of the navigation and structure of the application—for example W3DT [20], WebML [1], WAE [5], and WSDM [3]—are closer to the implementation. For this reason they are more useful for understanding and maintenance tasks. Moreover, models based on navigation and structure of the application are simpler to extract (semi)automatically through a reverse engineering process than through higher-level abstraction models.

Bichler and Nusser [20] proposed the tool W3DT Web-Designer, developed especially to facilitate the design of Web sites. Their Web site model introduces the concepts of dynamic page and dynamic link, while it does not consider frames.

WebML [1] is a visual notation for specifying the content, composition and navigation features of Web applications. WebML proposes three models: one for the data (data model), one for the navigation (hypertext model) and one for defining the look and feel of pages (presentation model). It is built on the top of ER (entity–relationship diagrams) and UML. WebML is accompanied by a CASE tool called WebRatio**, which automatically generates Web application templates from WebML diagrams.

The UWE [22] methodology provides a systematic, iterative and incremental approach for the development of Web applications. UWE, like WAE, is an extension of UML. Similar to other Web design methods, UWE distinguishes the following concerns: content, navigation structure, business processes and presentation. In a single interaction, a use case model, a conceptual model, a navigational model and a presentation model are built or refined. Similar to WebML, UWE is also

**<http://www.webratio.com>.



accompanied by a CASE tool. The tool ArgoUWE [23] offers tailored editors for the UWE notations and provides several semi-automatic model transformations that occur in the UWE process.

5.2. Experiments aimed at assessing the comprehension of software documentation

The usefulness of graphical elements in software architecture diagrams has been assessed experimentally by Bratthall and Wohlin [24], who compared 10 different representations aimed at enriching the design with qualitative information. As in their case, the presence of graphical elements improved the comprehensibility of Web application design elements. A series of controlled experiments, performed both in academia and in industry, focused on the use of stereotypes for comprehending object-oriented applications in the telecommunication domain was conducted by Kuzniarz *et al.* [16,25]. They showed that the use of stereotypes significantly helps both students and industrial developers to improve the comprehension. To our knowledge, their study is the most similar to ours, although there are some important differences:

1. programming style (traditional object-oriented system vs Servlet/JSP Web application);
2. size and complexity of the considered applications (the applications in [25] count 14 classes, while the two systems we used have 78 and 92 files comprising Java classes and JSPs);
3. application domain (telecommunication vs Web applications for workflow management and Web mail client);
4. stereotypes being evaluated (*ad hoc* stereotypes introduced by Staron *et al.* [16] for the telecommunication domain vs Conallen's stereotypes [5]); and
5. above all, while in Staron *et al.*'s experiments subjects relied only on diagrams, in our experiment they also had the source code available. This is, in our opinion, more realistic for a software maintenance task. Also, the presence of source code explains why, in our case, high-ability subjects obtained less benefits from the use of stereotypes.

UML limitations in aiding program understanding are highlighted in experiments performed by Tilley and Huang [26]. They highlight that UML does not provide sufficient support to represent domain knowledge, support that stereotypes surely contribute to improve.

In a companion paper, we report some partial analyses and results from the same experiment [27]. The present paper describes all the details of the experiment and tests a second, relevant hypothesis: effects on the time needed to perform the comprehension task. It also discusses in detail the effects of different co-factors (Ability, System, Lab) and measures the effect size. A thorough study about the effect of ability and experience on the use of stereotypes is part of an experimentation [28], conducted as a follow-up of the present work.

6. CONCLUSIONS AND WORK-IN-PROGRESS

This paper reported and discussed a controlled experiment aimed at assessing the effectiveness of specialized design notations (specifically, Conallen's diagrams) compared with general-purpose notations (UML) in support to the comprehension and evolution of Web applications. Experimental results indicate that diagrams based on Web-specific stereotypes simplify the comprehension task in a substantial way, with a *medium* effect size. Subjects using Conallen's diagrams spent more



time on the design view of the application under maintenance and gain more information about it than subjects using standard UML diagrams only. Although the total time for task completion is not significantly different, subjects using Conallen's diagrams were able to perform maintenance devoting less time to code reading and eventually gaining a higher level of understanding of the application. In turn, this is expected to have positive effects on future maintenance activities as well as on the quality of the intervention at hand.

Replication of this study is fundamental to corroborate our findings and to produce a complete picture of the effects of using stereotyped diagrams. We provide the full experimental package to other researchers to support and encourage replication. We are also considering variants of the experimental settings described in this paper, involving professionals. The aim is to understand how subjects with different levels of ability and skill take advantage of Web-specific design models.

ACKNOWLEDGEMENTS

We are grateful to Anna Perini and Angelo Susi, the teachers of Laboratory of Software Engineering, who accepted to host the experiment inside their course. Luana Stedile contributed to the in-field execution of the experiment and to the collection of the data. Finally, we thank all the students who participated in the experiment. Without them this work would not have been possible.

REFERENCES

1. Ceri S, Fraternali P, Bongio A, Brambilla M, Comai S, Matera M. *Designing Data-intensive Web Applications*. Morgan Kaufmann: Los Altos CA, 2002.
2. Knapp A, Koch N, Zhang G. Modeling the Structure of Web Applications with argoUWE. *Proceedings of the 4th International Conference on Web Engineering (ICWE'04)*. Springer: Berlin, 2004; 615–616.
3. De Troyer OMF, Leune CJ. WSDM: A user centered design method for Web sites. *Proceedings of the Seventh International Conference on World Wide Web 7*. ACM Press: New York, NY, 1998; 85–94.
4. Schwabe D, Rossi G. An object oriented approach to Web-based application design. *Theory and Practice of Object Systems* 1998; 4(4):207–225.
5. Conallen J. *Building Web Applications with UML* (2nd edn). Addison-Wesley: Reading MA, 2002.
6. Schwabe D, Guimarães R, Rossi G. Cohesive design of personalized Web applications. *IEEE Internet Computing* 2002; 6(2):34–43.
7. Rumbaugh J, Jacobson I, Booch G. *Unified Modeling Language Reference Manual*. Addison-Wesley: Reading MA, 2004.
8. Booch G, Rumbaugh J, Jacobson I. *The Unified Modeling Language—User Guide*. Addison-Wesley: Reading MA, 1998.
9. Conallen J. Modeling Web application architectures with UML. *Communications of the ACM* 1999; 42(10):63–70.
10. Wohlin C, Runeson P, Höst M, Ohlsson MC, Regnell B, Wesslén A. *Experimentation in Software Engineering—An Introduction*. Kluwer Academic Publishers: Dordrecht, 2000.
11. Juristo N, Moreno AM. *Basics of Software Engineering Experimentation*. Kluwer Academic Publishers: Englewood Cliffs NJ, 2001.
12. Brugali D, Torchiano M. *Software Development: Case Studies in Java*. Addison-Wesley: Reading MA, 2005.
13. Ricca F, Di Penta M, Torchiano M, Tonella P, Ceccato M. How design notations affect the comprehension of web applications. *Technical Report*, RCOST, University of Sannio, Italy. <http://www.rcost.unisannio.it/mdipenta/jsme07-tr.pdf> [April 2007].
14. Briand LC, Labiche Y, Di Penta M, Yan-Bondoc HD. An experimental investigation of formality in UML-based development. *IEEE Transactions on Software Engineering* 2005; 31(10):833–849.
15. De Lucia A, Di Penta M, Oliveto R, Zurolo F. Improving comprehensibility of source code via traceability information: a controlled experiment. *Proceedings of the 14th International Conference on Program Comprehension (ICPC)*, June 2006. IEEE Computer Society: Silver Spring MD, 2006; 317–326.
16. Staron M, Kuzniarz L, Wohlin C. Empirical assessment of using stereotypes to improve comprehension of UML models: A set of experiments. *Journal of Systems and Software* 2006; 79(5):727–742.
17. Oppenheim AN. *Questionnaire Design, Interviewing and Attitude Measurement*. Pinter: London, 1992.

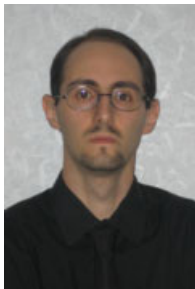


18. Frakes WB, Baeza-Yates R. *Information Retrieval: Data Structures and Algorithms*. Prentice-Hall: Englewood Cliffs NJ, 1992.
19. Cohen J. *Statistical Power Analysis for the Behavioral Sciences* (2nd edn). Lawrence Earlbaum Associates: Hillsdale NJ, 1988.
20. Bichler M, Nusser S. Developing structured WWW-sites with W3DT. *Proceedings of the WebNet'96 World Conference of the Web Society*, 1996; 7–12.
21. Isakowitz T, Kamis A, Koufar M. Extending the capabilities of RMM: Russian dolls and hypertext. *Proceedings of the Thirtieth Annual Hawaii International Conference on System Sciences (HICSS-30)*. IEEE Computer Society: Silver Spring MD, 1997; 177–187.
22. Hennicker R, Koch N. A UML-based methodology for hypermedia design. *UML 2000—The Unified Modeling, Language, Advancing the Standard, Third International Conference*, 2–6 October 2000; 410–424.
23. Knapp A, Koch N, Zhang G, Hassler HM. Modeling business processes in Web applications with ArgoUWE. *UML 2004—The Unified Modelling Language: Modelling Languages and Applications. Seventh International Conference*. 11–15 October 2004; 69–83.
24. Brathall L, Wohlin C. Is it possible to decorate graphical software design and architecture models with qualitative information? An experiment. *IEEE Transactions on Software Engineering* 2002; **28**(12):1181–1193.
25. Kuzniarz L, Staron M, Wohlin C. An empirical study on using stereotypes to improve understanding of UML models. *Proceedings of the International Workshop on Program Comprehension (IWPC)*. IEEE Computer Society: Silver Spring MD, 2004; 14–23.
26. Tilley S, Huang S. A qualitative assessment of the efficacy of UML diagrams as a form of graphical documentation in aiding program understanding. *SIGDOC '03: Proceedings of the 21st Annual International Conference on Documentation*. ACM Press: New York, 2003; 184–191.
27. Ricca F, Di Penta M, Torchiano M, Tonella P, Ceccato M. An empirical study on the usefulness of Conallen's stereotypes in Web application comprehension. *Proceedings of the International Symposium on Web Site Evolution (WSE 2006)*. IEEE Computer Society: Silver Spring MD, 2006; 58–65.
28. Ricca F, Di Penta M, Torchiano M, Tonella P, Ceccato M. The role of experience and ability in comprehension tasks supported by UML stereotypes. *Proceedings of the International Conference on Software Engineering*. IEEE Computer Society: Silver Spring MD, 2007.

AUTHORS' BIOGRAPHIES



Filippo Ricca is with CINI (Laboratorio Iniziativa Software FINMECCANICA/ELSAG spa) at DISI (Department of Computer and Information Science, Genova) and collaborates with the Software Engineering Group of DISI. He received his laurea degree in Computer Science from the University of Genova, Italy, in 1994, and his PhD degree in Software Engineering from the same University, in 2003, with the thesis 'Analysis, Testing and Re-structuring of Web Applications'. From 1999 to 2006, he worked with the Software Engineering group at ITC-irst, Trento, Italy. During this time he was part of the team that worked on reverse engineering, re-engineering and testing. His current research interests include reverse engineering, AOP, empirical studies, Web applications and testing.



Massimiliano Di Penta is an assistant professor at the University of Sannio in Benevento, Italy, and research leader at the Research Centre On Software Technology (RCOST). He received his PhD in Computer Engineering in 2003 and his laurea degree in Computer Engineering in 1999. His main research interests include software maintenance, reverse engineering, empirical software engineering and service-oriented software engineering. He has authored of over 90 papers published on referred journals, conferences and workshops. He is the program co-chair of the 14th Working Conference on Reverse Engineering (WCRE 2007), the 9th International Symposium on Web Site Evolution (WSE 2007), the 9th IEEE International Workshop on Principles of Software Evolution (IWPSSE 2007) and a steering committee member of CSMR and STEP. He has been the program co-chair of WCRE 2006, SCAM 2006 and STEP 2005. He serves and has served the program and organizing committees of conferences and workshops such as CSMR, GECCO, ICSM, IWPC, SCAM, SEKE, WCRE and WSE. He is a member of the IEEE, the IEEE Computer Society and of the ACM.



Marco Torchiano is an assistant professor in the Department of Control and Computer Engineering at Politecnico di Torino, Italy; previously he was a post-doctoral research fellow at the Norwegian University of Science and Technology (NTNU), Norway. He received his MSc and PhD degrees in Computer Engineering from Politecnico di Torino. He participated in several EU and national research projects and served in the program committees of several international conferences. He recently organized the second International Workshop on Empirical Studies in Reverse Engineering. His current research interests are: empirical software engineering, OTS-based development and software engineering for mobile and wireless applications. He published more than 40 research papers in international journals and conferences. He is the co-author of the book 'Software Development—Case studies in Java' from Addison-Wesley, and co-editor of the book 'Developing Services for the Wireless Internet' from Springer.



Paolo Tonella is a senior researcher at Fondazione Bruno Kessler-Irst, Trento, Italy. He received his laurea degree cum laude in Electronic Engineering from the University of Padua, Italy, in 1992, and his PhD degree in Software Engineering from the same University, in 1999, with the thesis 'Code Analysis in Support to Software Maintenance'. Since 1994, he has been a full-time researcher with the Software Engineering group at ITC-Irst (now Fondazione Bruno Kessler-Irst). He is the author of the book 'Reverse Engineering of Object-oriented Code', Springer, 2005. His current research interests include reverse engineering, AOP, empirical studies, Web applications and testing.



Mariano Ceccato is a post-doctoral researcher in FBK-Irst (Fondazione Bruno Kessler) in Trento, Italy. He received his degree in Software Engineering from the University of Padova, Italy, in 2003, and his PhD in Computer Science from the University of Trento in 2006, with the thesis 'Migrating Object-oriented code to Aspect-oriented Programming'. The thesis addresses the problems of aspect mining and aspect refactoring to introduce aspect-oriented constructs into already existing software systems. His research interests are aspect-oriented programming, source code analysis, program transformations and empirical software engineering. He has been involved in the organization and program committee of a number of AOP-related events, such as Late Workshop (which he co-founded) held within the major Aspect-Oriented Programming conference (AOSD) and Third European Workshop on Aspects in Software. He reviews papers for international journals such as the *Journal of Software Maintenance and Evolution* and *IET Software*.