Radical Design:
From Pencils to Software to Processes to Clothing

Georges Grinstein
Fred Martin
Department of Computer Science
University of Massachusetts Lowell
grinstein, fredm@cs.uml.edu

Sarah Kuhn
Department of
Regional Economic and Social Development
University of Massachusetts Lowell
Sarah_Kuhn@uml.edu

ABSTRACT
Software development is presently in a crisis. As tool chains evolve, developers are expected to build increasingly complex software systems. Yet, creative and breakthrough applications often only arise from the combination of computing resources from multiple, conflicting domains. A new approach is needed to allow teams of designers—some software engineers, others subject matter experts—to combine their expertise in the design of products based on large, heterogeneous software systems. Here, we describe our formative work in this area, which we call Radical Design. We discuss the development of a graduate course that provokes students to think and work creatively, our pyramid model for creating a Radical Product, and a research plan for formalizing our approach.

INTRODUCTION
Most companies still focus on developing a competitive advantage within constraints imposed by perceived, often conservative, beliefs. Creating revolutionary products, whether these products are pencils, processes, clothing, or (most importantly) software requires breaking conceptual boundaries.

Both current processes of software development and the resulting software architectures result only in incremental software systems and software applications. We argue that there are new approaches that can support creative and breakthrough software development. We also argue that a new architecture for software applications is possible that will yield new radical applications of current "base technologies."

Much of software development has evolved from its earlier, simpler state (when the whole world of programming was far simpler and less powerful) into very complex, options-laden environments where the novice is given everything at once: both a hugely complex visual screen environment (IDEs), with many confusing choices, plus a fully-generalized programming language and set of APIs that can be used to make anything, but only after the user has climbed a very steep learning curve.

Further, even for the expert user (i.e., programmer) the contemporary environments (e.g., Microsoft Visual C++ / .net Studio, IBM's open-source Eclipse) are not very supportive of prototyping or other rapid development. Myers [3] argues that programs that do small things should be simple but often are very large. The typical "hello world" program for Motif requires close to 2 pages, in Java it requires the use of special overhead tokens (e.g., class, public, static, void) whereas the same in the Processing language [1] requires almost none.

Software development has often been compared to the building of other human-designed artifacts. Yet in the case of software, our ability to control and predict the course of the development process has been questioned. Companies are given grades on their development process "maturity" based on how well they can control their process. The paradigm shift we now ponder is whether control is the goal. Maybe adaptability is a higher goal.

PRIOR WORK
In the fall of 2005, co-author Grinstein offered a new course in the computer science department at UMass Lowell. This course was called Radical Design: From Pencils to Software to Processes to Clothing. The goal was to have students, most of whom become software engineers, break out of the classic mode of incremental software development.

The course's goal was to get students to begin to think out of the box and develop software applications in a more intuitive and simple manner. Product design and numerous other areas in the design world have often succeeded at breaking ground. What makes software applications and tools such behemoths?

Presented at the workshop "Exploring Design as a Research Activity" at Designing Interactive Systems (DIS2006), Penn State. See http://sigchi.org/dis2006/main.php?page=workshop .

The main goal was to have the students develop knowledge, insight and expertise in the radical development process for new objects. The course explored both evolutionary and revolutionary trends in design by first reviewing the fundamental principles of design and their history. The course then looked in more detail at the design process and associated theories, explored innovation in various technologies, and then reviewed modern approaches for breaking ground with designing objects.

The course was organized around three projects, each of which was redefined as the course progressed:

A. Introduction to Radical Design: design a radical piece of hardware (a concept design)

B. Design and implement a radical multimedia presentation, as a 1-minute product pitch (the midterm project)

C. Design a radical software tool (e.g., a language, algorithm, or UI) and prototype it (the final project)

Our successful experience in this class, in which students developed a diverse set of unexpected projects, has informed the ideas discussed in this paper, and our plan for subsequent work.

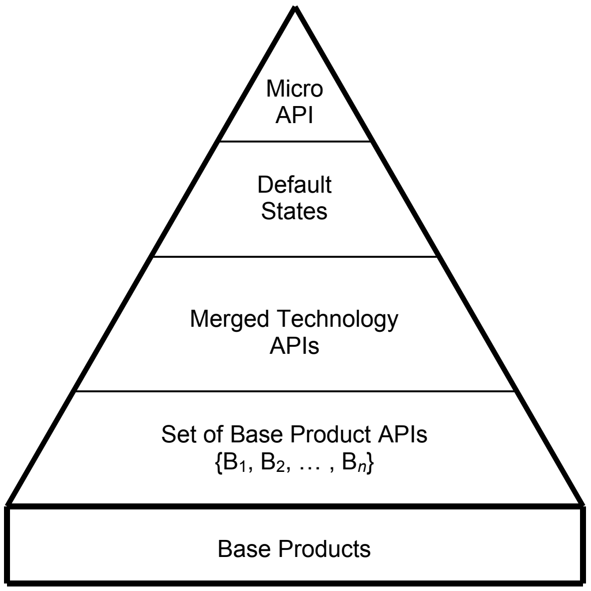

Figure 1: Pyramid Model for Creating a Radical Product

THE RADICAL DESIGN APPROACH

We propose a new model, the Pyramid Model, which provides support for users of development environments to be able to easily start with a simple application and change it to their needs (Figure 1). This is reminiscent of end-user programming, a field with a rich past (e.g., Nardi [4] and Goodell [2]).

This is a four-layer model or architecture. At its foundation is a set of "base products"—complete APIs for a diverse collection of computing environments, similar to Logo microworlds, but inclusive of both new commercial and open source APIs, such as the programming languages Python and Processing, as well as APIs for specific environments, such as geographic information systems, database search, telephony, Google maps, LabVIEW, online news, home monitoring, TiVo, mobile computing, and sensor networks. These APIs tend to be large and often very low level so as to be able to provide as wide a range of applicability as possible.

At the apex is a domain specific API (a microAPI) that is highly tuned for implementing a particular, focused, real-world application (in our research plan, this will be the radical product designed by the students). The microAPI contains 10 or so commands and few parameters. The layer supporting the microAPI consists of defaults assumed for the product thereby anticipating some parameters, but avoiding having the programmer deal with boundary and constraint checking.

The layer above the base product APIs represents the merger of all the base product functions minimally necessary to run the new product. The layer above that provides support for parameter changes to the default API.

The middle layers thus negotiate between the full APIs of these heterogeneous systems and the highly domain-specific, intentionally designed microAPI for the radical product. This architecture dramatically reduces the complexity of the base layer into a tiny set of commands that are easily composed into the functional application. Further, the microAPI at the pyramid's apex is generative, capable of easily implementing not only the radical product for which it was designed, but a range of others that were not anticipated.

This provides for ease in product support as well as in development. Our argument is that the small tailored product APIs are flexible enough to support a wide range of products and that the process can be repeated if necessary. The results however are wonderfully small microAPIs supporting end-user programming due to the simplicity and broadness of the APIs.

RESEARCH PLAN

We are pursuing the Radical Design approach by involving graduate students as co-researchers in successive offerings of the Radical Design course. Our goals are to (1) validate our new approach including the pyramid model and methods of software design, (2) create curriculum materials for teaching, (3) create genuinely useful Radical Products, and (4) create simple APIs (the microAPI) usable by non-programmers.

A new two-semester Radical Design course will be developed. The sequence will bring together students from computer science and other disciplines in the creation of these programming environments. In the first semester, student teams will design several highly innovative specific technology demos around a particular theme of interest (e.g., bioinformatics, GIS) but merging 5 to 6 base technologies from the collection described above.

During the second semester, teams will work from the prior sets of examples to extract fundamental principles or ingredients from each class of examples and use these as the basis of designing a language/environment toolkit for exploring the specific domain problem.

The students will start with disparate technologies and the goal of eventually creating a simple API to control them as a system. There will be 6 or 8 technologies from which to choose. Students may propose others which we will approve at our discretion. Each person on a team will become "intimate" with one of the technologies—and thus be that technology's expert. The group will then use the radical design methodologies to conceptualize and eventually realize a new product based on their combined technologies. The API designed will be a simple, small API with defaults supporting some generalization of their product's need.

AN EXAMPLE OF WHAT CAN BE BUILT

Suppose a group starts with online news, home monitoring, Google Maps, mobile computing, and sensor networks. After each student in the group becomes very familiar with at least one of the APIs for these base products, the process described above is used to conceptualize a new product.

While designing the product, a merger of the base APIs is undertaken with a minimalist's view. This helps firm up the underpinnings of the new product they are designing. Such a new product could be, for example, a telephone that senses its location and receives selected relevant news summaries based on its location and time with directions and satellite views of the news area. This would be a useful application for news reporters.

The students, having conceived of such a product, will design the microAPI (with UML diagrams to support their minimalist arguments). They will then implement the small API and demonstrate it with a working prototype. During the second semester, significant refinement and refactoring will go on with the students conceiving of several (five) other products, thereby highlighting that a small API (this microAPI) can be generalized. As an example of such alternative product, the telephone might send its location to another phone for tracking one's child or a product tracking system using RFID for the sensor. Finally the students will compare and discuss the microAPIs for all the products they conceived.

Note that the original prototypes may be thrown away, and the new microAPI will be used to re-implement the prototypes—ideally in something like 10 lines code per demo (each of which might have been a 1,000+ line program using current approaches). Then, this microAPI would be used by both the computer scientists (programmers) and the subject-matter experts (SMEs) to build many more interesting demos—especially ones that are of professional interest to the SMEs.

EVALUATION

These ideas are put forth as a bold new vision for software and product development. As we develop and conduct the Radical Design graduate course, we are interested in both the effectiveness of this software design model and the educational aspects of our work. In evaluating our work, we will ask the following:

1. Does the Pyramid Model provide a viable and attractive alternative to current methods of software-intensive product development?
2. Do the Pyramid Model and Radical Design techniques, used in combination, have the potential to create breakthrough products?
3. Are the apex APIs created with the Pyramid Model generative? Can they be ported from the original application space to new, unanticipated applications?
4. Are the apex APIs created usable by end-users to modify and potentially generate new applications?
5. Are the Pyramid Model and Radical Design teachable? If so, what supporting curriculum materials can be developed?

CONCLUSION

We address problems of complexity in software intensive systems with a combination of approaches coming from the product design fields. Our new architecture returns us to simplicity for coding. This is important with today's overly complex software environments, and will facilitate the development of many new software and hardware products.

The authors are scheduled to co-teach the Radical Design course, revised based on the ideas presented in this paper, in the Fall 2006 semester. We are looking forward to teaching and evaluating the course and to bringing these ideas to the larger design research community.

REFERENCES

1. Fry, B., Processing, http://processing.org/
2. Goodell, H. "End User Programming in an Industrial Research & Development Group," Tech report, http://csdl2.computer.org/comp/proceedings/hcc/2001/0474/00/04740234.pdf, 2001.
3. Myers, B. "Towards More Natural Functional Programming Languages," (invited keynote talk abstract). The Seventh ACM SIGPLAN Intl. Conference on Functional Programming, ICFP 2002. Pittsburgh, PA. p. 1.
4. Nardi B. "A Small Matter of Programming: Perspectives on End User Computing", The MIT Press, 1993, ISBN 0-262-14053-5