Research Statement (Daqing Hou, Clarkson University)

The long-term goal of my research program is to find ways to improve the effectiveness of software reuse as a general approach for software productivity. I am interested in the problem of how developers reuse existing software, the situations when reuse is a good idea, and how to make the reuse more effective when it occurs. This is a worthwhile and important problem as software is increasingly being built from existing parts. So far, my research has involved identifying the actual obstacles for effective software reuse [3, 7, 12]; understanding how reusable software and API's (Application Programming Interfaces) evolve as well as the intents behind the evolution [1]; and finally, building and evaluating tools that find errors and provide advice for client code that uses an existing design [5, 11, 13]. In our research, we frequently draw on the usability principles for software design [1-4, 7-8, 13]. Our tooling efforts often involve the customization of existing software development environments and tools, such as Eclipse [2, 5, 10, 11] and gcc [8], as well as the application of static program analyses [2, 5, 10, 11] and other formal methods [5]. As research methodologies, I have used case studies [1, 3, 9], laboratory user studies [6], and tool building and empirical evaluation with practical data [2, 5, 10, 11]. I treat rigorous research evaluation seriously and prefer to validate my research in industrial settings [11]. *I publish papers mainly in peer-reviewed, selective conferences organized by the IEEE Computer Society or ACM.*

Empirical Studies on API Obstacles: Our prior empirical studies characterize some of the obstacles to the effective use of API's [3, 7, 12]. For example, in a recent study [3], we analyzed 172 randomly selected newsgroup discussions about the Swing API, using a two-dimensional approach. The first dimension concerns the programmer's goal and status when asking a question. The second dimension concerns the complexity of the solution sought in the newsgroup discussion, for which we distinguish between *simple method calls* and *customization tasks*. This study results in a list of specific obstacles for using APIs. In addition, it shows that customization tasks are challenging to use because they involve multiple pieces of dispersed information. For example, to display an empty folder as an internal tree node, a programmer would have to call three methods from three classes (Figure 4 in [7]). This is consistent with the result of another prior study [7], where we have shown the prevalence of customization tasks (twelve) for the JTree widget. The study also shows that the same API-related problems do occur multiple times, and that a programmer often provides code as contextual information to help illustrate his or her needs (92 cases). *These findings have been used as the basis for our current tooling effort on building a critic for critiquing the use of API's [4, 13].*

Enforcing Design Intent in Code: As part of my PhD work, we have investigated an approach for checking code conformance to important structural properties required by an API [11]. Our current work extends that research by treating behavioral properties as well as the evaluation of tool usability more thoroughly. For example, recently we have developed a concept recognition algorithm to recognize how a Java equals() method is defined [5]. Our algorithm can recognize abstractions such as array and set comparisons and translate these abstractions into an equality model in Alloy [14]. The extracted Alloy model is checked by Alloy Analyzer for violations of the equivalence properties. The analysis is inter-procedural, path-based. Our empirical evaluation using large systems such as JDK 5.0 shows that the tool can produce human-understandable Alloy models, find true bugs with reasonably low rates of false positives and negatives, and be efficient [5].

API Critics: With PhD student Chandan Rupakheti, currently we are investigating a tool to try to infer a developer's goals by symbolically executing and analyzing his or her code that uses a certain API. Based on the inferred goals, the tool offers pertinent, pre-packaged advices appropriate for the inferred goals. *We call such a tool a critic. A real-world analogy for a critic would be a specialty shopping-assistant working in a hardware store such as Lowes and The Home Depot, who listens to a customer's problems and guides the customer to the appropriate tools and their use. More than an error-finding tool, however, a critic not only tries to find errors in the client code, but also gives other helpful advices.* To assess the potential usefulness of the proposed critic, we have investigated another set of API discussions in the Swing Forum [13]. To narrow down the scope of our analysis, we chose to focus on issues related to GUI hierarchies and layout because these are fundamental topics that many find difficult to work with. We analyzed 124 of the 274 threads that contain the keyword "layout". We conclude that 63 threads could have been helped by our proposed critic. The other 61 were too general and lacked code snippets for us to concretely assess whether our critic could be helpful. Moreover, we formalized a dozen API rules that can be used by our proposed critic. These rules cover aspects of composing the GUI trees such as using a layout manager versus absolute positioning; creating a dynamic GUI; and common mistakes such as idempotent actions, sharing a layout manager by multiple panels, and parent switching in the GUI trees. These rules cover a substantial portion of the obstacles related to GUI hierarchies and layout. This provides encouraging initial evidence that the proposed critic can be both useful (in the sense of solving specific instances in the problem space) and near-complete (in the sense of covering a significant portion of the problem space). *We are testing the effectiveness of our first prototype of a critic in my GUI design course EE408 this semester.*

Managing Copy-and-Paste Programming: With PhD student Patricia Jablonski, we have also worked on improving the effectiveness of copy-and-paste programming (To answer the question *"Will our tools help a programmer code faster and produce fewer errors?"*) [6, 10]. The code clone literature has reported strong evidence that lack of clone awareness on the part of a developer often causes bugs. However, to our knowledge, there has not been a user study conducted to investigate the potential effects of clone awareness on bug reduction. To that end, we conducted a *first* laboratory user study to test the effects of increasing clone awareness and making the cloning relationship explicit inside the Eclipse IDE. Our study used a within-subject design and involved 14 student subjects and 8 clone-related programming tasks. While our study showed that subjects who used our tools in their tasks (CReN [10] and LexID for renaming identifiers) were statistically significant faster than those without, the effect of clone-awareness within the IDE was not shown to be statistically significant. In hindsight, we suspect that clone-awareness would be helpful mostly when programmers compare and reason about clones systematically (systematic approach). However, our detailed analyses of the captured videos revealed that a major confounding factor would be that the subject programmers use multiple other strategies to complete the programming tasks, for example, trial and error; and less than half of them actually employed the systematic approach. A better-controlled user study is needed to confirm this conjecture in the future.

Collaborative Work: I have been fortunate enough to have the opportunities to conduct two major collaborative projects with two ECE colleagues, Dr. Thomas Ortmeyer and Dr. Paul McGrath, as well as with Dr. Michael Schuckers from St. Lawrence University. The first effort is in the area of design methodologies for electric power distribution systems reliability. My contribution is the advanced computational analysis of the actual utility fault data to determine fault causes and establish historical fault rate data [15]. This work has resulted in a journal publication [16] and a technical report [15], both jointly with my collaborators. The second effort involves developing a usable software application for Dr. Schuckers' statistical methods for biometric authentication performance evaluation, PRESS v2 (See description in the Software Section below). PRESS v2 is currently being used by both the biometric industry and universities.

Software Development Tools and Applications: Over the years, we have developed and maintained several open-sourced software tools and applications, which are listed as follows (See http://serl.clarkson.edu/site/?page_id=24 for more details):

1. **Structured Constraint Language (SCL)** Software developers often fail to respect design intent due to either missing or ignored documentation of intent. SCL helps capture and confirm aspects of design intent by using structural constraints on a program model extracted through static analysis. The original designer expresses design intent in terms of constraints on the program model using the SCL language, and the SCL conformance checking tool examines developer code to confirm that the code honors these constraints.
2. **Copy and Paste (CnP)** Programmers often copy and paste code so that they can reuse existing code to complete a similar task and/or save time. Many times, they modify the pasted code. The CnP project includes several IDE features (e.g., CReN and LexId) to track the copy-and-pasting relationship and facilitate the process of modifying the pasted code.
3. **Better Code Completion (BCC)** BCC is a research prototype that is meant to improve one of the most commonly-used IDE features: Code Completion. The new strategies include sorting, grouping, and filtering API methods.
4. **Equals Checker (EQ)** EQ is a static analysis tool for finding problems from Object.equals() in Java in two layers. The low level detects errors through data flow analysis. The high level detects equality-related semantic errors through model finding using Alloy and its Analyzer.
5. **PRESS v2 (Program for Rate Estimation and Statistical Summaries)** PRESS v2 is a software tool that implements a set of domain-specific statistical methods for biometric authentication performance evaluation. The methods are documented in a book that Dr. Michael E. Schuckers has recently published. PRESS v2 is implemented in Java and AWT/Swing to offer an easy-to-use user experience.

Assessment of Research Impact: As of October 8, 2011, Google Scholar (link clickable[1]) reports a total of 186 citations of my papers. The most-cited two papers are about my PhD work on SCL [11] (45 citations, published in IEEE TSE June 2006), and a copy-and-paste management tool CReN [10] (33 citations, published in October 2007 with Clarkson PhD graduate Patricia Jablonski). My empirical study on programmer questions about API's has also received some attention [12] (2005, 17 citations). The findings of these papers have been cited as supporting evidence for their own research by authors from several prominent Software Engineering research groups, such as CMU, McGill, University of Calgary, UCLA, University of Washington, and University of Waterloo. More recent work at Clarkson University [1-9] expands and refines my earlier research in the area of supporting software reuse, but with a higher standard of rigor and depth in terms of research methodology, so I expect them to be better received by the software engineering research community gradually in the years to come.

[1] Use Google Scholar to search for D. Hou's publications: **http://scholar.google.com/scholar?as_q=daqing+hou&num=30&btnG=Search+Scholar&as_epq=&as_oq=&as_eq=relay+petroleum&as_occt=any&as_sauthors=&as_publication=&as_ylo=&as_yhi=&as_sdt=1.&as_sdtf=&as_sdts=33&hl=en**

My PhD work on SCL has attracted external attention from practitioners as well as others active in the software engineering research community. This shows that SCL is probably solving a problem that is relevant to the software industry.

- SCL has been used by practitioners at Comverse and at Ml Global Solutions, whom I have never met in person.
- In 2008, a researcher from Siemens Research, Princeton, NJ tried out the SCL idea in their software projects and published lessons-learned as a paper in FSE 2009 (Foundation of Software Engineering Conference).
- Several MS and PhD theses have been completed with major influence from SCL. SCL has been used by researchers at University of Alberta, University of California at Santa Cruz, University of Waterloo, Universidade Federal de Minas Gerais in Brazil, IIT India, and Technical University of Darmstadt, Germany.

My work on CnP (support for Copy-and-Paste Programming) and CReN is less than four years old, yet the CReN tool [10] has attracted 33 citations as well as several technical inquiries from National University of Singapore, the Free University of Berlin in Germany, and the University of Calgary.

Since the release of PRESS v2, we have received several inquiries about the use of PRESS v2 from organizations such as the GREYC laboratory in France and Northrop Grumman Corporation. Furthermore, Dr. Stephanie Schuckers' biometric class at Clarkson University is using PRESS v2 this semester. Our NYSERDA funded distribution systems reliability project was conducted in close collaboration with key industrial partners (such as National Grid, Central Hudson, and Massena Electric Department) to address their needs for improving reliability.

Ongoing Research and Future Funding Opportunities: In addition to my core research on API Critics, other ongoing research projects include applying text mining techniques to discover knowledge from API newsgroup discussions, a semantic-oriented clone differencing tool, automated algorithm recognition and its applications, and free-text keystrokes as a continuous authentication method.

References (with acceptance rates when known)

1. Daqing Hou, Xiaojia Yao. Exploring the Intent behind API Evolution: A Case Study. IEEE WCRE 2011. 10 pages. (29%)
2. Daqing Hou, David M. Pletcher. An Evaluation of the Strategies of Sorting, Filtering, and Grouping API Methods for Code Completion. IEEE ICSM 2011. 10 pages. (28%)
3. Daqing Hou, Lin Li. Obstacles in Using Frameworks and APIs: An Exploratory Study of Programmers' Newsgroup Discussions. IEEE ICPC 2011. 10 pages. (24%)
4. Chandan Raj Rupakheti, Daqing Hou. Satisfying Programmers' Information Needs in API-based Programming. Student Research Symposium, IEEE ICPC 2011. 4 pages.
5. Chandan Raj Rupakheti, Daqing Hou. An Abstraction-Oriented, Path-Based Approach for Analyzing Object Equality in Java. IEEE WCRE 2010: 205-214. 10 pages. (31%)
6. Patricia Jablonski, Daqing Hou. Aiding Software Maintenance with Copy-and-Paste Clone-Awareness. IEEE ICPC 2010. 10 pages. (20%)
7. Daqing Hou, Chandan Raj Rupakheti, H. James Hoover. Documenting and Evaluating Scattered Concerns for Framework Usability: A Case Study. IEEE APSEC 2008. 8 pages. (30%)
8. Cheng Wang, Daqing Hou. An Empirical Study of Function Overloading in C++. IEEE SCAM 2008. 10 pages. (38%)

9. Daqing Hou. Investigating the Effects of Framework Design Knowledge in Example-based Framework Learning. IEEE ICSM 2008. 10 pages. (26%)
10. Patricia Jablonski, Daqing Hou. CReN: A Tool for Tracking Copy-and-paste Code Clones and Renaming Identifiers Consistently in the IDE. ACM ETX 2007. 5 pages.
11. Daqing Hou, H. James Hoover. Using SCL to Specify and Check Design Intent in Source Code. IEEE Trans. Software Eng. 32(6): 404-423 (2006).
12. Daqing Hou, Kenny Wong, and H. James Hoover. What Can Programmer Questions Tell Us About Frameworks? IEEE IWPC 2005. 10 pages.
13. Daqing Hou, Chandan Rupakheti. Toward a Critic System for API Client Code using Symbolic Execution. Unpublished working paper. 2011. 4 pages.
14. Daniel Jackson. Alloy: A Lightweight Object Modelling Notation. ACM Trans. Softw. Eng. Methodol., 11(2):256-290, 2002.
15. Thomas Ortmeyer, Paul McGrath, Daqing Hou. Development of a Practical Reliability-based Design Methodology for Electric Power Distribution Systems. NYSERFA Technical Report 10-26, October 2010. http://www.nyserda.org/publications/development-of-a-practical-reliability-formatted-edited-final.pdf. Last accessed: October 8, 2011.
16. Thomas Ortmeyer, Joel Reeves, Daqing Hou, Paul McGrath: Evaluation of Sustained and Momentary Interruption Impacts on Reliability-Based Distribution System Design. IEEE Transactions on Power Delivery. 25(4): 3133-3138. 2010.

Teaching Statement (Daqing Hou)

I find that teaching programming and software engineering is both challenging and rewarding. Programming and software engineering are challenging because they require close attention to both abstractions and details simultaneously at multiple levels. To most learners, these topics offer a completely new experience: novel notations and new ways of thinking about computational processes, neither of which has been covered as much as Sciences and Mathematics in K-12 education. Yet, programming and software engineering are increasingly being recognized as vital skills for the future, and computer science and software engineering are considered the third pillar of Science along with theory and experimentation. Thus, teaching programming and software engineering is also important and can be rewarding. For example, I found that determining where/why a learner is having difficulty involves similar kinds of observations and analyses as in Software Engineering research that involves industrial programmers. I have also had some success in integrating Software Engineering research and my own classroom teaching [2].

Based on slightly more than five years of experience in teaching four undergraduate and graduate courses, I have learned to apply four strategies in my classroom teaching. I have also learned some lessons in advising graduate students. I summarize these in the following paragraphs. *Selected written comments from student evaluations are included in the end.*

One, develop a *conceptual framework* [1] for the subject matter to be taught, and use the framework to organize teaching. For example, in my Introduction to Programming course EE261, I focus my teaching on a set of fundamental concepts such as data values and data types, variables, expressions, assignment statement, basic control structures, program state, function, and class. This conceptual framework helps structure the course content, providing a clear path for the topics that need to be covered throughout a semester. Furthermore, it also prevents me from being distracted by the many additional features of C++, which could be covered in more advanced courses. Such a framework has proven even more critical in my upper level GUI Design course EE408, because without its guide, I could have been easily overwhelmed by the rich details of an industrial framework such as Java Swing. Instead, I focus on teaching the core design of a GUI framework [2]. Therefore, the development of a conceptual framework for my course helps focus my teaching on the most important concepts and allocate enough of the precious lecture time to ensure that students really understand them. It also helps to avoid the "mile wide, inch deep" problem where too many topics are only superficially covered in too short a period of time. In this sense, I believe in "less is better."

Two, motivate and engage students by developing course materials that connect with things and current societal needs that students can relate themselves to. For example, in my Introduction to Programming course, I spent the first couple of lectures introducing the broader notion of computational processes and its practical applications, quoting such diverse examples as the New York Clearing House, Tax Law, LEGO MINDSTORM Robot, as well as an in-class activity of bubble sorting with the help of ten students. As another example, when teaching recursion, I used Russian Dolls and a wooden toy of the Hanoi Towers for illustration. I could clearly sense the excitement from my class, and some students even expressed in their written teaching evaluation comments that they liked to have more of these.

Three, promote deep, active learning and understanding, and de-emphasize rote learning. In my programming lectures, I strive to give students carefully planned demonstrations and examples with the necessary scaffolding. I model the problem solving process in a step-wise fashion, making sure that the process always starts from what the students already know. To enable students to experience non-trivial design problems within limited time, I provide the necessary scaffolding to embed the concepts to be learned within a background project. Finally, I always design multiple examples that cover the same concept so that students can generalize from them.

Four, build multiple feedback channels and adapt teaching to learners' situations and needs. To gather feedback about my teaching, I have used the assessment of results from homework grading, in-class polls, office hours, online forums, and the course management system (moodle). At the end of each semester, I spent time analyzing the written comments from student evaluation to identify opportunities for improving a course for the next offering. This effort seems to have paid off as can be seen from my improved course evaluation scores. I also seek opportunities to interact with students in informal settings such as cafeteria and libraries to gather feedback. For example, the first time when I taught Introduction to Programming, I spent four lectures on logic. In the second offering, when chatting with a student, I was surprised to learn that logic is actually covered in New York's high schools, where most of my students were from. As a result, I devote two lectures instead.

Course	**Average enrollment**	**Times taught**
EE261 (Undergrad): Introduction to Programming	67	3
EE408 (Undergrad): GUI Design and Principles of Usability	11	5
EE564 (Grad): Enterprise Computing	6	2
EE569 (Grad): Program Analyses for Software Engineering	4	4

I have taught four courses for a total of fourteen times. The two undergraduate courses are part of Clarkson's ABET accredited Software Engineering and Computer Engineering degree programs. I re-developed the content for the four courses based on the existing descriptions in the course catalog, including weekly lectures, laboratory exercises, homework assignments, and term projects. In addition to these four courses, I can also teach other courses such as data structures, algorithms, software design (including but not limited to object orientation and design patterns), capstone project courses, and compiler construction.

The median score for student evaluation of my instruction is 4.2 (out of a scale of 5: 1, very poor; 2, poor; 3, satisfactory; 4, good; 5, excellent), which is the same as the announced university median. Similar to elsewhere in North America, our pool of students is a mix of various ability levels, although ours would be more critical in teaching evaluation due to the higher cost of a private Clarkson education. To give a better idea on how our students are like, the US News and World Report's college ranking placed Clarkson University in the "A+ Options for B Students" list, where they "identify those where non-superstars have a decent shot at being accepted and thriving -- where spirit and hard work could make all the difference in admissions offices."

Graduate Advising: I enjoy working with graduate students. I treat students as adults and tend to provide a lot of encouragements. I try to instill a sense of humor during our interaction. I realize that most of my graduate students need me to provide the broader contextual and motivational background for their research projects, so I assume that responsibility. I often follow a bottom-up process in guiding student projects. Occasionally, I write code with them, hoping that this can serve as a model for them to see interactively how things can be done. I regularly (weekly) have lengthy discussions with graduate students on technical details from their projects. I find that these discussions force both the students and me to learn to build the right conceptual framework for the research project, to ask the right research questions, and to clarify and refine the situations. This effort often pays off when it comes to writing theses and research papers. *I have graduated 1 PhD student, 7 MS students, and 1 ME student. All of my graduate students have published at least one full research paper in an IEEE or ACM conference. Currently, I advise 1 PhD, 4 MS students, and 1 ME student.*

Written Comments from Student Evaluations

To provide a quick overview on how students think about my teaching, I include a set of selected written comments from student evaluations.

EE261: Introduction to Programming and Software Design

- *"Professor Hou is extremely knowledgeable on the course topics & very fluent in C++. I was very impressed by how available he made himself to help students during evening and weekend hours. He truly went well above and beyond what is expected."*
- "Prof seemed to be very prepared for every class and on time, knew what he was talking about and gained credibility with the class. He really cared about his students and wanted them to succeed in a difficult course where it is natural for students to get lost at points and need more clarifications on topics. He also had multiple study sessions/hw help sessions which was good."
- *"Professor Hou does his best to interest the students and to ensure they understand."*
- "Very nice instructor, very kind and great at teaching clearly. Seems to like what he does and is very good at it. "Very good course. I learned a lot."
- *"The HW was challenging and was very useful for practice. [Prof. possessed] strong understanding of material; very eager to help, was enjoyable to attend lecture."*
- "Well structured. I feel as though I get good credit for things I worked hard on. Good teacher. Helped me out numerous times in office hours and really helped me along in the course."
- *"Prof Hou is a smart man who really knows his C++. He is a good teacher. ... He is very helpful in office hours."*
- "Very thankful for the support received through email and how prompt it was." "More [in-class examples of] washing machines and Russian doll stories"

EE408: GUI Design and Principles of Usability

- *"The course provides insight into the development and creation of GUIs by starting at fundamental level of hand-coding GUI's."*
- "The course really helped me understand GUIs from the users perspective."
- *"I enjoyed the course ... teaching is very clear and concise, spends a lot of time explaining things for everyone (a good thing)"*
- "The course covers a lot of information effectively in a short amount of time."
- *"The in-class examples were very helpful"*
- "Good projects …"
- *"Very concerned about students"*
- "The Professor is always willing to help students and provides great feedback on how to improve and fix problems."

EE564: Enterprise Computing (grad course)

- *"There was a large range of skills developed during this course, including programming, reading, writing, presentation, and code exploration."*
- "It is a practice-oriented course, and almost cover all aspects of the Enterprise program."
- *"He is very prepared for his class. He managed to do justice to such a huge course with lots of technologies involved."*
- "I really liked the in-class discussions…"
- *"[Prof] forces us to perform truly critical thinking."*

EE569: Program Analysis for Software Engineering (grad course)

- *"I really enjoyed this class a lot. It was challenging, yet fair. It helps us develop many skills … The division of the course by month was effective. Really good teaching, encouraged class participation, and students got attention because it was a small class. It was more of a discussion than a lecture."*
- "It is very helpful to EE graduate students!"
- *"Topics were useful."*
- "Teach very clearly with concepts and implementation. Help students learn new concepts easily and useful."

References

1. How People Learn: Brain, Mind, Experience and School. Editors: John D. Bransford, Ann L. Brown, and Rodney R. Cocking, National Research Council (U.S.). ISBN: 9780309070362. National Academy Press, 2000.
2. Daqing Hou: Investigating the Effects of Framework Design Knowledge in Example-based Framework Learning. IEEE ICSM 2008. 10 pages.

Service (Daqing Hou)

As a faculty member in the Electrical and Computer Engineering Department, I have served on several standard departmental and inter-departmental committees such as the Software/Computer Engineering Curriculum Committee, the ECE PhD Comprehensive Exam Committee (Software Design), and thesis committees. Instead of going over the full list of services which I have contributed to, which can be found in my CV, I would like to highlight two particular items on the list. One is my voluntary participation in most of our Fall/Spring Open House activities in the last five years. By assisting the Director of Software Engineering in organizing the Open House, I had the opportunities to directly interact with the prospective students and their families, to understand their concerns about the selection of college majors and career plans, and to communicate to them what we believe is the bright future of Software Engineering as a profession and its relevance in the future. This in turn helped me in teaching our students in classroom because I knew my students and understood their needs better. Overall, I enjoyed this experience and believe that I have made a positive contribution to the growth of our Software Engineering degree program.

The other service that I want to highlight is serving the broader, international Software Engineering research community. I have been a Program Committee member or a Session Chair for several major IEEE and IBM research conferences in the areas of Program Comprehension, Software Maintenance, and Reverse Engineering. As a Program Committee member, I have always strived to do my best to write high-quality paper reviews and actively participated in the process of selecting the best papers for the conference programs in a professional, fair, yet rigorous manner. In the past five years, I have reviewed a total of 149 full research papers. This experience is unique and valuable in that it allows me to observe first hand the conference process and how decisions were made on paper selection. Perhaps more importantly, I learned how to judge a piece of work for its research contributions in Software Engineering, which in turn helped me in defining my own research program. Last but not least, this professional service has also helped increase the visibility and awareness of Clarkson University and our Software Engineering degree program internationally in the Software Engineering research community.