Michael Toomim

Intelligent Interfaces for End-User Information Engineering

WYSIWYG interfaces have made it easier for novices to author small documents such as web pages, drawings, and simple spreadsheets. However, large documents are still fundamentally difficult to maintain. To edit a page on CNN.com, for instance, a novice needs to learn its architecture of headers, navigation bars, stories, and hundreds of other conceptual elements that are repeated across its thousands of pages. Tables and figures follow patterns of formatting that differ depending upon whether they contain statistics, commentary or links to featured articles. Some conceptual patterns are repeated within the same page, such as the formatting codes to display sub-headings, or each row in a table of contents. Duplication is also present in many other types of documents; there are repeated and parallel melodies and rhythms in music scores, repeated formatting in magazines and brochures, repeated objects and sub-objects in CAD documents, repeated diagrams in posters and Powerpoint slides, and repeated formulae in spreadsheets. If left unmanaged, patterns of duplication in large documents would make modifications intractable. Modifying a navigation bar of CNN.com, for example, would require editing thousands of pages.

Developers manage patterns of duplication by replacing them with *abstractions*. In our CNN.com example, a developer might write a PHP function to produce navigation bars so that all navigation bars can be modified by editing the single function. Unfortunately, not all duplication is abstracted in practice. Even expert programmers leave a sizable amount of duplicated software source code unabstracted; recent studies show that respected software systems like the Linux kernel, Java JDK, FreeBSD, MySQL, PostgreSQL, and X Window System are all 20–30% duplicated, and some software is as much as 60% duplicated. (Li, 2004; Toomim, 2004)

Moreover, end-users are much less likely to create abstractions than expert programmers. For example, Blackwell (2001, 2004) found that, in a group of Microsoft Word users, end-users were more than ten times less likely to create "text style" abstractions in their documents as programmers, even if they knew how to use that feature of Word. The study also showed that these end-users are dramatically less likely to create file-system abstractions. In another domain, Bellotti (1995) reports that designers principally work in terms of concrete representational artifacts, rather than abstract concepts, even if abstractions are available. In software, novice programmers are known to create fewer abstractions than experts. (Detienne, 2002) Thus, while programmers create fewer abstractions than they think they should, end-users and novices create very few at all.

This is unfortunate. If end-users will not abstract away patterns of repetition, they will be unable to author, understand, and modify digital documents beyond a certain size and complexity. An abstractionless user certainly could not edit CNN.com, for example, and possibly not even a subset of CNN.com. Nor could such a user easily author work in other domains; *e.g.* creating a computer-graphic landscape with hundreds of trees, or an electronic music score with hundreds of voices. This would be a regretful scenario, since the *ideas* published in CNN's news are not difficult for an end-user to comprehend, edit or express, but rather the structural characteristics of their transcription in a website.

I believe an intelligent user-interface can remedy this problem. I plan to extend an interaction technique I developed for programmers and duplicated code, called Linked Editing (Toomim, 2004), to help end-users manage duplicated content in non-programming domains. My dual hypotheses are that end-users will prefer its concrete interaction-style over abstraction, and that it will enable them to author and manage larger, more complex documents than they would otherwise be able to.

I shall illustrate my proposal with a scenario where an end-user leverages such an environment to modify CNN.com. First, the system automatically analyzes the website and finds all patterns of duplication. Then, as the user moves her cursor over a duplicated block of content, such as a navigation bar, the system provides a visualization of the block's corresponding copies—miniature depictions of the navigation bars on other CNN web pages. The user can now change all navigation bars simultaneously by simply editing any one. Additionally, the user can make changes to any single duplicated instance, or subset of instances; imagine, for example, that the navigation bars on all "science" pages need to have additional entries for the "national science foundation" news category. To implement this, the user selects science pages from the miniaturized visualization, and the system infers (by example) that the user is selecting all pages with "science" in their header. It briefly highlights, in orange, the word "science" in each document's header to indicate its inference. The user now simultaneously edits the desired entry into all science navigation bars.

Note that introducing a new type of *difference* amongst elements, as was done here, would be much more difficult using a traditional (*e.g.* PHP) functional abstraction: the user would have had to add a parameter in the function definition to represent the new type of difference (science or not science page) as well as modify each use of the function to provide the appropriate parameter. In general, abstraction systems can become more complicated as additional differences are required. The system described here, however, adapts automatically to the concrete content created and infers an implied inheritance structure.

Although I have proposed this system as an alternative to abstraction, it is, in fact, designed to coexist with traditional abstractions. Firstly, the user can click a button to elide repeated boilerplate content from view, leaving only its unique differences visible, in the same way that a function call hides the function's body and shows only parameters. Later, the system can semi-automatically transform the content into a traditional abstraction, requesting a name for it from the user. In this way, the system allows for a form of *incremental abstraction*, letting users work either concretely or abstractly, at their discretion.

Finally, this interface is *general*, and can be implemented in many domains of authoring environments. This allows a user's skill with managing duplication to transfer between domains. Most abstraction features, on the other hand, are dramatically different: users often must learn an entirely new set of concepts and interfaces in order to use the abstraction features of a new environment. This is difficult since many individual abstractions, like functions and style sheets, are complex in behavior.

Although a some of these ideas have already been developed in my research prototype on Linked Editing, a great deal remains before this scenario can be realized. First, the system will require a difference-analysis algorithm that accurately models the human process of finding differences between content structures; no existing diff algorithm has attempted to compute perceptually-optimized rather than edit-distance-optimized differences, and cognitive models like the Structure-Mapping Engine are not flexible nor efficient enough to map complicated document structure. Second, the duplication-finding algorithms and Linked Editing interface must be extended to work with non-textual content, such as vector graphics. Third, duplication-detecting algorithms are not yet sophisticated enough to detect fine-grained patterns, such as 10-character HTML formatting code applied to every cell in a table, nor higher-level patterns such as notes increasing in scale or an enumerated (but otherwise identical) sequence of list elements. Fourth, new visualizations will need to be developed to communicate such patterns to the user. Lastly, new empirical research will be needed to study in more detail the factors that affect abstraction usage. The advances in pattern-recognition contributed by this research could aid this effort by tracking duplication and abstraction uses over time, space, people, and situations.

I hope to pursue my PhD at CMU to work with Brad Myers, a member of the EUSES end-user software engineering consortium who has worked on and advised a variety of related end-user programming and programming-by-demonstration projects such as HANDS, Natural Programming, and Lapis. CMU also has other diverse resources in HCI, Cognitive Science, AI, and Software Engineering.

Victoria Bellotti, Simon Buckingham Shum, Allan MacLean and Nick Hammond. Multidisciplinary Modelling In HCI Design ...In Theory and In Practice. In *Proceedings of the conference on Human Factors in Computing Systems*. 1995. pp. 146–153.

Alan F. Blackwell. See What You Need: Helping End-users to Build Abstractions. In *Journal of Visual Languages and Computing*. 2001, (12). pp.475–499.

Alan F. Blackwell. *Personal Communication*. 2004.

Detienne, Francios. 2002. *Software Design – Cognitive Aspects*. Springer

Zhenmin Li, Shan Lu, Suvda Myagmar, and Yuanyuan Zhou. CP-Miner: A Tool for Finding Copy-paste and Related Bugs in Operating System Code. In Proceedings of the *6th International Conference on Operating Systems Design and Implementation*. December, 2004.

Michael Toomim, Andrew Begel and Susan L. Graham. Managing Duplicated Code with Linked Editing. In Proceedings of the *IEEE Symposium on Visual Languages and Human-Centric Computing*, Rome, Italy, September 2004.

Previous Research

My first interest in research came when I decided (naively) that the programming world needed me to write a new syntax-directed programming editor. I talked to some professors and graduate students, and soon discovered that the Harmonia research group on campus worked in such an area, and was preparing to develop a yet more advanced programming editor. I have now worked with the Harmonia group for four years, investigating new user-interfaces to programming and completing two large projects. My first was to develop such a new editor, called *Harmonia-Mode*. Rather than being fully syntax-directed, it provides language-aware analysis-driven editing and navigation abilities, but over textual source-code. I overcame a variety of technical hurdles in development, designing multiple novel algorithms, architectures, and protocols to incrementally synchronize different views and editors to the syntax tree maintained by our group's incremental lexer and parser. Then I designed a novel programmer's user-interface powered by on these analyses. The synchronization problems were quite challenging; in fact I later learned that Stanford database researchers had attacked the same problems, and published similar solutions, a few years earlier. I wrote roughly 23,000 lines of code in all, integrating it with the much larger Harmonia program analysis framework, and worked a great deal to make it stable and robust.

This effort had many practical benefits. Harmonia-Mode is now used by students in Berkeley's compilers class to write their programming assignments—as they write their compiler's program analyses, the editor I wrote is analyzing the programs they write (and providing editing services). This gives students interactive exposure to the processes they are studying. I managed the software's introduction to the class, writing a web-based user's manual, coordinating with the class's staff, and assisting students when they had trouble. I also conducted a usability survey to evaluate the editor's effectiveness. I found that the software worked! Following the successful classroom experience, we released the software on the internet. The public has now used Harmonia-Mode for two years. Harmonia-Mode has also been used or extended internally by five other researchers, to support seven of their research projects, and has been ported to the Eclipse IDE in a combined effort by myself and others. I learned a lot about the effort that goes into a large-scale research project in this process.

My second Harmonia project was one I worked on much more independently (from its initial idea to publication), and was very successful. The project, called "Linked Editing," is a new way of managing duplicated code in programs, using an augmented programming environment rather than the programming language. Programmers know that duplicated code is difficult to read, modify and maintain, and that they should abstract it with functions, macros or methods. There have been decades of research devoted to languages, tools, and methodologies to help them abstract code. It turns out, however, that even expert programmers, on respected projects, still chronically copy and paste code in practice: recent versions of the Linux Kernel, X Window System, FreeBSD, Java JDK, and PostgreSQL, for instance, have all been shown to contain 20–30% duplicated code. Some private systems contain as much as 60% duplication. Apparently, duplicated code is not going away. My explanation is that programming abstractions are inadequate solutions: they can be hard to introduce, make code hard to understand, and make it hard to modify. Linked Editing manages the problems of duplication without these side effects.

I pursued every step of this project independently, meeting weekly with my advisors Andrew Begel and Susan L. Graham for feedback. I elaborated upon my initial idea with colleagues and other researchers on campus, refined it with a literature review, drew up a plan, and submitted the project as a research proposal for the Intel Undergraduate Student Research Award Contest, on a whim. Surprisingly to me, it was accepted and I was awarded $2,000 in funds to continue work! I carried out my plan over the next nine months, performing a contextual inquiry and interviewing expert and novice programmers to learn their work habits, designing and evaluating multiple ideas and iterations on how to interact with a duplication-managing editor, implementing a final highly-usable prototype, and running a formal HCI user study to evaluate it with respect to my research goals. I was elated when the results were positive: not only did Linked Editing avoid the steep costs of functional abstraction in the study (requiring only 2.7% of the initial programming effort), but Linked Editing was reported to to make code even more maintainable than functional abstractions. Going through this process was extremely educational, in terms of administrative issues (planning a research program, proceeding through human subjects' committees, spending my own grant money), research issues (defining and refining a research topic, focusing on what's important, finding related work and framing the issue), and knowledge issues (I audited a Psychology research methods course to learn statistics and data analysis).

After the nine months were up, all 17 students with accepted proposals were flown to Intel's headquarters in Santa Clara to present their final results. I was surprised again when I won second place, and another $3,000! (An intel employee later mentioned, in jest, that I might have done better if I hadn't performed the demo on my AMD-based desktop.) I also recently submitted the work as a full paper to the IEEE symposium on Human-Centric Computing (an HCI+Languages conference with a 30% acceptance rate), where it was accepted with an average peer review of 6.3/7.0—I just returned from my presentation in Rome a month ago. In my senior semester, the Berkeley EECS department also recognized this work by presenting me with the Warren Dere Award in Engineering Design.

Since submitting the paper, I have been working on improving some of the analyses used in the prototype for finding alignments and differences between blocks of duplicated code (which may not be exact copies). The analyses must find an alignment as close as possible to one that a person would choose, or else Linked Editing's edit generalization inferences may need to be corrected. It turns out, however, that no existing alignment or differencing algorithm has been designed to optimize for "human-like" alignments, and a review of the Cognitive Modeling literature found a variety of similarity judgment models and structural alignment models, but I found that each makes critical predictive errors with source code-like structures when compared with human subjects, via an informal study I ran. My current research efforts are directed at filling this gap, and designing a principled cognitive model and usable algorithm for predicting such similarity judgments of structured perceptual information. Such a model, I believe, can make contributions to Cognitive Psychology (understanding how people align perceptual structured objects), as well as Artificial Intelligence and Intelligent User Interfaces.

I have completed a number of other HCI research projects in coursework, which I will describe here briefly. My first HCI research project was in an experimental CS98 course entitled "HCI Research Studio." My group designed and evaluated a hybrid physical/virtual collaborative brainstorming application on a digital whiteboard. As a sophomore, I was flattered to learn that our results informed Scott Klemmer's work on *The Designer's Outpost*. Jason Hong also liked our design enough to re-implement it as a sample application in *Satin*. In my main User-Interfaces course, my group designed a firefighting command-post, performing elaborate contextual inquiries and user studies with real firefighters to assess their needs. Our project was selected as one of the top three in the course, and a software engineering class later re-implemented it for their project. Two years later, a graduate research group became interested in our project, picked up where we left off, and published two conference papers. In CS294 "Design Realization," my artist's workbench project was featured in two publications. I am proud that every major research project I have worked on has resulted in lasting impacts or publications.

FULL CONFERENCE PAPERS

Michael Toomim, Andrew Begel, Susan L. Graham, Managing Duplicated Code with Linked Editing. Proceedings of the *IEEE Consortium on Visual Languages and Human Centric Computing*. Rome, Italy, September 2004, pp. 173–180.

MINOR REFEREED ARTICLES

Stephen McCamant, Michael Toomim, Gruia Pitigoi-Aron, Duy Lam, Brian Chin, Dmitriy Ayrapetov, The Harmonia Research Project. *Berkeley EECS Research Journal*, Spring 2002, 1.

Michael Toomim, Ibrahim Merchant, Maribeth Back, Steven Harrison. Shazam—An Artist's Workbench for Choreographing Visual Effects. *LOOP: AIGA Journal of Interaction Design Education*, June 2003, 7.

MEDIA

Pescovitz, David, A Symphony of Data. *Lab Notes*, March 2003, 3(2). Public Affairs Office of the UC Berkeley College of Engineering.

Why I Desire Advanced Study

The chief reason for which I know I desire advanced study is that I love research, and have loved it for a long time. I have done undergraduate research for over four years. During that time I have looked forward to summer and winter breaks so that I could be free from coursework to do more research. When I took a break from research after finishing a project, I would relax for a bit, and then start dreaming about ideas for future research.

Like any certain self-knowledge, this conviction has been tested by experience. The miserable experience, in this case, of taking a programming internship at an optics company to convert legacy source code from C to C++. I spent my first summer away from college on the most remote building in the company campus, in the most isolated corner of the building, which was occupied by men in their 50's who knew nothing about computers. I tried to keep myself stimulated. I survived each day by taking notes on how I thought programming environments could be improved for end-users. In the next year, this motivation would guide me to start research with a programming tools group at Berkeley; by my senior year, I would be publishing the result of pursuing an idea I took notes on that summer. Being in industry has made me long for academia and its concomitant pursuit of things that are analytical and new.

Like any other researcher, my *goals* in research have also been motivated by experience. In this case, it is my experience of moving to Berkeley for college from a poor, welfare-class neighborhood. As an idealistic philosophical teenager, I valued art and personal expression, and was peeved by the fact that most people consumed so much more art—TV, CD's, toys—than they produced. I was convinced that the mentality of consumption posed a major impediment to the human condition: that it lead people to feel incapable of affecting the world, preventing them from standing up for themselves and pursuing proper goals, and thus obstructed their socio-economic advancement and impaired their emotional well-being. I did not, however, have a clue about what could be done to resolve the issue.

When I moved to Berkeley, I met people with similar aptitudes and personalities as the people I knew back home, but with attitudes of confidence; attitudes of expecting to make their own way in the world. Most of these people came from neighborhoods with higher income levels than mine. They often had parents and friends with advanced degrees. My Berkeley girlfriend's dad was a psychiatrist from Yale. I concluded that there was no appreciable difference in the nature of people at Berkeley, but that there had been an enormous difference in their environments. Through nurture, they had been shown ways in which they can affect the world.

In high school, I constructed my own creative outlet within the personal computer my rich uncle had generously bought my family. (Every kid, by the way, should have a rich uncle.) I connected to local BBS's, wrote and distributed music with low-tech synthesizers I downloaded, and was the only person I knew that could program. I believe that the act of expressing myself in such a creative outlet, by communicating a creative message to the world, showed me that I could affect the world. In turn, I believe that giving creative outlets to other disadvantaged computer users can show them how to affect the world.

Poor people often have computers now. They don't, unfortunately, have good creative outlet software. Software to support creativity is either expensive and designed for professionals, or a hack and designed for geeks. I want to design creative outlets for the disadvantaged people of the world. I think, if I can design interfaces and software that help people express themselves, that they will become better people. I do not have scientific evidence for why I believe this to be true. It remains what I believe, however, and is my goal in doing research.

Broader Impacts

I have never viewed education and research as being separable. I do not believe that the goal of research is primarily to identify new knowledge, but to disseminate important results to the right people. New knowledge, in my view, just often happens to be the most important information for researchers to disseminate. A common distinction between an educator and researcher may be that educators tend to have more broadly-useful knowledge and less knack for finding new things, whereas researchers tend to investigate new things without placing them in a broadly-useful context.

I have practiced both of these skills in college. Not only did I do undergraduate research, but I also taught a discussion section of introductory computer-science students as an undergraduate student instructor. I often found myself explaining abstract class topics in terms of concrete examples of issues I found in research, and using the knowledge gained while doing research on the psychology of programming to understand the difficulties that students faced in class, programming labs, and problem sets. I look forward to teaching again.

Also, my research goal of end-user authorship requires, in particular, for me to both research interfaces that match an end-user's capabilities *and* to educate end-users such that their capabilities are extended to better control computers. End-user authorship is a process of bringing computers and end-users closer together, which requires work on both sides of the equation of research and education. End-user programming is also a necessary subgoal in promoting the disadvantaged in computer science. The initial learning curve to becoming a programmer is steep, and people without exposure to the prerequisites often do not make the jump. If we can lower the initial barriers with end-user programming research, it will help expose disadvantaged individuals to a field that they man have never otherwise realized they enjoy.

The cognitive issues that prevent end-users from taking full advantage of computers is poorly understood. In my previous research, for instance, I found that there is little recognition of the difficulty people have in applying abstractions to their document structures. Even when we have acknowledged end-user's difficulties with abstractions, we have not equally investigated the degree to which programmers and expert users have difficulties (of lesser degrees) with the same problems. By investigating these questions, I hope to contribute to our general understanding of people's cognitive strengths and limitations.

I also hope to improve the state of underrepresented groups in the sciences via means other than HCI research. Although I believe that new interfaces can help these groups, I am not naïve enough to believe that social action is not necessary. In the same sense, I hope to that some of my future research prototypes become feasible, releasable free software projects, as my past Harmonia-Mode source code editor did. Working on released software may not further research goals as much as working on papers, but the benefit it makes to the intended users—who may not be able to afford commercial products—is worth the effort.

NSF Graduate Research Fellowship Program

APPLICANT RATING SHEET

Applicant: **TOOMIM, MICHAEL**
Panel: Computer Science 1

Intellectual Merit Criterion

Demonstrated intellectual ability and other accepted requisites for scholarly scientific study, such as the ability (1) to plan and conduct research; (2) to work as a member of a team as well as independently; and (3) to interpret and communicate research findings.

Overall Assessment of Intellectual Merit [✓ Excellent] ☐ Very Good ☐ Good ☐ Less Competitive

Basis for assessment:

WELL DEFINED, CONTINUOUS RESEARCH (HARMONIA GROUP) THAT SHOWS STRONG PROMISE. PUBLICATION RECORD IS GOOD AND SHOULD BE EXPANDED DURING GRADUATE STUDIES

Broader Impacts Criterion

Contributions that (1) effectively integrate research and education at all levels, infuse learning with the excitement of discovery, and assure that the findings and methods of research are communicated in a broad context and to a large audience; (2) encourage diversity, broaden opportunities, and enable the participation of all citizens--women and men, underrepresented minorities, and persons with disabilities--in science and research; (3) enhance scientific and technical understanding; and (4) benefit society.

Overall Assessment of Broader Impacts ☐ Excellent [✓ Very Good] ☐ Good ☐ Less Competitive

Basis for assessment:

USER INTERFACES ARE CRUCIAL IN EFFECTIVE USE OF COMPUTERS. DISADVANTAGED POPULATIONS CAN BENEFIT FROM IMPROVED ACCESS TO EDUCATIONAL AND OTHER INFORMATION. AN IMPORTANT ISSUE IS HOW TO MEASURE PROGRESS DUE TO IMPROVED USER INTERFACE. WHAT ARE THE METRICS TO MEASURE "BENEFIT TO SOCIETY."?

Reviewer comments to the applicant on this rating sheet should be constructive and reflect the two NSF merit review criteria. Applicants

NSF Graduate Research Fellowship Program

APPLICANT RATING SHEET

Applicant: **TOOMIM, MICHAEL**
Panel: Computer Science 1

Intellectual Merit Criterion

Demonstrated intellectual ability and other accepted requisites for scholarly scientific study, such as the ability (1) to plan and conduct research; (2) to work as a member of a team as well as independently; and (3) to interpret and communicate research findings.

Overall Assessment of Intellectual Merit [✓ Excellent] ☐ Very Good ☐ Good ☐ Less Competitive

Basis for assessment:

important ideas on an important topic

Broader Impacts Criterion

Contributions that (1) effectively integrate research and education at all levels, infuse learning with the excitement of discovery, and assure that the findings and methods of research are communicated in a broad context and to a large audience; (2) encourage diversity, broaden opportunities, and enable the participation of all citizens—women and men, underrepresented minorities, and persons with disabilities—in science and research; (3) enhance scientific and technical understanding; and (4) benefit society.

Overall Assessment of Broader Impacts ☐ Excellent [✓ Very Good] ☐ Good ☐ Less Competitive

Basis for assessment:

interesting point on ratio of consumption to production in the arts

Reviewer comments to the applicant on this rating sheet should be constructive and reflect the two NSF merit review criteria. Applicants will be able to view a copy of their rating sheets without the name of the reviewer on a secure web site. Maximum protection will be given

NSF Graduate Research Fellowship Program

APPLICANT RATING SHEET

Applicant: **TOOMIM, MICHAEL**
Panel: *Computer Science 1*

Intellectual Merit Criterion

Demonstrated intellectual ability and other accepted requisites for scholarly scientific study, such as the ability (1) to *plan and conduct research; (2) to work as a member of a team as well as independently;* and (3) to interpret and communicate research findings.

Overall Assessment of Intellectual Merit [✓ Excellent] ☐ Very Good ☐ Good ☐ Less Competitive

Basis for assessment:

· Interesting research plan with good potential.

Broader Impacts Criterion

Contributions that (1) effectively integrate research and education at all levels, infuse learning with the excitement of discovery, and assure that the findings and methods of research are communicated in a broad context and to a large audience; (2) encourage diversity, broaden opportunities, and enable the participation of all citizens—women and men, underrepresented minorities, and persons with disabilities—in science and research; (3) enhance scientific and technical understanding; and (4) benefit society.

Overall Assessment of Broader Impacts ☐ Excellent ☐ Very Good [✓ Good] ☐ Less Competitive

Basis for assessment:

Needs to creatively and explicitly address broader impacts (diversity, benefit to society) with credible substantial plans. (and preliminary steps in that direction)

Reviewer comments to the applicant on this rating sheet should be constructive and reflect the two NSF merit review criteria. Applicants