Journal of Information Technology Education Volume 2, 2003

Empirical Evidence Justifying the Adoption of a Model-Based Approach in the Course Web Applications Development

Borislav Roussev
Susquehanna University, Selinsgrove, PA, USA

roussev@susqu.edu

Executive Summary

With the ever-increasing role of business people in software development there is a growing need for business schools to offer courses in e-business and e-commerce applications development.

This paper presents the results of a student survey evaluating the applications development skills acquired by business students exposed to two different approaches to teaching the course E-business applications development. The first group was taught using a model-based approach, while the second one was taught using a traditional code-based approach. In the model-based approach the environment model of evaluation was used to introduce the basic programming constructs. The UML Web Modeler profile and statecharts were employed to abstract from the intricacies and the distributed nature of Web-based information systems. A major constituent of this approach was the development of a system model. The underlying assumption was that adopting a model-based approach would enhance students' ability to think and reason formally about, develop rigorously, and program better E-business applications. The contention was that learners would perceive coding as yet another view in the system model. It was believed that having defined the components' interfaces, students would be bound to experience fewer difficulties when writing the code. In the code-based approach students are exposed to Web programming without being required to develop a system model.

The two groups of students had to develop an e-commerce application using the auction model. To determine if there is a predictive relationship between teaching method and applications development skills a regression analysis was performed. It was considered whether other factors, such as student abilities and academic standing, would have any effect on applications development skills. When applications development skills was regressed on five independent variables, the equation was found to be statistically significant (F=4.92, p value=.000). When the regression coefficients for the individual predictors were examined, only teaching method was statistically significant.

The quantitative results and the qualitative analysis showed that teaching using a model-based approach has brought about tangible changes in students' modeling and programming skills. The enhanced growth in critical thinking and synthesis skills is attributed to the interrelatedness and interdependence of modeling and abstraction whose conjoint teaching and practice proves very fruitful.

Keywords: e-business applications development, learner-oriented approach, model-driven design, web-engineering, web-programming

Introduction

Current trends in software development exert an enormous influence on both end users and IS de-

Editor: Lorraine Staehr

velopers. As software becomes ubiquitous, the relationship between business people and software developers is undergoing fundamental changes. For a system to be of value to its users, it has to reduce the user's entropy, i.e. level of uncertainty (Shannon & Weaver, 1949, Cohen, 1999). In this respect, the role of business people has become crucial. To reduce the system's entropy, adequate and precise user requirements have to be elicited. This can only happen when all stakeholders are actively engaged in the software process. Modern software development processes, e.g. the Unified Process (UP), engage business people and are driven by user requirements (Jacobson et al., 1999). Another reason for end users to understand better software development is the ever-growing trend toward software reuse and customization. At present, software is commonly adapted, composed from reusable components and frameworks, and created by business people rather than IS developers (Shaw, 2000). Modern software processes are also architecture-centered (Jacobson et al., 1999). The basic promise of the software architecture approach is that better software systems can result from modeling their important architectural aspects early on in the development lifecycle (Medvidovic et al., 2002). Object-oriented modeling, dominated by UML (Booch et al., 1999), has become the de-facto and de-jure standard in software development (Johnson, 2000). Though object-oriented modeling is supported by object abstraction, this device alone cannot capture aspects bigger than objects. As a result, the gain from reusability at object/class level is insignificant (Johnson, 2000). The real benefit from reusability comes at component level, and even more so at architecture level (Gamma et al., 1994).

To sum up, the days of the closed workshop model, where developers work in isolation from end users, are over (Shaw, 2000). Modern software practices call for the active involvement of business people in the software process and the comprehensive understanding of the enterprise's business processes by IS developers (Nuseibeh & Eastserbrook, 2000). Are these new requirements reflected adequately in the IS and Business curricula and in particular in the E-business applications development course (a.k.a. Web-based programming, E-commerce systems), the basic hands-on-experience course, where students develop Web-based e-commerce applications?

E-business applications development is part of the IS component of the core curriculum at our business schools. It is the first programming course for IS majors and minors and the only such course mandatory for all business students. The IS core of the business curriculum offers four courses: (1) Using databases; (2) Systems analysis and design; (3) E-business applications development; and (4) Management support systems. The typical organization of E-business applications development is shown in Table 1.

#	**Topics**
1.	Intermediate HTML (formatting, hyperlinks, tables, forms).
2.	Network-centric computing (client-server model, Internet and HTTP protocol).
3.	Client-side JavaScript (or VBScript).
4.	Server-side VBScript with Embedded SQL

Table 1: Topics in E-business applications development.

Each programming language taught in the course is based on a different programming paradigm. JavaScript and VBScript are imperative, loosely typed, scripting languages enhanced with object-oriented features. SQL is a declarative 4GL language, and HTML is a markup language. In addition, students have to learn the client-side extension of JavaScript (or VBScript) and the server-side extension of VBScript. Finally, students have to master the client-server model, networking with HTTP, and the document object model (DOM) event model.

Course material assimilation and practical skills acquisition become difficult tasks. The consequences of this course organization are negative. E-business applications development is mainly a code-based course. Business and IS students learn little about software development, nor can they gain a better understanding and appreciation of the power and limitations of E-business applications, i.e. how e-commerce systems function, since they struggle with coding every step of the way. As a result, the broad-brush picture of e-business is blurred.

This paper presents results from the evaluation of a new approach to teaching E-business applications development. The aims of the new approach are to teach students how e-commerce applications operate and how such applications are built using modeling and abstraction. To assess the knowledge and skills acquired in this course, we conducted a formal experiment with two groups of students at the end of the semester. The treatment group, consisting of sixty five students, was taught E-business applications using a model-based approach. The controlled group, consisting of forty seven students was taught E-business applications using a code-based approach. The experiment yielded interesting and positive results.

The rest of the paper is structured as follows. Section 2 presents the new course structure, and reviews the topics covered in the course. Next, Section 3 describes the study conducted to evaluate the proposed approach and discusses the way the course was assessed. Then, Section 4 attends to the statistical analysis of the study's outcome. Section 5 discusses the results and presents important observations made. The final section summarizes the experience gained and concludes.

#	**Aims**
1.	Appreciation of the power and limitations of E-business applications.
2.	How Web-based information systems and e-commerce systems function.
3.	To expose learners to the whole software development lifecycle.
4.	See clearly the relations between user requirements and software architecture.
5.	See the relation between software architecture and program code.
6.	Acquire skills to develop software architectures using a subset of Web modeler UML profile (Conallen 1999).
7.	Implement successfully software architecture components, including user interfaces, controllers, and persistent entities, using Web technology and RDBMS.
8.	To understand that software must be designed before it is programmed.

Table 2: Intended learning outcome.

Model-Based Approach to Teaching E-Business Applications

In the previous section we discussed the ever-growing role of business people in software development. We can reverse-engineer the analyzed trends to obtain the course aims and objectives. An important guideline in this endeavor will be the ladder of cognitive skills (Bloom, 1956, Huba & Freed, 2000, Gersting et al. 2001, Machanick, 1998).

The aims of our E-business applications development course are summarized in Table 2, ordered from more general ones to more specific ones. Note, that we do not advocate a software engineering perspective such as explicit consideration of the software development process. We want our students to experience the added power of abstraction resulting from modeling in order to understand better how Web-based information systems function. We try to make clear the distinction between the "analysis" perspective and the "implementer" perspective, thus supporting learners in the creation of a sound and consistent mental model for developing and reasoning about e-business systems. It is important to observe, that unless learners develop effective analytical and synthesis skills, the course aims and objectives are unattainable.

We developed and gave a course on E-business applications development employing a model-based approach. A major constituent of this approach is the development of a system model. The system model abstracts from (in the sense of suppress) irrelevant real world details as well as from implementation details. The underlying assumption is that adopting a model-based approach will enhance students' ability to think and reason formally about, develop rigorously, and program better E-business applications. We contend that learners will perceive coding as yet another view in the system model. A clear-cut asset is that students, having first defined the components' interfaces, are bound to experience fewer difficulties when writing the code. We want learners to embrace the idea that software can be designed before it is programmed, and even that software need not be programmed at all, at least not by humans (Mellor & Balcer, 2002). Another upshot we are hoping for is that students will feel lost if asked to code before designing a model. We aim at liberating students' thinking from the constraints of programming languages. The material in E-business applications is structured as follows:

#	Topics	# weeks
1.	Environment model of evaluation and core JavaScript	3 weeks
2.	UI in HTML: formatting, tables, forms, hyperlinks and HTTP	3 weeks
3.	Relational databases and SQL	1 week
4.	Analysis/Design system model, Web Modeler UML profile, Server-side JavaScript with UML, and embedded SQL	7 weeks

Client-side scripting with JavaScript is completely missing from the topics covered. The rationale behind this decision is that client-side scripting gives little insight into the workings of enterprise information systems.

Models are used to introduce the basic programming constructs. To this end, we use the environment model of evaluation (Baber, 1987) as a virtual machine interpreting the programs written in JavaScript, and UML class diagrams (Booch et al., 1999) and statecharts (Booch et al., 1999, Harrel, 1987) to abstract from the intricacies and the distributed nature of Web-based enterprise information systems. The environment model of evaluation is a notation used in formal proofs of program correctness. The model was simplified to suit the learners' level and needs (Roussev, 2003).

Modeling program execution: Environment model of evaluation

To introduce the notion of state and thus help students appreciate the environment model of evaluation, we begin by modeling with labeled transition systems, a subset of the statecharts model. Our experience confirmed the findings of (Davis 1988) that the introduction of this model takes an hour on average. This model is a prelude to use case scenarios and program state.

For practice, a JavaScript interpreter developed by Nombas, called ScriptEase (Script Ease 2003), is used. The availability of a JavaScript interpreter decreases the complexity of the course radically. Without the interpreter, students have to write programs based on two different paradigms and in two different languages—HTML and JavaScript. The former is declarative while the latter is imperative enhanced with object-oriented features. To make matters worse, the snippets of code written in the two different languages have to interact through the event model defined by DOM.

The semantics of the assignment statement, the sequence of statements (block), the *if* statement, and the iterator *for*/*in* are defined operationally by specifying the effect of their execution on an environment. The programming constructs are practiced using the JavaScript interpreter. Functions as computational objects expressing named compound computations, collections, and objects as software analogs of real-world objects are introduced without defining their semantics in the environment model of evaluation.

The exercises given are mainly to test students' skills in applying the material rather than asking them to synthesize programs. Synthesis is one of the highest order cognitive skills. By the end of the first three weeks, students are able to apply mechanically the environment model of evaluation to predict the behavior of a program of any size. Tasks on object-oriented problems are restricted to using the `String`, `Array`, `Math`, and `Date` objects and their respective methods. This organization of the material is in accordance with the ladder of cognitive skills (Bloom, 1956), where factual knowledge, theory comprehension and theory application come before analysis and synthesis. The negative impact of reversing the skills order in the computer science curriculum is discussed in (Machanick, 1998, Gersting et al., 2001).

Modeling Web applications: Web modeler UML profile

A *Web system* is a system that uses Web server, network, HTTP, and browser. A *Web application* is a Web system in which user input changes the system state. What distinguishes a Web application from a Web site or from a database driven Web system is that a Web application executes business logic to modify the business state of a system. The emphasis of the modeling efforts in a Web application, therefore, is on business logic and business state.

Web applications are based on the client-server paradigm. A Web application's principal communication protocol is the connectionless HTTP. It is assumed that business logic is executed only on the application server (embedded in a Web server). Business logic and business objects are implemented in JavaScript. The server-side extension to JavaScript provided by the Active Server Pages scripting environment (ASP, 2003) is used.

In E-business applications development we focus on analysis and design of Web applications. Analysis is primarily concerned with conceptual decomposition into components in terms of their meaning to the users of the system (Jacobson et al., 1992). The analysis level class model is the "first cut" at the system architecture. The design model defines the realization of the conceptual components, identified during analysis, as collaborations among subsystems, classes and interfaces. For small- and middle-scale projects, there is no compelling reason to keep around two models (especially if a CASE tool is not used), so we work with one analysis/design model (A/DM).

The UML models employed are class diagrams, occasionally sequence diagrams, and statecharts diagrams (already introduced in the environment model of evaluation).

The architecturally significant components of Web applications need to be modeled. These components are pages, hyperlinks and dynamic content creating the pages. To map these artifacts to modeling elements a subset of the UML profile presented in (Conallen, 1999) with a few simplifications is used.

In Web modeler profile, each web page is modeled with a UML class, and its relationships to other pages represent hyperlinks. However, this abstraction breaks down for server-scripted pages. These pages exhibit one behavior on the server and a completely different behavior on the client (browser). Conallen proposes that the principle of "separation of concerns" be used. He suggests that the server side aspect of a server page be modeled with one class and the client side aspect with another. In Figure 1, `Clock.asp` and `Time.html` represent respectively the server-side and client-side view of one and the same server-scripted page.

We enrich gradually core JavaScript by introducing the server-side API classes: `Request`, `Response`, `Session` and `Application`. Finally, we cover embedded SQL with basic `select`, `insert`, `update` and `delete` queries.

Students create applications from all major e-commerce models: business-to-customer (e-store using a virtual shopping cart and ticket booking); customer-to-customer (payment service); peer-to-peer (chat

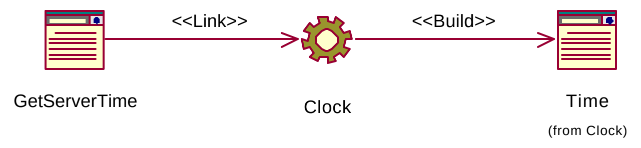

Figure 1: Generic client-server interaction in Web applications.

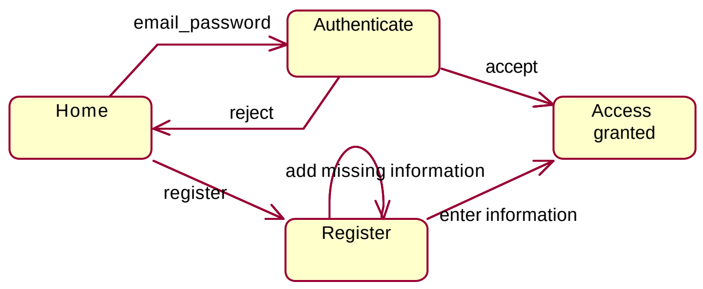

Figure 2: Sign up and Register statechart diagram.

room); business-to-business (integration of a payment service with an e-store).

Learners are given tasks described in natural language as seen by end users. Then, they have to formalize the informal description, build the system architecture and subsequently create an implementation. A sample task, a customer sign-up/register module, is given next. Figure 2 shows a behavioral specification of the module. This is the first step in converting an informal description into a more formal, not necessarily complete, specification. The module's A/DM is presented in Figure 3.

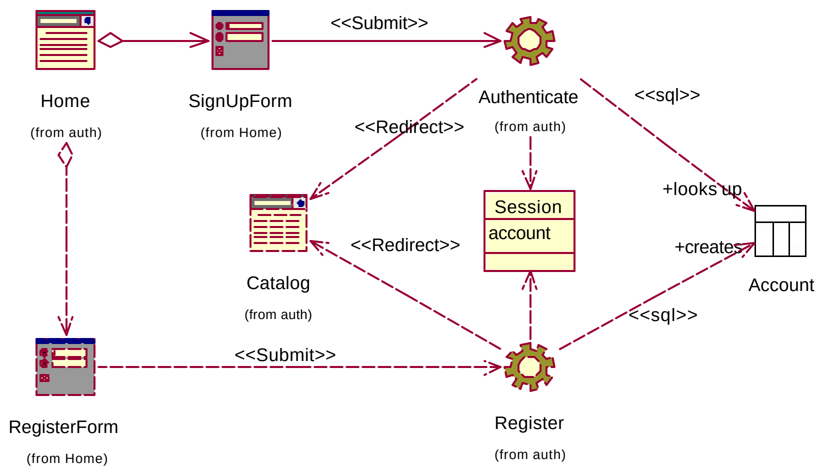

Figure 3: Sign up and Register class diagram.

Student Survey

A study was conducted to determine if there is a predictive relationship between teaching method and applications development skills. The study was based on two samples of 65 and 47 students, each drawn from our business student body. The treatment group with 65 students was exposed to the model-based approach described in Section 2, while the controlled group with 47 students was taught using a traditional code-based approach described in Section 1. The two groups were taught by the same instructor.

In both groups, students were asked to develop a Web-based e-auction, called *eBid*, given a textual specification of the application. The system goal of *eBid* is as follows.

eBid acts as a forum for registered members, through which Internet users can log-on and assume the role of either bidder or seller. Sellers post items. When a seller enters an item to be auctioned, the seller provides a description of the item, keywords, initial price, date and personal information. This data is used to create the item profile seen by potential buyers. As a bidder, a user may search the site for availability of the item s/he is seeking, view the current bidding activity and place a bid. The flows of the use case scenarios are as follows.

Register as Member

1. To sign up as a member a customer needs to fill out the Personal Contact Information form.
2. A confirmation message is sent to the customer's email address.
3. The customer goes to their email program and clicks the blue underlined link to complete their registration.

Bid on Item

The bidder must be registered as a member

1. The use case starts when the bidder finds the item s/he is interested in by browsing the catalog of available items.
2. The bidder enters the bid in the bid box.
3. The bidder enters his/her email and password and then clicks the "Place Bid" button.
4. If the bidder wins the auction, s/he is notified by email. Then, the bidder goes to their email program and clicks the underlined blue link in the eBid's message to make a credit card payment.

Register as Seller

1. The seller must be registered as a member.
2. To create an online payment account allowing bidders to pay by a credit card, the seller fills out the Create Account form indicating a bank account.

Sell Item

1. Click on the Sell button at the top of any page. This will bring you to Sell Your Item form.
2. Choose the category under which to list the item (e.g., antiques? coins?).
3. Enter member's User ID and Password.
4. The seller enters the item title, description and photos and how long the seller wants the auction to last.
5. The seller enters the start price and the reserve price.

Glossary and Business Rules

Personal Contact Information form: The required fields for the form are email, confirm email, password, confirm password, street address, city, state, zip code, and telephone.
Create Account form: The required fields bank account, bank routing number, account holder.
Start price: Bidding for an auction will start at this price. The start price is used to generate bidding activity.
Catalog: A nonempty set of items.
Reserve price: The reserve price is a hidden amount that only the seller knows. This is the lowest price the seller is willing to sell the item for.
Business rule BR1: Reserve price should be higher than the start price.

The survey tasks given to the students in the two groups are shown in Figure 4. Part of the intended outcome from Task 1 is shown in Figure 5. We take into account the fact that there is not only one right answer. There are five important subsystems in the Web application that had to be taken into consideration: (1) `Register Member`; (2) `Authenticate Member`; (3) `Bid on Item`; (4) `Register Seller`; and (5) `Sell Item`. For task 1, a design is graded with 100%, if it realizes all of these five subsystems and shows how the subsystems interact. For task 2, an implementation of a server (asp) page is graded with 100% if its program logic complies entirely with the specification of the server page given in the answer to task 1. For task 2, how server-side JavaScript objects are used, how students combine html tags and server-side JavaScript scripts, how database queries are made, and how database query results are processed are taken into consideration. Then, the average of the two grades is taken to get the final grade for a student.

Task 1 (Develop the software architecture for eBid).

List the names of and/or depict graphically the html pages, asp pages, and database tables of eBid. Describe in one sentence the responsibility (goal) of each page and database table. For each page show the other pages and/or database tables it interacts with.

Task 2 (Implement an asp page in JavaScript)

Choose one of the asp pages you created for Task 1 and implement it in server-side JavaScript. Show all html pages that submit or link to this asp page. Show also the asp pages that forward to this asp page, if any.

Figure 4: Survey tasks.

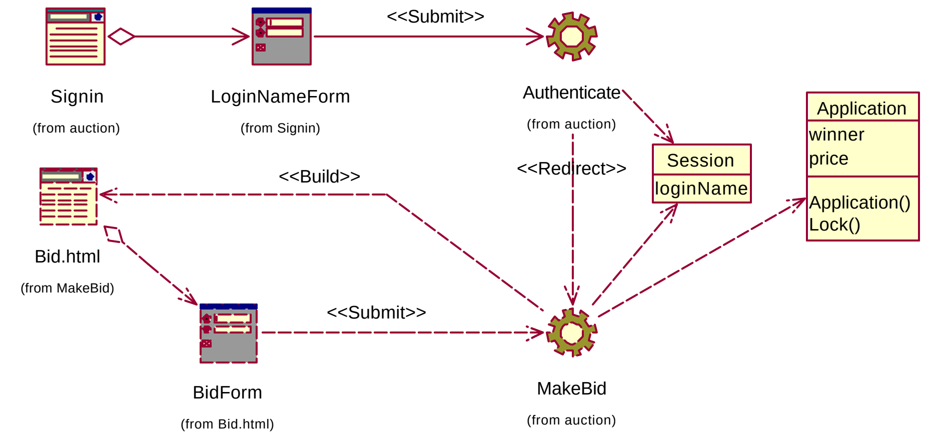

Figure 5: Auction class diagram.

Statistical Analysis

The study described in the previous section was designed to measure the strength of the relationship between teaching method and applications development skills. The question arose whether other factors, such as student abilities or academic standing, would have any effect on applications development skills. Four key factors were considered: student scores on the verbal section of the scholastic achievement test (vSAT), student scores on the mathematical section of the scholastic achievement test (mSAT), student grade point average (GPA), and student rank in class (Rank). vSAT and mSAT are discrete variables in the range of 410 to 790, GPA is a continuous variable in the range of 1.5 to 4.0, and Rank is a discrete variable in the range of 0 to 1.

Factor	Mean	Std. dev.	SE	95% Confidence
vSAT	549.68	51.69	2.25	545.27 ÷ 554.09
mSAT	51.69	63.66	2.77	565.96 ÷ 576.84
GPA	2.97	.55	.02	2.92 ÷ 3.02
Rank	.5	.29	–	–

Table 3: Descriptive statistics for the student population.

Factor	Treatment group Mean	Std. dev.	Controlled group Mean	Std. dev.
vSAT	545.85	51.69	536.6	58.69
mSAT	553.72	67.73	568.34	64.31
GPA	2.82	0.52	2.92	.48
Rank	0.61	0.27	.57	.26
Appl.dev.skills	76.35	23.17	62.98	19.73

Table 4: Descriptive statistics for the treatment and controlled groups.

Applications development skills are measured on a 100-point discrete scale. Descriptive statistics for the factors captured in this research are set out in Tables 3 and 4.

First, the student population was studied for normality. Then, the treatment and the controlled groups were proven representative samples of the student population. Next, whether the treatment and the controlled groups were of similar ability was investigated. Finally, a multiple regression analysis was undertaken to determine if there is a statistically significant relationship between applications development skills and teaching method, and to assess the impact of vSAT, mSAT, GPA and Rank on applications development skills.

Evaluating the Student Population for Normality

First, we tested whether the population's vSAT, mSAT, and GPA scores, each conforms to a normal distribution. We want to determine whether the set of vSAT frequencies, f_o^{vSAT}, matches a set of expected vSAT frequencies, f_e^{vSAT}, that conforms to a normal distribution. The student population mean vSAT is 549.68, and standard deviation is 65.44. The data was partitioned into 23 categories. We combined the first three and the last two data partitions, since there were cells having less than six values. Part of the frequency distribution is shown in Table 5. Column 2 shows the observed frequencies and column 4 the expected frequencies. The null and the alternative hypotheses are:

No	BinRange	f_o^{vSAT}	%	f_e^{vSAT}
1	440	27	5.09%	25
2	477	34	6.42%	34
3	490	46	8.70&	37
	…	…	…	…
20	791	20	3.77%	12

Table 5: vSAT frequency distribution of the student population.

$H_0$1: The vSAT frequency distribution follows a normal distribution (μ=549.68, σ=65.44).
$H_1$1: The vSAT frequency distribution does not follow a normal distribution.

This hypothesis was tested using χ^2-goodness-of-fit test. Since the mean and the standard deviation of the hypothetical normal distribution are known, the degrees of freedom is equal to the number of data partitions minus one, 19. The critical value of χ^2 at the .05 significance level is 30.14. The computed χ^2 is 14.22, indicating that the null hypothesis cannot be rejected. We conclude that the vSAT distribution follows a normal distribution.

We ran analogous tests of hypothesis for mSAT and GPA, the other two important factors whose distributions we do not know. In both cases, the null hypotheses that the frequency distributions follow normal distributions were not rejected. To sum up, we determined that the variables (vSAT, mSAT, and GPA) describing the student population follow normal distributions. To get a deeper insight for each variable of interest, we depicted its normal probability plot and inspected it for deviations from the expected trajectory. The plot in Figure 6 is in support of the analysis made above.

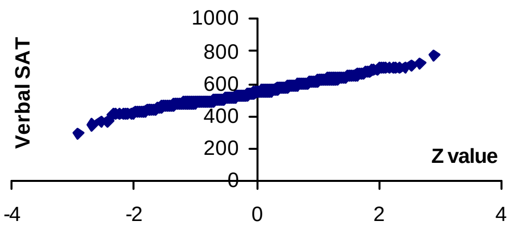

Figure 6: vSAT normal probability plot for the student population.

Comparing the Treatment and the Controlled Groups to the Student Population

Next, we prove that the treatment and the controlled groups are representative samples of the student population. We carried out tests to compare the mean and the standard deviation of each sample's vSAT, mSAT, GPA and Rank with the corresponding population ones. For all tests, the sample size is large (65 or 47), the population mean and standard deviation are known, the population is considered normally distributed, and the level of significance is .05.

To determine if there is any difference in the verbal SAT scores between the treatment group and the student population, we defined the following null and alternative hypotheses:

$H_0$2: There is no difference between the mean vSAT scores for the treatment group and for the student population.

$H_1$2: The treatment group mean vSAT is different from the population mean vSAT.

The hypothesis was tested with a two-tailed *Z* test. The test revealed a *z* statistic=–.06 (*p*-value>.55), indicating that there is no significant deference between the sample mean and the population mean, thus the null hypothesis was not rejected.

A two-tailed ANOVA was run to test if the variance in vSAT of the student population is equal to the variance in vSAT of the treatment group. The null and the alternative hypotheses are expressed as follows:

$H_0$3: There is no difference in the variation in the vSAT scores for the treatment group and the student population ($s_1^2=s_2^2$).

$H_1$3: The treatment group variance is different from the population variance ($s_1^2\neq s_2^2$).

The alternative hypothesis does not state a direction. The *F* distribution employed to test this hypothesis revealed an *F* statistic=0.73 (*p*-value=0.08), indicating no significant difference between the two variances, therefore the null hypothesis was not rejected. The outcomes from all *Z* and *F* tests for the treatment group and the student population are reported in Table 6 and Table 7. It is important to note that the *Z* tests for GPA and Rank are one-tailed tests (Table 6 columns "GPA" and "Rank"). The alternative hypotheses for GPA and Rank state that students in the treatment group do not outperform students in the population.

The tests comparing the parameters of the controlled group and the student population are not included since they are identical to the tests for the treatment group and the student population. The results from the tests lead to the conclusion that the two samples are representative of the student population.

	vSAT	mSAT	GPA	Rank
Sample size	65	65	65	65
Sample mean `X	545.85	557.72	2.82	.61
Sample std. dev. S	60.4	67.63	0.52	.27
Population mean m	549.68	571.4	2.97	.5
Population std. dev. s	51.69	63.66	0.55	.29
Significance level a	.05	.05	.05	.05
$H_0$2	$\overline{X}$=549.68	$\overline{X}$=571.4	$\overline{X}$=2.97	$\overline{X}$=.5
$H_1$2	X≠549.68	X≠571.4	X<2.97	X>.5
Critical value	±1.96	±1.96	-1.64	1.64
Z test statistic	-.6	-1.73	-2.2	3.06
P-value	.55	.08	.01	1.0
$H_0$2 *decision*	Accept	Accept	Reject	Reject

Table 6: *Z* tests for difference in means between the treatment group and the population.

	vSAT	mSAT	GPA	Rank
Population size (df)	529(528)	529(528)	529(528)	529(528)
Std. deviation s_1	51.69	63.66	0.55	0.29
Controlled group size (df)	65(64)	65(64)	65(64)	65(64)
Std. deviation s_2	60.4	67.73	0.52	0.27
$H_0$3	$s_1^2=s_2^2$	$s_1^2=s_2^2$	$s_1^2=s_2^2$	$s_1^2=s_2^2$
$H_1$3	$s_1^{2\,1}s_2^2$	$s_1^{2\,1}s_2^2$	$s_1^{2\,1}s_2^2$	$s_1^{2\,1}s_2^2$
Lower critical value	.71	.71	.71	.71
Upper critical value	1.49	1.49	1.48	1.49
F-test statistic	.73	.88	1.12	1.15
P-value	.08	.47	.59	.48
$H_0$3 *decision*	Accept	Accept	Accept	Accept

Table 7: *F* tests for variance for the treatment group and the student population.

Comparing the Treatment and Controlled Groups

Our next objective is to compare the scholastic abilities of the two sample groups. We are interested in discovering whether the mean vSAT for the treatment group is different from the mean vSAT for the controlled group. The null and alternative hypotheses are:

> $H_0$4: There is no difference in the mean vSAT scores for the two groups.
> $H_1$4: The mean vSAT scores for the two groups are different.

We conducted a two-tailed *t* test to test the null hypothesis $H_0$4. At the .05 significance level, the critical values are ±1.98. The computed *t* test statistic=.81 (*p*-value=.42) is a strong evidence that there is no difference between the two means, thus the null hypothesis cannot be rejected.

In addition to comparing the central tendencies in the abilities of the two groups, we want to compare their variation. This is expressed by the following null and alternative hypotheses:

> $H_0$5: There is no difference in the variation in the vSAT scores for the two samples ($s_1^2=s_2^2$).
> $H_1$5: One of the groups has more variation than the other ($s_1^2 \neq s_2^2$).

We ran a two-tailed *F* test for variances. The calculated *F*-test statistic=1.05 (α=.05, *p*-value=.85) strongly indicates that the null hypothesis cannot be rejected. The tests comparing the two samples are summarized in Table 8 and 9. Based on their outcome, we conclude that the two sample groups are of similar abilities.

	vSAT	mSAT	GPA	Rank
Treatment group size (df)	65(64)	65(64)	65(64)	65(64)
`X_1	545.85	553.72	2.82	0.61
s_1	60.4	67.63	0.52	0.27
Controlled group size (df)	47(46)	47(46)	47(46)	47(46)
`X_2	536.6	568.34	2.92	0.57
s_2	58.69	64.31	.48	.26
$H_0$4	`$X_1$ = `X_2	`$X_1$ = `X_2	`$X_1$ = `X_2	`$X_1$ = `X_2
$H_1$4	`$X_1$ ¹ `X_2	`$X_1$ ¹ `X_2	`$X_1$ ¹ `X_2	`$X_1$ ¹ `X_2
Lower critical value	-1.98	-1.98	-1.98	-1.98
Upper critical value	1.98	1.98	1.98	1.98
t-test sta tistic	.81	-1.4	-1.04	.79
P-value	.42	.16	.3	.43
$H_0$4 decision	Accept	Accept	Accept	Accept

Table 8: *t* tests comparing the means of the treatment and controlled groups.

	vSAT	mSAT	GPA	Rank
Treatment group size (df)	65(64)	65(64)	65(64)	65(64)
s_1	60.4	67.63	.52	.27
Controlled group size (df)	47(46)	47(46)	47(46)	47(46)
s_2	58.69	64.31	.48	.26
$H_0$5	$s_1=s_2$	$s_1=s_2$	$s_1=s_2$	$s_1=s_2$
$H_1$5	$s_1$¹s_2	$s_1$¹s_2	$s_1$¹s_2	$s_1$¹s_2
Lower critical value	.59	.59	.59	.59
Upper critical value	1.74	1.74	1.74	1.74
F-test statistic	1.06	1.11	1.17	1.08
P-value	.85	.72	.57	.8
$H_0$5 decision	Accept	Accept	Accept	Accept

Table 9: *F* tests comparing the variances of the treatment and controlled groups.

Multiple Regression Model

Multiple regression was run to determine if there is any predictive relationship between teaching method, vSAT, mSAT, GPA, and Rank and applications development skills. The multiple regression model we developed is defined as:

Applications development skills = *a* + b_1(Teaching method) + b_2(vSAT) + b_3(mSAT) + b_4(GPA) + b_5(Rank)

Teaching method is a dummy variable taking two values: 1, if the model-based approach was employed; and 0, otherwise. The overall regression statistics is set out in Table 10. We wish to verify that the coefficient of multiple determination, R^2, is statistically significant, to evaluate the individual net regression coefficients to see which are not equal to zero, and to verify that the regression assumptions are met.

Multiple R	.434
R2	.189
Adjusted R2	.15
Standard Error	20.915
Observations	112

Table 10: Regression statistics.

	Df	SS	MS	F	Prob. F
Regression	5	10770.99	2154.2	4.92	.0004
Residual	106	46366.5	437.42		
Total	111	57137.49			

Table 11: Analysis of variance table.

In the test we conducted, there are 112 observations, so the total degrees of freedom is 111. The number of independent variables is 5, so the degrees of freedom in the "Residual" row of the analysis of variance table is 106, see Table 11.

Global Test: Testing Whether the Multiple Regression Model is Valid

The ability of the independent variables—teaching method, vSAT, mSAT, GPA, and Rank—to explain the behavior of the dependent variable applications development skills was tested with a global test. The null and alternative hypotheses are:

$H_0$6: All independent variables have zero coefficients ($b_1 = b_2 = b_3 = b_4 = b_5 = 0$).
$H_1$6: Not all coefficients are zero.

When application development skills was regressed on these five variables, the equation was statistically significant (F=4.92, p-value=.000), therefore the alternative hypothesis is accepted.

Evaluating Individual Regression Coefficients

We found out that some, but not necessary all, of the regression coefficients are not equal to zero and thus useful for prediction. Next, we test the variables individually to determine which regression coefficients may be zero and which are not. The null and alternative hypotheses for teaching method are:

$H_0$7: Teaching method is not statistically significant in predicting applications development skills, i.e. the coefficient for teaching method, b_1, is zero.
$H_1$7: Teaching method is of value in predicting applications development skills ($b_1 \neq 0$).

The test statistic is the t distribution with 106 degrees of freedom. Teaching method was found significant (t=3.31, p-value=.001) when all other factors were held constant.

Evaluating the individual regression coefficients with PHStat produced the output shown in Table 12. When the regression coefficients for the individual predictors were examined, only teaching method was statistically significant.

	Coefficients	SE	T stat.	P-value	Lower 95%	Upper 95%
Intercept	-59.12	102.34	-.58	0.565	-262.02	143.78
Teaching method	13.83	4.18	3.31	0.001	5.54	22.11
vSAT	.03	.03	1.01	0.317	-.03	.1
mSAT	.02	.04	.58	0.566	-.05	.1
GPA	25.45	26.66	.95	0.342	-27.39	78.3
Rank	31.04	49.66	.63	0.533	-67.42	129.5

Table 12: Evaluating individual regression coefficients.

Verifying the Regression Assumptions

To check the residuals distribution for normality, the residuals were plotted against the Z values. The plot shown in Figure 7 confirms that the residuals follow a normal distribution. To check for homoscedasticity, the residuals were plotted against the fitted values of the dependent variable. The plot in Figure 8 confirms that the homoscedasticity assumption is met.

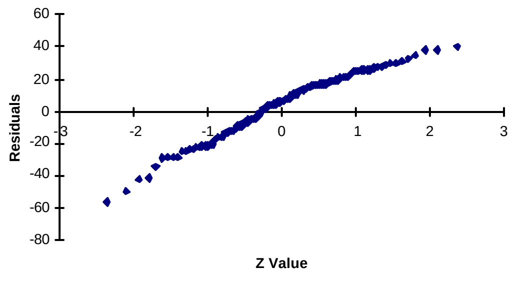

Figure 7: Normal probability plot.

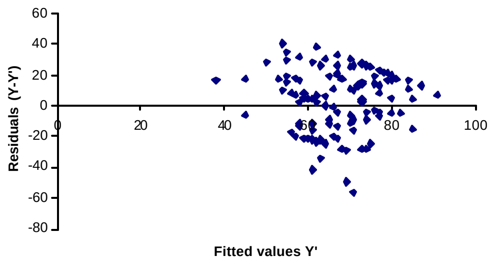

Figure 8: Homoscedasticity plot.

Interpretation of the Survey Results and Qualitative Observations

In this section, we discuss the results from the study and present important observations about deficiencies encountered in the survey papers of both groups.

A different cross-section of the study results is shown in Table 13. Among those students in the treatment group, 90% designed correct class diagrams. Seventy one percent of this group converted one of the client-server interactions to code, easily and generally fault-free. Only 29% of the students in the controlled group wrote text descriptions, and almost the same number of students, 27%, created correct diagrams that show important structural elements of the e-commerce application.

	Treatment group	**Controlled group**
Class diagram models	90%	7%
Textual description	5%	29%
Non-satisfactory models	5%	12%
Correct Diagrams, Storyboards	0%	27%
Incorrect diagrams	0%	12%
Process oriented diagrams	0%	5%
Isolated client-server interactions	0%	7%
Satisfactory implementation	71%	52%

Table 13: Detailed view of the survey.

Several deficiencies emerging in the survey papers of the controlled group, which are not present in those of the treatment group, are summarized in Table 14. Findings about the group exposed to modeling are listed in Table 15.

It is interesting to note that the combined number of students in the controlled group who attempted to create diagrams (whether correct or incorrect), 58%, is far greater than those who wrote only a text description, 29%. We could explain this fact with the need students feel for a notation to define the structure of the designed system.

#	Deficiency
1.	Process-oriented (behavioral) descriptions where structural ones are expected.
2.	Use of graphs with free text nodes.
3.	The types of the interactions between the structural elements are not shown (in UML parlance-no stereotyped associations).
4.	Database elements, e.g. tables, are either missing or if present are considered active objects building HTML pages or forwarding to asp pages.
5.	Isolated client-server interactions. Interactions among subsystems rarely shown.
6.	The server-side object model not well understood. Session and Application objects are missing from the system descriptions.
7.	Failure to identify important application logic and business rules
8.	Failure to allocate application logic and business rules to structural elements.
9.	Inappropriate reuse of previously seen elements.
10.	Lack of the notion of subsystem, package in UML.
11.	Overemphasizing the role of UI at the expense of business logic
12.	Students cannot distinguish well between the major responsibilities of client and server pages.

Table 14: Deficiencies in the works of the group exposed to a code-based approach.

#	Deficiency
1.	Students tend to gloss over the association stereotypes.
2.	The number of students who have developed correct CD models exceeds by 19% the number of students who have written correct implementations.
3.	Not a single student has developed a correct implementation without creating a correct model.

Table 15: Important observations from the works of the group exposed to modeling.

Turning to the deficiencies listed in Table 14, it should first be noted that item 2 is not necessarily a deficiency. It merely illustrates the need for a graphical modeling language, while at the same time demonstrating how intuitive and appealing graphs are to learners. Although students in the controlled group have not been taught how to stereotype associations, it was expected that at least a few of them would be able to indicate the types of relationships existing between the pages forming the system. Item 5 attests to the fact that students in this group cannot define the high-level system organization. Item 6 enhances further the confidence in the conclusion from the previous topic. It is common in Web-based applications for subsystems to communicate indirectly through the `Session` and `Application` objects. One reason students do not have thorough understanding of the server-side object model is the lack of time available to practice it. A second reason, probably more significant than the first, is the lack of skills in application logic modeling. The observations presented in items 7 and 8 (directly related to item 6) are inevitable consequences of the low-level code approach used in that class. Item 9 shows a lack of synthesis skills. Since use case scenarios are not covered in this class, students cannot decompose logically a system into subsystems as suggested by item 10.

This leads to the conclusion that the analysis skills developed by the students are not at a satisfactory level. Learners in the controlled group tend to overemphasize the role of GUI at the expense of business logic (item 11). This result springs from teaching client-side scripting. It is for the same reason that students do not distinguish well between the major responsibilities of client pages, server pages, and database tables (item 12). Teaching client-side form validation in JavaScript is the main culprit for blurring the boundaries of these concepts. The Boundary-Controller-Entity design pattern (Jacobson et al., 1992) is not acquired by the learners. Most of the deficiencies described in Table 14 stem from the inadequacy of natural language as a modeling and design notation.

Turning to Table 15, we noted that students exposed to modeling do not realize the importance of labeling and stereotyping associations in their models. Precision is of utmost importance because the basic advantage of modeling is the possibility to capture subtle nuances before they have turned into bugs difficult to track down and debug in the vast code view. Item 2 in Table 15 suggests that business students exposed to modeling could acquire good understanding of how Web-based information systems function without possessing comprehensive coding skills. Item 3 in Table 15 is the contra-positive of item 2. No coding skills are possible without modeling skills.

The data suggests that students in the controlled group do not have an in-depth understanding of the notion of component interface (Table 14, items 3, 4, 5, 10) and in addition lack application modeling skills (Table 14, items 4, 6, 7, 11, 12). Not being able to define precisely the responsibility of a server page and its collaborations results in a poor implementation of that server page. This would account for the inferior results on implementation, 52%, obtained by the controlled group as compared to the results, 71%, obtained by the treatment group. This is particularly interesting, since one would have expected the controlled group to show better results on implementation than the treatment group because of the longer time they spent on topics related to programming. However, as these findings suggest, the programming skills become immaterial if students cannot clearly define what they are implementing.

It was again quite interesting to find several correct class diagrams in the works of the controlled group. Further exploration revealed that there had been a cross-class transfer (root-grass transfer) of expertise. While working on similar projects, students from the treatment group, unable to think in low-level (code) terms had taught their colleagues from the controlled group how to think abstractly and how to specify components and interfaces. This offers strong support for the initial hypothesis put forth in this paper. Students exposed to modeling felt lost without models. This fact further corroborates the proposition that a model-based approach is intuitive and natural, and should therefore be taught at an early stage in the curriculum.

Human mind actively creates what it knows (Gardner, 1991). The model-based approach is more congenial to the creating of strong learner-centered environment that promotes in-depth learning and understanding essential to problem-solving. In the code-based approach, on the other hand, students encounter predominantly well-defined problems, which have right answers. Since this type of problems do not reflect the ambiguity of real life, they do not enhance students' knowledge as to when and how the acquired skills should be used, that is students tackling well-defined problems fail to develop analysis and synthesis skills. Programming problems require perfect performance. A misplaced comma or semicolon makes the whole effort futile. Errors in programming are perceived as failures. However, errors are an indispensable part of problem solving and therefore learning. As Louis Pasteur has powerfully observed, "If you shut your door to all errors truth will be shut out." In this respect, high-level models are more ill-defined than code-base ones. There is usually more than one correct solution. Students are involved in lively discussions weighing alternative solutions (i.e. evaluation, the highest-order cognitive skill). The inhibition barrier of being perfect does not exist. Students are enabled to capitalize on mistakes rather than desperately trying to avoid mistakes. The environment fostered is one of "honesty without fear" (Deming, 1986).

In the model-oriented approach, skills such as analysis, design and implementation are taught as an integrated whole. Programming is not learned in isolation. In contrast, in the code-based approach, students do not have the opportunity to apply their programming skills in different settings. Research shows that without the opportunity to use knowledge to achieve a goal, such knowledge is recalled only in the context in which it has been learned (Huba & Freed, 2000). This might help explain why students who have spent more time doing programming are unable to use their skills effectively when tackling the survey task. We may well say with Huba that their knowledge is inert (Huba & Freed, 2000).

Conclusion

This paper set out to present the results of a survey evaluating the skills acquired by two groups of business students in a course on E-business applications development. Two different approaches to teaching this course have been analyzed. In teaching this course, the first group of students had been exposed to a model-based approach, while the second one had been exposed to a traditional code-based approach.

Compared with the code-based approach, in which modeling was not used at all, the proposed model-based approach not only allows for a more comprehensive coverage of material in e-commerce models, Web-based e-business applications, and modeling business logic, but also proves conducive to bringing students' cognitive skills onto a higher level of complexity. The upshot is a qualitative breakthrough in students' performance.

The quantitative results show that teaching using a model-based approach has brought about tangible changes in students' modeling and programming skills. The enhanced growth in critical thinking and synthesis skills is attributed to the interrelatedness and interdependence of modeling and abstraction whose conjoint teaching and practice proves very fruitful.

Students who have been exposed to modeling outperform not only in high-level modeling skills but also in programming skills. The difference of almost 20% in programming skills in favor of these students confirms our hypothesis that interface specification and component modeling must precede implementation.

Quantitative and qualitative results obtained from the survey call for a change in approach in teaching E-business applications development to business students. The confidence that abstraction can be taught before programming, and very successfully at that, springs from experiential results. It is evident, both from the conducted experiment and observations gleaned throughout this research, that students become receptive to the potential and merits of abstraction, which they get to use methodically and more confidently in their work.

Knowledge is bound to remain quiescent unless it is something active, discriminating and critical. The proposed model-based approach has proved instrumental in creating this much needed new condition for learning, without which inert knowledge can never become active.

References

ASP home page. (2003). http://msdn.microsoft.com.

Baber, B. (1987). *The spine of software: Designing provably correct software - theory and practice*. Chichester: John Wiley & Sons.

B. S. Bloom, B. S. (ed.) (1956). *Taxonomy of educational objectives: Book 1 Cognitive domain*. London: Longman.

Booch, G., Rumbaugh, J. and Jacobson, I. (1999). *The unified modeling language*. Addison-Wesley.

Cohen, E. (1999). Reconceptualizing information systems as a field of transdiscipline informing science. *Journal of Computing and Information Technology, 7* (3), pp. 213-219.

Conallen, J. (1999, Oct.). Modeling web application architecture with UML. *Communications of the ACM*, 42(10), pp. 63-70.

Davis, M. (1988, Sept.). A comparison of techniques for the specification of external system behavior. *Communications of the ACM*, 31(9).

Deming, W. E. (1986). *Out of the crisis*. Cambridge, MA: MIT Center for Advanced Engineering Study.

Gamma, E., Helm, R., Johnson, R. and Vlissides, J. (1994). *Design patterns. Elements of reusable object-oriented software*. Addison Wesley Professonal Computing Series.

Gardner, H. (1991). *The unschooled mind: How children think and how schools should teach*. New York: Basic Books.

Gersting, J., Henderson, P. B., Machanick, P. and Patt, Y. N. (2001, Feb.). Programming skills as an outcome in panel discussion: Programming Early Considered Harmful. *Proceedings of SIGCSE-2001*. Charlotte, NC.

Harel, D. (1987). Statecharts: A visual formalism for complex systems . *Science of Computer Programming*, 8, New York.

Huba, M. E. and Freed, J. E. (2000). *Learner-centered assessment on college campuses*. Allyn and Bacon.

Jacobson, I., Christerson, M., Jonsson, P., and Overgaard, G. (1992). *Object-oriented software engineering: A use case driven approach*. Addison-Wesley.

Jacobson, I., Booch, G. and Rumbaugh, J. (1999). *The unified software development process*. Addison-Wesley.

Johnson, R. A. (2000, Oct.). The ups and downs of object-oriented systems development. *Communications of the ACM*, 43(10), pp. 69-73.

Medvidovic, M., Gruenbacher, P., Egyed, A. and Boehem, B. W. (2002). Modeling software architectures in the UML. *ACM Transactions on Software Engineering and Methodology*, 11(1).

Mellor, S. J. and Balcer, M. J. (2002). *Executable UML: A foundation for model driven architecture*. Addison Wesley Professional.

Nuseibeh, B. A. and Easterbrook, S. M. (2000). Requirements engineering: A roadmap, *22nd Int'l Conference on Software Engineering*, Limerick, Ireland.

Roussev, B. (2003, June). Teaching introduction to programming as part of the IS component of the business curriculum. To appear in *Informing Science + Information Technology Education Joint Conference*, Pori, Finland.

ScriptEase (2003). http://www.nombas.com, JavaScript interpreter home page.

Shannon C. E., and Weaver, W. (1949). *The mathematical theory of communications*. Urbana: University of Illinois Press.

Shaw, M. (2000). Software engineering education: A roadmap. *22nd Int'l Conference on Software Engineering*, Limerick, Ireland.

Biography

Borislav Roussev is an Assistant Professor of Information Systems at Susquehanna University. He was educated in Bulgaria, and previously was a faculty member in higher education in South Africa. Dr. Roussev's research interests are in the areas of object-oriented modeling and software engineering.