In Proceedings of the 11th IASTED International Conference
SOFTWARE ENGINEERING AND APPLICATIONS
November 19–21, 2007, Cambridge, MA USA
ISBN Hardcopy: 978-0-88986-705-5 / CD: 978-0-88986-706-2

A RADICAL DESIGN COURSE: LEVERAGING APIS FOR CREATIVITY AND INNOVATION IN SOFTWARE

Fred Martin and Georges Grinstein
Department of Computer Science
University of Massachusetts Lowell
Lowell, MA 01854 USA

Sarah Kuhn
Department of Regional Economic and Social Development
University of Massachusetts Lowell
Lowell, MA 01854 USA

ABSTRACT
Software design is becoming increasingly complex; yet great opportunity now exists with the proliferation of powerful APIs and other design toolkits. Here we describe an approach to software development that combines formal creativity methods with deliberative use of published APIs towards the development of a "micro-API"—a highly focused interface to a "radical product." We have developed our ideas in the context of a graduate seminar course. This paper presents the ideas we pursued—including a discussion of creativity in software design, and distinctions among API, language, and application framework—and results of student work.

KEY WORDS
education, software design, creativity, API

1 Introduction

In today's software world, it is no longer possible to build an entire system from scratch. Software engineers use rich application programming interfaces (APIs) that allow them to build sophisticated systems. With the power of these APIs, there is an explosion of possibilities in software design.

Still, innovation is often hampered by perceived, often conservative, beliefs. Creating revolutionary products, whether these products are consumer goods, processes, or software requires breaking conceptual boundaries.

We propose an approach that can support creative and innovative software development. Our approach combines deliberate use of creativity-enhancing processes, using existing APIs in unexpected ways, and an API-pyramid model of software design.

The ideas were tested in a seminar-style, project-based graduate course. We titled the course "radical design," a deliberately provocative phrase that represents our aspirations. In this paper, we present the approach, including results from this course.

2 The Pyramid Model and the micro-API

We propose a pyramid-based model to describe our approach to working with software and related technologies (Figure 1).

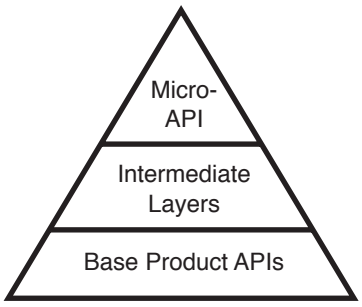


Figure 1. The micro-API pyramid model.

At its foundation is a set of "base products"—complete APIs for a diverse collection of computing environments. This includes programming languages, as well as APIs for specific environments (e.g., GIS, database search, telephony, video, Google maps, etc.).

At the apex is a domain specific API (a micro-API) that is highly tuned for implementing a particular, focused, real-world application (the "radical product"). The micro-API should contain 10 or so commands and few parameters. Ideally, the micro-API is scriptable, as an end-user programming language [7].

The middle layers thus negotiate between the full APIs of these heterogeneous systems and the highly domain-specific, intentionally designed micro-API for the radical product.

This architecture should dramatically reduce the complexity of the base layer into a tiny set of commands that are easily composed into the functional application. Further, the micro-API at the pyramid's apex should be generative, capable of easily implementing not only the radical product for which it was designed, but a range of others that were not anticipated.

3 Course Design

The model was tested in our graduate course, offered during the fall 2006 semester. The course drew mostly computer science students with a few students from other departments and one business professional.

The course had two distinct phases. In the first half of

the semester, we studied creativity and process methodologies. We developed our understanding of these approaches by putting them to use in the design of conceptual projects (which were proposed but not implemented). Each project was developed by a group of 4 to 5 students.

In the second half of the course, we embarked on hardware-software implementation projects. These were traditional student projects in the sense of being rapid-prototyping exercises (i.e., implementing a new project over a period of six weeks that could be viably demonstrated), but with two extra considerations: (1) the development processes were guided by the design and process discussions from the first half of the course, and (2) the projects were inspired by the formative hypothesis of the course—that combining disparate software APIs can lead to innovative software projects, and that a generative micro-API will emerge from this process.

4 Creativity and Radical Design

As the course got underway, we searched for a useful definition of creativity and our own notion of "radical design." Initially, we defined radical design as something which produces the "Wow!" reaction—you'll know it when you see it. This was partly borrowed from Goldenberg and Mazursky's definition: "a creative idea is an idea about which field experts agree that it is creative" [4] (p. 30).

Some of us were unsatisfied with this definition, which seemed to skirt the essence of creativity, avoiding "looking inside the box" of the ideation process. A person might experientially perceive a process as creative, but if the end result were not innovative to others, the process would not be rewarded. This led to a definition of creativity as an act that brings together two or more things that were previously considered unrelated.

This definition allowed us to recognize both creative acts and outcomes. An act might be creative, but the outcome might not be. With this in mind, we defined "radical design" as something which combined both—a creative process and an innovative outcome.

5 Creativity Methods and the Process Project

Broadly, we looked at three conceptual frames for understanding creativity: (a) specific processes and techniques, (b) community expectations, norms, and related environmental factors, and (c) the role of individuals.

5.1 Processes and Techniques

From a broad literature review, we focused on three particular points of study: the *Creativity Templates* approach developed by Goldenberg and Mazursky [4], the *Six Thinking Hats* method of Edward De Bono [3], and the IDEO Method Cards [5].

Creativity Templates are a set of approaches for systematically searching a design space, looking for unexpected affordances or ways of relaxing constraints. The techniques involve thinking abstractly about a product's qualities or properties, and the mechanisms for achieving these results. By considering these characteristics separately and systematically (e.g., as a 2D matrix or connected graph of relationships), one can dispassionately and comprehensively make analyses. The "creative" part emerges when the method leads you to consider implausible connections—as hidden opportunities.

Six Thinking Hats is a process approach to harnessing and focusing individual and group brainstorming sessions. The process recognizes that different emotional or cognitive mind-sets are valuable at different phases of a design or problem-solving process, and encourages participants to consciously "switch hats" during conceptual work.

De Bono proposes six particular "hats"—i.e., thinking modalities—each with an associated "color": neutral or fact-oriented (white hat), emotional or feeling-oriented (red hat), critical or judgmental (black hat), positive and optimistic (yellow hat), creative and innovative thinking (green hat), and process-oriented / meta-cognitive (blue hat).

In a session, a leader organizes the collective "switching of hats," suggesting that (e.g.) the group share its gut feelings on a matter by wearing the red hat (and thereby validating any emotion without the need for it to be rationally justified), brainstorm in new directions with the green hat, or voice concerns with the black hat. (For our seminar, we bought the set of six plastic hats offered by De Bono's consulting firm, which did have a fun, constructive effect on classroom work.)

We made use of the IDEO Method Cards, a collection of 51 large-format "playing cards" developed by the IDEO design firm. Organized into four categories—Learn, Look, Ask, Try—each card presents one design method used by the firm, along with a brief story about using it. For example, in a technique called "bodystorming," the designer "sets up a scenario and acts out roles, with or without props, focusing on the intuitive responses prompted by the physical environment." This method "helps to quickly generate and test many context and behavior-based concepts."

In another technique called "affinity diagrams," designers "cluster design elements according to intuitive relationships such as similarity, dependence, [or] proximity." This method is like a less-structured version of the creativity templates mentioned earlier.

5.2 Cultural Norms

Not everyone agreed on the focus on formal methods as being central in understanding and encouraging creativity. In addition to using the Method Cards, we read Tom Kelley's book on the company [6], and viewed the ABC Nightline episode which documented the company's 1-week "deep dive" process of re-conceptualizing the common grocery store shopping cart [1].

While it was clear that IDEO's designers make extensive use of the methods and processes they have documented in the Cards, we also believed that cultures of practice established within the company play a substantial role in its success. These cultural expectations encourage individuals to constantly stretch themselves and contribute to each others' thinking. As explained by Kelley, "There are specific elements which we believe will help you and your company to be more innovative. But it's not a matter of simply following directions. Our 'secret formula' is actually not very formulaic. It's a blend of methodologies, work practices, culture, and infrastructure." [6] (p. 5)

Kelley further describes the effect that the company's culture and practice has on its individual members: "[Other companies] tend to believe that truly creative individuals are few and far between. We believe the opposite. We *all* have a creative side, and it can flourish if you spawn a culture to encourage it, one that embraces risks and wild ideas and tolerates the occasional failure." (*ibid.*, p. 13)

This intersection of culture and practice was further illuminated by an online essay recently published by a Google engineer [8]. The essay combines a criticism of reflexive adoption of agile design processes with a discussion of cultural practices inside of Google. These include the well-known 20% of time on side projects, managers who are also half-time coders, and a sustainable work pace. But he also describes a "peer-review oriented culture," where individuals care about earning each other's respect, and disciplined coding practices, so that code bases are easily (and routinely) shared across the company.

In both the IDEO case and the Google case, the overall picture is certainly one where the whole is more than the sum of its parts. Each of the practices or methods is useful, but as a coherent whole, represents an organization that is creative and productive as a matter of routine.

This realization encouraged us to modify the pedagogy of course. While we couldn't hope to become an IDEO or Google (in one semester), we did strive to establish a discussion-oriented and dissent-permitting process, both our whole group setting (the weekly seminar meeting) and in project teams. Even if the culture of our whole academic department could not readily be transformed, we strived to have this happen in the microcosm of the course.

6 Concept Projects

To engaging with the creativity methods, we organized concept projects during the first half of the semester. Each course participant identified an existing "radical product"—an invention that has had a significant impact on society, and one that was also personally interesting.

Participants identified the product, described its significance, and proposed some "radical extensions" that could be taken. The list of products included headphones, insulation, microbes, the internet itself, the Theremin, the toilet, wearable computing input devices, smart showerhead units, hearing aids, bean bag chairs, flexible solar panels, chairs, and the iPod.

Students pursued a deeper understanding of their product using techniques from the Method Cards (choosing one each from the Learn, Look, Ask, and Try categories) and using one of the Thinking Hats.

Inspired by the IDEO methods, we organized a process to form a smaller number of student teams from the full set of radical products. We listed the complete set of products on the wall, identifying each product on a single sheet of poster-size paper. We used the Hats method to encourage comments on each product, and asked each student to vote the three different products he or she found most intriguing. This was done in parallel, with students applying sticky-notes to the poster sheets.

After the voting was accomplished, six of the products were clear leaders. We quickly realized that these six could be grouped into three categories: Hearing Devices (headphones; hearing aids), Toilet/Bathroom (toilet; shower), and Furniture (bean bag chair; chair).

At this point, a participant suggested we should "vote with our feet" and simply stand next to the product he or she was interested in working on. This shortly led to the formation of three similarly-sized working groups of 4 to 5 students each; these teams stayed together for the next several weeks of the semester. Each pursued the creativity and design methods, produced working documents on a course Wiki, and participated in full-class debrief sessions.

Teams performed a competitive product survey (learn), a still photo survey (listen), "extreme user" interview (ask), and bodystorming (try), used the Goldenberg/Mazursky templates approach to generate unexpected extensions of their product, and prepared a final report, as a written document. The final product concepts were imaginative and potentially commercializable:

- The Toilet group proposed the development of integrated, seamless, modular entire bathroom units, expanding and combining their initial products (toilet and showerhead) into a holistic hygenic experience.
- The Hearing Devices group proposed a hearing aid that also served as a fashion accessory for normal-hearing people (inspired by the now-ubiquitous Bluetooth earpieces), but used off-board DSP-based audio processing capability for enhanced performance. (We later learned this idea is being developed by an accomplished design firm.)
- The Workstation group proposed design of computer and desk furniture that is highly and dynamically reconfigurable, to adapt easily to people's desire to customize their work environments.

At the end of the concept-product phase of the course, we were impressed with the potential of the creativity methods, had built a culture in the class where public participation was commonplace, and had come up with some

interesting ideas. We were ready to work on actual software, where practical implementation constraints would more readily assert themselves.

7 Implementation Projects

To introduce the implementation work, we restated the hypothesis which framed the concept of the course: that radical (innovative; extensible) sofware projects can arise from combinations of disparate software APIs, particularly when the resulting software product is restated as its own, new "micro-API."

7.1 Introducing APIs

Initially, we thought it would be unproblematic to create a working definition of an API. We defined API as an abstract interface to an application or system, available to programmers to request services, data, operations, responses. The API or library may have several implementations, and is often distributed as part of a software development kit (SDK). Its interface may include functions, variables, and data. In short, in the words of Joshua Bloch, "An API should do one thing and do it well" [2].

Since the beginning of the semester, we had begun accumulating a list of APIs on our course Wiki. From this list, we made an in-class presentation of a small but diverse collection of examples, including both software-only and hardware/software systems: OpenGL, Lucene, OpenHaptics, Google Maps, Google Calendar, TiVo Home Media Engine, the Player/Stage robotics package, and AppleScript. For each API, we summarized its function and provided a snippet of sample code.

7.2 Rapid API Mashups

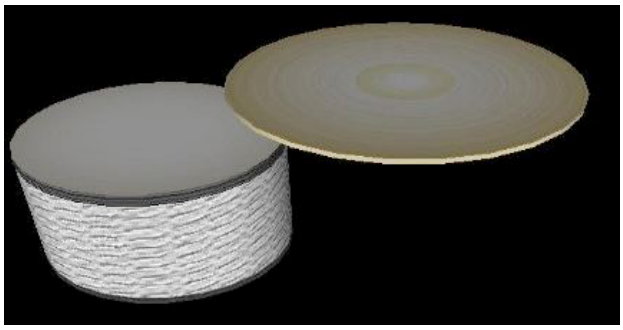
After this introduction, we handed out index cards that had been pre-printed with titles of the various APIs we had described. Each student got a different API, and we randomly sorted the students together into small groups. Each group performed a short, 5-minute brainstorming session to come up a software application that combined all of the APIs. We ran the activity three times in succession.

There were some humorous examples as well as some intriguing ones. A group with a voice recognition API, a physical sensor interface API, OpenGL, and a local-file search API invented a hands-free coroner examination tool.

7.3 What is an API?

For the following class session, students were to each choose two APIs (either already on the master list, or they could add them) and prepare 3-minute brief overviews of each and a live demonstration of one.

At this session, a more complex picture of the nature and structure of APIs emerged. The debate first opened



Init	SetMusicalProducer
SetScale	CreateMusicalProducer
SetMalletSize	SetMusicalMaterial
SetMalletMaterial	DestroyMusicalObject
GetMalletVelocity	Close

Figure 2. GraHapSo drum/cymbol and micro-API.

around PostScript: should this be considered an API (as in a page-description toolkit) or is it really a language? Does it depend on how you use it? In addition to the classic library-with-a-set-of-function-calls model, we observed at least two other categories for the systems we had put on our API list:

- *Pure languages vs. scripting languages*. In addition to PostScript, we had many other examples of programming languages, including PHP, Perl, and JavaScript, as well as systems languages like AppleScript.
- *Application frameworks*. A number of systems seemed more properly characterized as *extensible application frameworks*, perhaps with some kind of "pluggable" architecture. Examples of these were the Apache web server, Player/Stage, Mozilla, and Eclipse.

Further, we underestimated the domain-specificity of each technology. Many APIs were only implemented on one operating system platform (and/or existed only with a particular application framework). Even well-supported, multi-platform APIs had significant language-related implementation dependences. In principle it is straightforward to make calls to a C library from within a Java application, but in practice, it is complex.

7.4 The API Groups

In a similar fashion to the way we fostered the formation of the concept-project teams, we allowed students to self-select implementation-project teams. We gave the guidance that we wanted to keep the teams small, with a maximum of four students per team. Students self-organized around a combination of APIs they already knew and were interested in working on, producing five project teams. Here, we highlight the work of two of the five teams.

7.5 GraHapSo Group

Two of the members of the prior Hearing Devices group continued to work together on a haptics-based project. Both had extensive experience with a commercial haptics product—one as a company founder, and the other as a graduate student and programmer.

For the new project, they looked to combine the haptics technology, with which they were quite familiar, with several other APIs, including sound synthesis, a simulation of materials physics that produces audio energy, and a blogging API. The resulting product would be a palpable physical simulation of musical instrumentation, including visual and tangible modes, as well as an adjoining teaching and learning interaction space for students to use in a physics/music class.

When the idea was presented in class, the general reaction was extremely supportive, with the observation that they were proposing two significant systems—the physics-haptics-audio simulator, and the teaching/learning subsystem. We recommended that they focus on one of the two. Naturally, they choose the physical simulation.

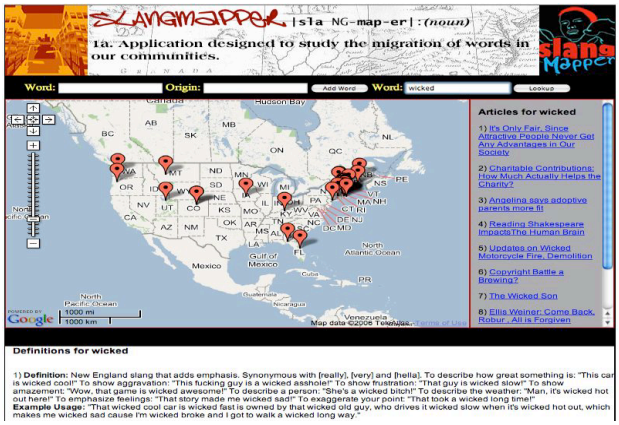
As work progressed, it became evident that a low-level simulation of physical resonance of material such as wood or metal would be quite difficult to obtain. None of the available physics simulator APIs seemed to have anything resembling this functionality. Eventually, the team scaled back its goals, and settled on an parameter-based soundwave synthesis algorithm that could be adjusted to produce a variety of sounds, including bell-like, glass-like, and wood-like tones.

Figure 2 shows a screensnap of one of the team's early prototypes—dubbed "GraHapSo" (graphics + haptics + sound). The drum and cymbal exist as OpenGL graphic objects *and* as OpenHaptics palpable objects. A mallet (not visible in the screensnap) can be used to strike either object; when this is done, the user feels the resistance of the object from the physical haptics interface, and hears the resulting sound, as generated by the audio synthesis.

The group then developed a micro-API which they used in the final version of their implementation (Figure 2). Calls in the micro-API initialized their entire system and allowed placement of different simple geometric solids in the virtual space. The solids could be of different simulated material (wood, metal, or glass) and could be various size and position; these parameters fed into the audio synthesis to produce a corresponding pitch and sound quality. The final demonstration was of a xylophone of a dozen slabs of appropriate size to play a musical scale.

7.6 SlangMapper Group

The core of group was three advanced programmers. They had explored Google Maps and Google Calendar for the intro-APIs exercise, and had noted the prevalence of web-based mashups that included Google Maps, thanks in part to the generally agreed-upon format for sharing addresses.




```
APIinit()
addEntry(term, location, paramList)
lookUpByLocation(location, paramList)
printMap()
APIutilities.php file (edit to set startup parameters)
```

Figure 3. SlangMapper system and associated micro-API.

One member of the group had an interest in linguistics, and the team coalesced around a project that became called "SlangMapper." The following narrative, prepared at the beginning of the six-week implementation, accurately captures the final product:

> The current idea is a linguistics website for tracking slang words, their origin and their usage. Our goal is to track, verify and record slang across the United States. The primary tools will be the Urban Dictionary (lookup and verification), Google Maps (location tracking and display), MySQL (recording and searching) and findory (RSS feeds to find occurences). . . . It's our hope that a community of users can establish patterns of use through updating current slang being used.

Figure 3 illustrates the system in action. First, the system's database is populated with slang words and their corresponding locations. The system implicitly trusts users to enter accurate mappings of words to locations, but validates words (using the Urban Dictionary) before a word is added to its database.

In the screensnap, the user has requested a lookup of the word "wicked" (a common New England slang word). The system's database has already been populated with sample locales where this word has been heard; these cluster around the Northeast region of the United States. To the right of the map, recent blog entries that have employed the word are excerpted. Below the map, the definition of the word (as obtained from the Urban Dictionary) is displayed.

While this tool seems highly specific, the team had kept the micro-API goal in the front and center during their design process (Figure 3). The key subsystems—the authentication engine (e.g., Urban Dictionary) and the blog

feed (e.g., Findory) can be trivially replaced with other sources. After implementation, the designers describe their system:

> [Our project is] a framework for the validation, collection and display of user defined topics. Our example application will take user-supplied slang words and their usage location. It then validates the information, adds it to the map and displays relevant articles on the word.
>
> The generalized framework includes: A validation engine, a user based database and reference sources (such as articles, blogs, RSS feeds). It is our goal to allow users to track a variety of topics, their location and any recent information.

In exploring this generality, the team proposed two other specific applications of their system: (1) a disease-tracking tool, where users could enter particular bugs and locations where they had been observed, and (2) a bird-watching database, with corresponding functionality. In thinking about future extensions of the project, the team proposed a time-lapse visualization mode, which would display the spread of a database entry throughout geographic space over time.

8 Conclusions and Future Work

One of the unusual aspects of our work is the integration of the creativity focusing techniques (e.g., templates and hats) into the software design process. While it is difficult to be scientific in an analysis of their impact, in general the approaches caused us to be more deliberative and reflective in our design processes. For example, a member of the SlangMapper team commented, "Although we did not say 'let's use the white hat now,' we certainly were more aware of how we communicated with one another and how to more efficiently present our ideas to each other."

Another student related impact of the techniques on projects outside of the course context:

> For people in Department of Health and Safety, I maintain a table form that is a list of disposable chemicals. Users can check-mark Yes, No, and N/A options, and I was supposed to maintain a matrix for insertion of the data and for the retrieval of the same. Here I used the Template concept, and used the technique of grouping and replacing the unnecessary columns in the table.

Another student described how the Replacement Template was used to generalize his micro-API:

> This redeployment is basically the same product as the original vision, except we have thrown out the email aspect of the micro-API. In place of reading mails from a specified email account, we instead read text files from a specified local directory (essentially the Replacement Template from Goldenburg and Mazursky)

We do perceive a growing pervasiveness and power of APIs and need for software developers to become facile in using them. Students have also commented on the value of this orientation in their projects outside of the class.

We anticipate that a thorough vetting of APIs for compatibility, usefulness, and ease of use would produce substantially better results. This could partly be done by ourselves and graduate researchers before the next iteration of our course; also, we could allocate more time for students to become experienced with a single API before asking them to combine them.

As to the core assertion of our hypothesis—generative power of micro-API—we achieved mixed results. Certainly, teams that had the micro-API in mind as they progressed through their design process were more successful.

We do see that the power of the micro-API will only be revealed when it is itself embedded in an easy-to-use scripting environment. This will allow for true end-user programmability, ideally by persons other than full-time software developers.

While this work is not a controlled study, we believe that creativity techniques caused our students to consider alternatives that otherwise would have been overlooked. We are optimistic that micro-APIs present an approach to the growing complexity of programming systems today. Our goal is powerful software that is easily re-configurable and adaptable. We are confident that the Radical Design approach is a significant step in the right direction.

9 ACKNOWLEDGMENT

We would like to thank our students for their creativity and willingness to explore these ideas together, and we wish to thank our respective department heads for allowing them to co-teach the Fall 2006 Radical Design course and have it fully count toward our teaching loads.

References

[1] ABC News. *Nightline: The Deep Dive*. abcnewsstore.com, Howell, MI, 1999.

[2] Bloch, J. How to design a good API and why it matters. In *Conference on Object Oriented Programming Systems Languages and Applications (OOPSLA)*. Portland, Oregon, 2006, 506–507.

[3] De Bono, E. *Six Thinking Hats*. Back Bay Books, 1999.

[4] Goldenberg, J. and Mazursky, D. *Creativity in Product Innovation*. Cambridge University Press, Cambridge, UK, 2002.

[5] IDEO. *IDEO Method Cards*. William Stout Architecture, San Francisco, CA, 2003.

[6] Kelley, T. *The Art of Innovation: Lessons in Creativity from IDEO*. Doubleday, New York, NY, 2001.

[7] Nardi, B. A. *A Small Matter of Programming: Perspectives on End User Computing*. The MIT Press, Cambridge, MA, 1993.

[8] Yegge, S. Good agile, bad agile. `steve-yegge.blogspot.com/2006/09/good-agile-bad-agile_27.html`.