

Making Failure The Mother of Success

Xiang Fu, Kai Qian, Kent Palmer, Boris Peltsverger, Brian Campbell, Billy Lim, and Paul Vogt
 Xiang.Fu@hofstra.edu, kqian@spsu.edu, kpalmer@goshen.edu, plz@canes.gsw.edu, campbell@canes.gsw.edu,
 bllim@ilstu.edu, wpvogt@ilstu.edu

Abstract - Do students really welcome failure, the mother of success? This paper reports the implementation of a trial-and-failure learning strategy, in both entry and senior level computer science classes. The idea is simple: given a sophisticated course project, let students try project submissions as many times as they want before the project deadline. For each submission, a thorough inspection is performed by an automated grading system called APOGEE. A student project has to accomplish not only the functional requirements but also many desired quality attributes such as robustness and security. In 2009, the trial-and-failure strategy has been adopted by four universities in five class sections. We report some interesting observations from student survey results, e.g., one can find out if factors like students' positive experiences of programming, choice of programming language, years of working experience, and instructors are predictive variables for positive attitudes toward the trial-and-failure learning experience as a whole.

Index Terms - Automated Grading, Trial-and-Failure, Automated Testing, Active Learning.

INTRODUCTION

It has been widely accepted that if students can be actively engaged in a learning activity, they are more likely to achieve the learning goals. Active learning [1] and cooperative learning strategies [2] have been practiced in classrooms for decades. Group discussion, think pair share, and short written exercises [3] are regarded by most instructors as effective approaches in classrooms.

How do we bring more fun and challenges to students outside the classrooms? We present *TAF*, a trial-and-failure learning strategy, for computer science courses. The basic idea of *TAF* is simple: given a sophisticated course project, let students try project submission as many times as they want before the project deadline. For each submission, a thorough inspection is performed. A student project should accomplish not only the functional requirements but also many desired quality attributes such as robustness and security (e.g., it should not be crashed by a malicious data payload). By helping students make multiple revisions of their projects, *TAF* deepens students understanding of the learning subject using a cyclic improvement model. It helps to promote at least five of the seven good practices of active learning suggested by [4,5], e.g., giving prompt feedback and communicating high expectations.

To provide 24x7 service on project assessment and guidance, an automated project grading and tutoring tool is necessary. *TAF* relies on *APOGEE* [6,7,8] (Automated Project Grading and Rapid Feedback System), for providing the much needed instant feedback to students. *APOGEE* extends the application of automated GUI (Graphic User Interface) testing technique [9,10]. It goes beyond many automated grading tools in CS1/2 that handle text-mode programs only. The project is supported by NSF-DUE-CCLI grants 0836859, 0837275, and 0837020.

In 2009, *APOGEE* has been adopted by Hofstra University, Southern Polytechnic State University (SPSU), Wingate University, and Georgia Southwestern State University (GSW) in five class sections related to web application development. Based on the 42 copies of post-survey collected, 83% of students like being able to submit projects multiple times. It is interesting to note that the responses from different universities are quite diversified. A detailed analysis is provided in Section 4. We report our activities, experiments, success and failure stories with the model.

The rest of the paper is organized as follows. Sections 2 and 3 briefly describe the background information: the *APOGEE* tool and the related courseware. Section 4 presents the evaluation results. Section 5 discusses faculty responses. Section 6 concludes.

TECHNICAL BACKGROUND

APOGEE subsumes our prior work *ProtoAPOGEE* [6]. It consists of five major components: (1) a course management module, (2) a project specification tool for instructors to create test cases, (3) a grading engine that invokes the *WatiN* [10] library for evaluating student projects, (4) a back-end database that stores both test cases and grading report, and (5) a project report generator which is able to display failed test cases using both static and dynamic animation approaches. At this moment, *APOGEE* grades web application projects only.

1. Project Specification Authoring Tool

To provide instant, fair, and consistent grading to students, a collection of test cases have to be prepared in advance by instructors. In *APOGEE*, a GUI editor is provided for conveniently composing testing actions. Figure 2 shows a part of the GUI editor for specifying test cases.

A test case consists of a sequence of test actions. There are two categories of actions: (a) *control manipulation actions*, which simulate human testers and manipulate the

GUI controls in a web application; and (b) *verification actions*, which validate the logical correctness of the information generated by the system under test.

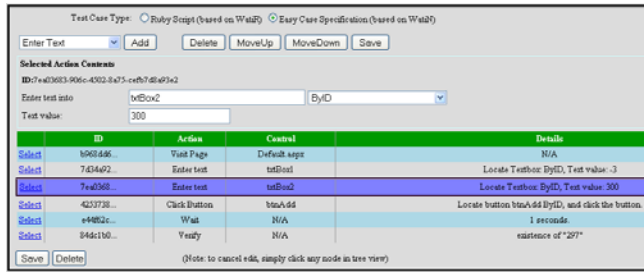


FIGURE 1
TEST CASE SPECIFICATION

For example, the test case in Figure 1 validates the correctness of a simple arithmetic calculator implemented as a web application. It visits the web page being tested, enters data (-3 and 300) into textboxes, clicks the submit button and verifies if a submission generates 297 as the result.

To manually specify the collection of test cases can be time consuming. To alleviate the burden on faculty, a collection of pre-compiled APOGEE project packs are available at [8]. Instructors can easily load packs into any APOGEE server instance, and then customize the grading methods if necessary. It is also possible to extract GUI interface using AWAT [11], however it has not been tested on large systems.

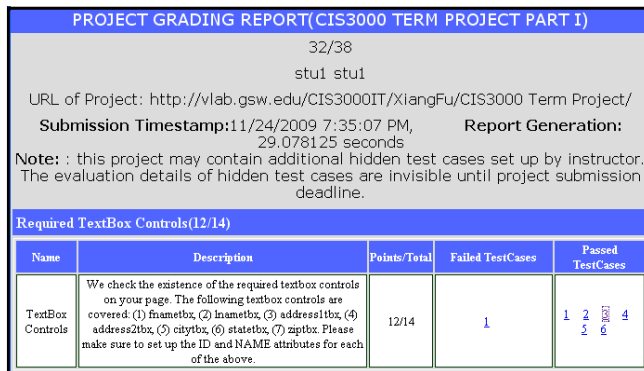


FIGURE 2
AN EXAMPLE OF PROJECT GRADING REPORT

II. Reporting Tool

A student can submit a given project as many times as he/she would like to APOGEE. A project report is generated instantly within one minute. Figure 2 shows a fragment of one sample project report.

A project report starts with a summary, e.g., the grade and timestamp. Then a report contains an itemized list of test cases. The running results of test cases are classified into two categories: the passed and the failed. For each test case, clicking the corresponding link redirects a viewer to the test case report. A sample report is given in Figure 3.

/10/\$25.00 ©2010 IEEE



FIGURE 3
AN EXAMPLE OF TEST CASE REPORT

A test case report can be displayed in two ways: printer friendly report and animation playback. The printer friendly report, as shown in Figure 3, consists of the detailed execution information of each test action. For each test action, there is a screenshot, description, and the running result. If a test action fails, additional information will be provided (e.g., the exception thrown by the testing engine).

A test case report could be displayed in animation by clicking the “View Test Case Demo” link. Watir [9] needs to be installed on the computer where the animation is displayed. The function is actually accomplished at the server side of APOGEE. From the serialized test case, APOGEE translates it to a complete Watir script so that the failed test is animated on the client computer.

III. Test Engine Implementation

APOGEE’s test engine is built upon WatiN [10], a unit testing library for web applications. Given a test case, the test engine starts a browser and then executes the test actions one by one. After the completion of each test action, a screenshot is taken and saved to the database. APOGEE [8] uses a distributed architectural design. The test engine is separated from the web portal of APOGEE and works in multi-process mode. Inter-process data exchange is accomplished via a shared file system. Thus, the time efficiency is greatly enhanced by improving the multi-threading granularity. The capacity of an APOGEE instance on an average 1.5 GHz PC server allows 10 concurrent grading jobs. One APOGEE instance is sufficient to support three concurrent lab sessions, each with 15 students.

CURRICULUM DESIGN

To broaden the impact of TAF and APOGEE, a collection of project-oriented course modules, called APOGEE packs, are developed and available at [8]. They address a wide variety of subjects, e.g., JavaScript programming, component based web design, backend database, and user input sanitation in web applications. Each APOGEE Pack is self-contained and consists of the following components:

- ❖ A formal project specification, which describes the required functions and desired quality attributes.
- ❖ A sample partial implementation, which allows students to clearly understand the project requirements. In most cases, students can start with the GUI design of the sample implementation and concentrate on the logical design.
- ❖ A collection of pre-defined test cases. They are serialized into binary form and can be easily loaded into an APOGEE server instance. The grading criteria have been configured according to the project specification. However, instructors can conveniently customize them using the GUI editor with APOGEE.
- ❖ A complete solution to the project, which is accessible by instructors only.

In the following we introduce several case study examples of APOGEE pack. All of them are available at [8].

I. Client Side Scripting

The first example demonstrates the use of APOGEE in teaching user input validation at client side. We would like to emphasize the importance of building robust software. Test cases are designed to aim at crashing a student's project using a combination of malformed user input.

Project "Travel Reservation" requires a student to create an airline ticket registration form for a fictitious airline. Students are asked to prepare both the XHTML interface and the JavaScript code for generating an itinerary and receipt for a customer. The complete solution contains about 150 lines of XHTML code and 200 lines of JavaScript code.

The testing pack provides a complete examination of the project. It has the following three categories of test cases.

- ❖ Required controls: These are used for verifying required controls in the XHTML file.
- ❖ Receipt Generation Function: A project submission has to correctly calculate the total expenses based on the number of passengers, the cabin type, and the other pertinent trip information. The built-in test cases cover a wide variety of the combinations of preconditions that lead to different pricing options.
- ❖ User Input Validation: Test cases are developed to help students build robust user input validation. Examples include: format check of zip code, date and time, required information validation of name fields, and temporal relation between leaving and return date.

The user input validation results are rendered in the form of alert dialogs. APOGEE supports a sophisticated mechanism for plugging dialog interceptors, parsing the contents of a dialog, and analyzing the correctness of the user input validation code written by students.

II. Robustness and Security of Server Side Scripts

Attacks on the application layer, e.g., SQL injection [12] and Cross-Site Scripting [13], now account for over 70% of the attacks on the Internet. The "Freelance Bulletin" pack [8] is designed for increasing students' awareness of these attacks and the countermeasures. Students are given a sample implementation of an electronic bulletin board which allows anonymous users to post messages. They are asked to strengthen the web application by securing the database layer and server-side user input sanitation. A collection of test cases are designed to incorporate real-world attack strings so that the student project submissions are exhaustively inspected. Intuitive hints and guidance are provided by APOGEE when student submissions fail.

EVALUATION

In 2009, TAF and APOGEE were adopted at four institutions in five class sections, impacting 42 students. These classes range from entry level to senior level, all of which have significant amount of learning materials related to web programming. In the following, we discuss the evaluation efforts based on the student survey results.

I. Pre-Survey

A pre-survey is performed at the beginning of a semester. The purpose is to investigate the backgrounds of the students. We would like to assess if there is any positive correlation between students' background and their attitudes toward the trial-and-failure learning strategy. The information could later be used to guide instructors in customizing their practice of the TAF strategy.

The scale for most of the questions is 1 to 5, with strongly disagree denoted by 1 and strongly agree denoted by 5. Some questions (Q16c, Q16d) have a scale from 1 to 3. We list a selected subset of pre-survey questions in the following.

- ❖ Q1a. Work experience. (1 for yes, 0 for no).
- ❖ Q1b. Total work experiences in months
- ❖ Q7a. Experiences with C/C++? (1 for yes, 0 for no)
- ❖ Q7b. Experiences with Java? (1/0)
- ❖ Q7c. Experiences with C#? (1/0)
- ❖ Q11. I learn better by seeing examples worked out on the board. (Scale 1 to 5)
- ❖ Q12. I learn better by listening to lectures (1 to 5)
- ❖ Q13. I learn better personally doing through examples. (1 to 5)
- ❖ Q14. I learn better reading materials on my own. (1 to 5)
- ❖ Q15. I learn better by having a learning/tutorial system that provides feedback. (1 to 5)
- ❖ Q16c. Level of familiarity with SQL injection (1 to 3)
- ❖ Q16d. Level of familiarity with XSS attack (1 to 3)
- ❖ Q17. Are you comfortable with fixing vulnerabilities? (1/0)

- ❖ Q18. Identify the vulnerability of SQL injection in a given piece of ASP.Net C# or Java code. (1 for success, 0 for failure or no response)

Table I presents the survey data. Since there are only 3 effective responses from students at Wingate University, our data does not include Wingate statistics. Under each university, there are two columns: “Avg” and “Yes%.” The “Avg” column is the average of the student responses on that question. The “Yes%” measures the percentage of the students that respond with “Agree” or “Strongly Agree” in the 1 to 5 scale (or at least 2 in the 1 to 3 scale, or 1 in Q17 and Q18).

TABLE I
PRE-SURVEY RESULTS

ID	Hofstra		GSW		SPSU		ALL	
	Avg	Yes%	Avg	Yes%	Avg	Yes%	Avg	Yes%
Q1a	0.72	72%	0.57	57%	0.35	35%	0.52	52%
Q1b	23.5	-	30.4	-	3.6	-	17.7	-
Q7a	0.91	91%	0.36	36%	0.82	82%	0.69	69%
Q7b	0.73	73%	0.29	29%	0.88	88%	0.64	64%
Q7c	0.91	91%	0.71	71%	0.06	6%	0.5	50%
Q11	3.81	82%	4.5	85%	4.00	82%	4.12	83%
Q12	2.81	27%	3.4	50%	3.76	71%	3.40	52%
Q13	4.91	100%	4.4	86%	4.47	94%	4.54	93%
Q14	3.73	64%	3.0	36%	4.05	82%	3.61	62%
Q15	3.82	64%	3.9	71%	4.29	94%	4.02	79%
Q16c	0.54	36%	0.64	79%	0.65	65%	0.62	62%
Q16d	0.54	45%	0.24	24%	0.29	29%	0.33	31%
Q17	-	18%	-	12%	-	12%	-	13%
Q18	-	9%	-	0%	-	12%	-	11%

There are some interesting observations that can be made here. First, as shown by Q1 to Q7, the student body in the experiment is quite diversified. For example, the major programming language is C++ at Hofstra, Java at SPSU, and C# at GSW.

In this study, we are interested in measuring students’ preparation in web security. Questions Q16 to Q18 serve the purpose. Clearly, SQL injection enjoys a higher awareness - the number of students who know SQL injection is almost twice of those who know Cross-Site Scripting attack. However, only 1 in every 10 students who claim to know SQL injection can actually identify the vulnerability in real code.

Questions Q11 to Q15 evaluate students’ potential interests in the trial-and-failure learning strategy and the use of automated project grading tool. We have two interesting observations. First, students’ attitude toward independent learning is positively associated with their attitude toward automated project grading system (see Q14 and Q15). Second, surprisingly, students’ interest in regular class-room lecturing is also positively correlated (see Q12 and Q15). At Hofstra and GSW, students’ favoring of lecturing is substantially lower than SPSU, so is their response in using automated tutoring system.

TABLE II
POST-SURVEY RESULTS

ID	Hofstra		GSW		SPSU		ALL	
	Avg	Yes%	Avg	Yes%	Avg	Yes%	Avg	Yes%
Q1	4.2	90%	4.1	92%	3.9	74%	4.0	83%
Q2	3.7	44%	3.0	31%	3.7	63%	3.5	49%
Q3	4.1	78%	2.8	31%	4.0	78%	3.7	63%
Q4	3.7	44%	2.7	38%	3.7	74%	3.4	56%
Q5	4.4	89%	3.6	62%	4.1	74%	4.0	73%
Q6	3.8	78%	3.2	46%	3.8	79%	3.6	68%
Q7	3.3	56%	2.8	30%	3.7	68%	3.4	53%
Q8	3.6	62%	3.6	70%	3.9	73%	3.8	70%

II. Post-Survey

A post-survey is performed at the end of a semester. The following is a representative subset of the questions. All of them use the scale of 1 to 5:

- ❖ Q1. I like being able to submit a project several times and get feedback before getting a final grade for a project.
- ❖ Q2. The automated tutoring available through APOGEE helps me learn more.
- ❖ Q3. APOGEE’s grading of my projects has been consistent and fair.
- ❖ Q4. I am better able to learn from my mistakes using APOGEE than with traditional methods of grading projects.
- ❖ Q5. The fact that APOGEE’s feedback is instant has been helpful to me.
- ❖ Q6. The automated demos showing errors are helpful for learning internet technology.
- ❖ Q7. I learned more using APOGEE’s automated grading system than I would have using a traditional grading system on the same projects.
- ❖ Q8. Compared to other automated testing and grading software I have seen, APOGEE is better.

As shown in Table II, the student responses from the three participating institutions are mixed. Students have quite positive feedback on the trial-and-failure strategy, instant feedback, and fair grading provided by APOGEE: 83% of students like being able to submit project multiple times, 73% of students agree or strongly agree that APOGEE’s instant feedback has been helpful, 68% of students state that the automated demo of APOGEE is helpful for learning internet technology, and 70% of students acknowledge that APOGEE is somewhat better or much better than other grading software. However, a marginal majority of the students claim that APOGEE, as a tool, better helps to learn more than traditional teaching methods. There are also noticeable differences between the student feedback of SPSU and those from Hofstra and GSW. This may be explained by the diversity of student body and differences of pedagogical approaches taken by instructors. At GSW and Hofstra, students were not introduced to the APOGEE tool until the second last week of the semester;

while at SPSU, students had the access of the system from the beginning of the semester and completed at least two projects using APOGEE.

DISCUSSION

The participating faculty welcome APOGEE as an automated grading tool. All of them acknowledge that APOGEE saves great amount of time in project grading. They agree on the effectiveness of the TAF strategy as a pedagogical approach. They also raise many interesting topics for discussion, as listed below.

A sophisticated project usually has a complex specification. A detailed specification of required GUI interface is tedious to students in many cases. Students who do not follow the naming convention of web controls usually receive an “outrageous” low score at the beginning. There are two possible solutions to this problem: (1) asking students to start from a partial solution with all GUI interface already defined, and (2) improving the user interaction provided by the APOGEE system to give students early warning about the required naming convention. Most faculty adopted approach (1) in teaching.

Student project deployment could be another potential obstacle. Unlike the manual demonstration approach used by many instructors for student project assessment, the APOGEE system requires students to officially deploy their web applications on a server. There are several different approaches taken by instructors to tackle the challenge: (1) setting up a public server with pre-configured folder for students to publish their web applications (e.g., using virtual-directory), (2) allowing students power user access to a public server for setting up services including database systems (e.g., using secure shell and remote desktop connection), (3) providing web hosting accounts (e.g., on godaddy.com) to each student, (4) using an automated upload facility which takes the zipped project folder and automatically extracts student projects on a destination server, and (5) the ostrich approach: letting students handle all the details. In our practice, Hofstra employed approach (1), GSW took (2) and (4), Wingate used (3), and SPSU adopted approach (5). From the response of faculty members, all of the five solutions worked.

The TAF strategy has been implemented both inside and outside of classrooms. At Hofstra, students completed the projects in a fully monitored environment. At SPSU, a brief introduction was given in classroom. At GSW, the APOGEE system was used in a pure distance learning (online) class. We found that an early introduction of the grading system and the trial-and-failure strategy can significantly improve the students opinion towards TAF. In some of the class sections, lack of experiences with APOGEE dismayed some students and pushed them to give up at the early stage of the project.

Another interesting question to authors is: how likely would a student make another trial after a project failure? This could be measured by the number of submissions by the students. One general observation is that the initial grade

received is positively correlated with the number of attempts made by a student. Thus, one future extension of the APOGEE system could be sending alert flags to instructors and teaching assistant about the information of students who underperform in their initial trials.

RELATED WORK AND CONCLUSION

This paper presents TAF, a trial-and-failure learning paradigm for deepening student understanding of software quality and security, using a cyclic improvement model. TAF is supported by an automated grading system called APOGEE. Both TAF and APOGEE are implemented in five class sections at four universities. We report our success and failure stories with the model.

The trial-and-failure learning strategy can be regarded as one application and extension of the active learning paradigm [1]. Unlike the frequently used teaching patterns of active learning [2,3], the TAF strategy relies much more heavily on the use of automated grading and tutoring system. This has created both opportunities and challenges to educators. On the one hand, automated grading allows providing 24X7 services to students, elevating faculty productivity, and ensuring the fairness of grading. On the other hand, reported by participating faculty, the automated service needs to be better integrated with human service – it is important to identify those students lagging behind at the early stage.

Compared to the earlier efforts in automated grading and tutoring (e.g.,[14]-[23]), the uniqueness of APOGEE is its ability to handle GUI web applications. APOGEE subsumes its predecessor ProtoAPOGEE [6] and includes a variety of new features such as GUI test action editor.

As our future work, we plan to further investigate the key factors that determine the success of the TAF paradigm. For example, it is interesting to use the number of re-submissions for measuring a student’s perseverance when facing a project failure. For another example, using techniques such as Bayesian Network [24], it is possible to provide intuitive project failure analysis and infer a student’s deficiency in certain knowledge area. The APOGEE system can be enhanced with a variety of features (such as binary data payload, multi-threaded stress testing) for providing a thorough inspection of project submissions. It is also interesting to broaden the impacts of the TAF strategy to other science and engineering courses, using automated grading tools such as WebCat [20].

ACKNOWLEDGEMENT

This work was supported by NSF under contract DUE-0836859, 0837275, and 0837020.

REFERENCES

- [1] C.C. Bonwell and J. A. Eison, “Active Learning: Creating Excitement in the Classroom,” *JB ASHE Higher Education Report*, No. 1, George Washington University, Washington, DC , 1991.

- [2] B. J. Millis and P. G. Cottell. "Cooperative Learning for Higher Education," *American Council on Education/Oryx Press Series on Higher Education*, 1997.
- [3] J. Bergin, J. Eckstein, M. L. Manns, H. Sharp, "Patterns for Active Learning", available at <http://www.pedagogicalpatterns.org/current/activelearning.pdf>
- [4] A. Chickering and Z. Gamson, "Seven Principles for Good Practice," *AAHE Bulletin*, March 1987,
- [5] M. Sorcinelli, "Research Findings on the Seven Principles," *New Directions in Teaching and Learning*, 1991.
- [6] Xiang Fu, Boris Peltsverger, Kai Qian, Lixin Tao, and Jigang Liu, "APOGEE – Automated Project Grading and Instant Feedback System for Web Based Computing," in *SIGCSE'08*, 2008.
- [7] Xiang Fu, Kai Qian, Boris Peltsverger, Kent Palmer, Chong-wei Xu, Yu Zhang, "New Trends in Automated Project Grading for Web Programming Classes," (poster) in *SIGCSE'10*, 2010.
- [8] Xiang Fu, "APOGEE Project Website", available at http://people.hofstra.edu/Xiang_Fu/XiangFu/Projects/APOGEEHomepage/index.php.
- [9] Jonathan Kohl *et al.*, "Watir: Web Application Testing in Ruby, " available at <http://wtr.rubyforge.org/>.
- [10] jymenen@users.sourceforge.net, "WatiN: Web Application Testing in .Net," available at <http://watin.sourceforge.net/documentatie.html>.
- [11] M. Sztipanovits, K. Qian, X. Fu. "The automated web application testing (AWAT) system." In *Proceedings of the ACM Southeast Regional Conference 2008*, pp. 88-93, March 2008.
- [12] C. Anley. "Advanced SQL Injection In SQL Server Applications," Next Generation Security Software LTD White Paper, 2002.
- [13] J. Rafail. "Cross-site scripting attack," CERT Coordination Center, Carnegie Mellon University. Available at <http://www.cert.org/archive/pdf/cross\ site\ scripting.pdf>.
- [14] G. Forsythe, and N. Wirth, "Automatic grading programs," *Communications of the ACM*, 8(5) pp. 275-529, 1965.
- [15] Urs Von Matt, "Kassandra The Automatic Grading System," *ACM SIGCUE Outlook*, Vol 22 No. 1, pp. 26-40, 1994.
- [16] Sandra Foubister, Greg Michaelson, and Nils Tomes, "Automatic assessment of elementary Standard ML programs using Ceilidh," *Journal of Computer Assisted Learning*, Vol. 13, No. 1, pp. 99-108, March 1997
- [17] José Paulo Leal, and Nelma Moreira, "Automatic Grading of Programming Exercises," Technical Report DCC-98-4, DCC-FC&LIACC, UP, June 1998.
- [18] Mikko-Jussi Laakso, Tapio Salakoski and Ari Korhonen, "The Feasibility of Automatic Assessment and Feedback," in *Proceedings of the IADIS International Conference on Cognition and Exploratory Learning in Digital Age (CELDA 2005)*, 2005.
- [19] Wiley Inc., "Companion Tool of Java Concepts, 5th Edition", information available at <http://he-cda.wiley.com/WileyCDA/HigherEdTitle/productCd-0470105550,courseCd-CX0500,pageType-supplements.html>, retrieved May 1, 2007.
- [20] S. H. Edwards, "Using Test-Driven Development in the Classroom Providing Students with Automatic, Concrete Feedback on Performance," in *Proceedings of 2003 International Conference on Education and Information Systems: Technologies and Applications (EISTA'03)*, pp. 421-426, 2003.
- [21] S. H. Edwards, "Using Software Testing to Move Students from Trial-and- Error to Reflection-in-Action," in *Proceedings of the 35th SIGCSE Technical Symposium on Computer Science Education*, ACM, pp. 26-30, 2004.
- [22] Man Yu Feng and Andrew McAllister, "A Tool for Automatic GUI Grading," in *Proceedings of 36th ASEE/IEEE Frontiers in Education Conference*, 2006.
- [23] CSUS, "Programming Contest Control System - PC^2", available at <http://www.ecs.csus.edu/pc2/>
- [24] K. B. Korb, and A. E. Nicholson. *Bayesian Artificial Intelligence*. ISBN-10: 1584883871, Chapman & Hall/CRC, 2004.

AUTHOR INFORMATION

Dr. Xiang Fu Assistant Professor, Department of Computer Science, Hofstra University, Xiang.Fu@hofstra.edu.

Dr. Kai Qian Professor, School of Computing and Software Engineering, Southern Polytechnic State University, kqian@spsu.edu.

Dr. Kent Palmer Associate Professor, Goshen College, kpalmer@goshen.edu.

Dr. Boris Peltsverger Professor, School of Computing and Mathematics, Georgia Southwestern State University, plz@canes.gsw.edu.

Brian Campbell Adjunct Faculty, School of Computing and Mathematics, Georgia Southwestern State University, campbell@canes.gsw.edu.

Dr. Billy Lim Professor, School of Information Technology, Illinois State University, bllim@ilstu.edu.

Dr. Paul Volt Emeritus Professor, Illinois State University, wpvogt@ilstu.edu.