Embedding a Design Studio Course in a Conventional Computer Science Program

Saul Greenberg
Department of Computer Science, University of Calgary
Calgary, Alberta Canada T2N 1N4
http://www.cpsc.ucalgary.ca/~saul/

Abstract. Within undergraduate Computer Science, Human Computer Interaction is often considered a blend of user-centered requirements analysis, design, implementation and evaluation. While most are teachable within the constraints of a conventional undergraduate lecture course, design is much more difficult to pass on. We know that design-oriented programs (e.g., arts, industrial design, and architecture) teach design practice as arising from the culture of a design studio. The problem is: how can we pass on the best practices of design studios within traditional programs that follow a standard lecture/tutorial format? My solution was to create a design studio atmosphere within a lecture/tutorial time-frame. Over the semester, students are introduced to four quite different state-of-the-art interaction domains, each chosen to minimize students' pre-conceived notions of what comprises a 'standard' design within these domains. They are given substantial freedom to design projects within these domains. They are required to sketch out their ideas and publicly show these sketches to other classmates for critique. Idea exchange is encouraged, where classmates can use parts of each other's ideas in their own work (conventional courses call this 'cheating'). Many lectures are replaced by studio work where students develop their designs during class time. Thus students and instructors see each other's work as it is being develop, they share tricks and techniques, and they engage in on-going commentaries. Students demonstrate final projects publicly within a design critique setting. Finally, every student has to create learning and professional portfolios illustrating their work using a mix of paper and electronic mediums.

1 Introduction

Human Computer Interaction (HCI) undergraduate education is expanding far beyond what it used to be. Teaching HCI was once constrained to a single class module or (if lucky) a single junior or senior-level course within computer science or psychology curriculum. Now, HCI programs are emerging that are centered on

Cite as:
Greenberg, S. (2007) Embedding a Design Studio Course in a Conventional Computer Science Program. Report 2007-870-22, Dept. Computer Science, University of Calgary, Calgary, Alberta, Canada. T2N 1N4. June.

training students to become HCI professionals. Such programs are often inter-disciplinary, where they train students through a balanced pedagogy including design, engineering, evaluation, requirements engineering, and business practices.

While these programs point the way, the reality is that they are few and far between. Most are offered as post-graduate professional programs at prestigious institutes (e.g., CMU's Human-Computer Interaction Institute; Stanford's D-School Institute of Design). A few institutions have similar offerings available at the undergraduate level, usually as specialized interaction design programs outside of Computer Science departments (e.g., University of Queensland's Information Environments program [4]; Simon Fraser University's School of Interactive Arts and Technology). Most programs, whether undergraduate or graduate, are tailored to students who want to be HCI professionals or information architects. They do not help the vast masses of computer science students who plan to be software engineers but who could still benefit from strong HCI training.

Thus for most computer scientists, HCI is taught as a mix of user-centered requirements analysis, design, implementation and evaluation within the constraints of a conventional undergraduate lecture course. While many of these topics are amenable to the lecture setting [7], the subject of design is much more difficult to handle in a lecture setting. Quoting Reimer and Douglas [15]:

> Traditionally, HCI design has been taught as an abstract process of iterative user-centered design with a recommended set of design aids such as task analysis, GOMS, guidelines, heuristic evaluation and usability testing. During a typical HCI course there might be an occasional practice exercise for the student to evaluate and even improve, for example, an interface with poor usability. Adding a team-based final project to design, implement and evaluate a working interface provides more experience in actual HCI design, but still falls short of teaching the student good design of a real-world artifact while engaging in a real-world design process. In these courses we leave it to faith that students will be able to make the transition from theory to practice. [15, p192].

Indeed, the whole process of generating and developing interesting design ideas within an HCI course structure is often at odds with usability and evaluation criteria. For example, creative design in HCI is often taught 'by example': students are introduced to many interesting and novel interaction techniques generated by others (e.g., I teach a long module on information visualization methods to encourage students to design interfaces using these methods [7]). Yet in my experiences, students who apply some of these novel methods to their initial project ideas tend to put them aside in favor of a traditional interface. This is because the many usability principles they are taught suggest that people will fare better with interfaces that are familiar to them, e.g., through positive transfer and learnability. As well, students consider unusual interfaces as 'riskier' than conventional interfaces in terms of how they would fair in a usability evaluation – and many instructors use usability evaluation as a criterion of design competence. For example, I have seen quite a few students suggest initial designs based upon information visualization techniques, and then discard these designs because they were concerned they would not do as well as (say) a traditional database table interface. And they are probably correct, for

relatively small factors in novel designs – factors that could perhaps be corrected over time – could have a profound effect on people's performance. Thus students tend to follow safe conventional designs rather than push the design envelope.

The question is: how can we engage computer science students in the practice of design by creating an environment that is conducive to it? The answer may seem obvious. We know that design-oriented programs (e.g., arts, industrial design, and architecture) teach design practice as arising from the culture of a design studio [11], and this practice can apply to HCI as well [18,1,2]. In related work, Reimer and Douglas [15] nicely summarize the design studio pedagogical model as: learning by doing through an experiential pedagogy and by using teachers as resources; integration of theoretical knowledge; creating realistic artifacts through a professional design process where these artifacts are the primary basis of assessment; and primary teaching through a design critique – *aka* design crit. They also elaborate several important properties of the studio environment that helps realize this pedagogy. These include specialized studio rooms that are inhabited by the student peer group for substantial periods of time. In turn, the studio becomes a place where students do their on-going designs, where students constantly see each other's work, where they communicate to each other, and where they share practices. This leads to a communal design input, design reflection, and on-going design critique.

Given that the design studio is key to nurturing design pedagogy, the real problem facing computer science educators is: how can we pass on the best practices of design studios within traditional programs that follow a standard lecture/tutorial format and classroom constraints?

This paper describes my solution to this problem: how to create a design studio atmosphere for computer science students within a traditional lecture/tutorial timeframe and limited workroom availability. Of course, others have also tackled this problem either in courses or by proposing design methodologies [e.g., 4,15,10,18; see also other publications in this book], but each in their own way. Indeed, a recent ACM CHI Workshop had as its theme the question of how one supports a design studio culture in HCI [1]. I don't argue that the course described below is 'better'. Rather, it complements and overlaps other approaches, where it adds to the richness of possibilities that course designers can choose from.

The purpose of my studio course is to have students experience the 'best practices' of design. In parallel, they learn several emerging areas within Human Computer Interaction, and become well versed with tools that let them develop prototypes within those areas. The gist of the course is summarized below.

- The course repackages the standard Computer Science lecture/tutorial time slots into a single 'studio' time slot.
- It creates a temporary studio by having all classes in a computer-equipped classroom, with at least one computer per student.
- Over the semester, students are introduced to four quite different state-of-the-art interaction domains. Each domain is chosen to minimize students' pre-conceived notions of what comprises a 'standard' design within these domains.
- They are given substantial freedom to propose design projects within these domains.

- They are required to sketch out a multitude of ideas, and publicly show chosen sketches to other classmates for critique.
- Idea exchange is encouraged, where classmates can use parts of each other's ideas in their own work (conventional courses call this 'cheating').
- Many lectures are replaced by studio work, where students develop their designs during class time. Thus students and instructors see each other's work as it is being develop, they share tricks and techniques, and they engage in on-going commentaries.
- Students quickly learn and use specialized software toolkits that let them rapidly prototype and implement their design ideas within each domain.
- Students demonstrate final projects publicly within a design crit setting.
- Every student has to create a portfolio illustrating their work using a variety of formats (e.g., paper, poster, video, web).

Details are described below. I begin by describing the basic course structure. I then elaborate on the primary pedagogical artifacts used in this course: the design projects, the sketchbook, and the portfolio. I close by summarizing student outcomes that I have seen after teaching this course for several years.

2 Course Structure

Making a design studio concept work within the constraints of a conventional computer science program structure is challenging but quite doable. I begin with a discussion of how I created and scheduled the studio and class size, and then move onto the life cycle of student projects that formed the student design activity. The context is that this is the student's second course in HCI; the first has already provided the basic background to HCI [7].

2.1 Scheduling

The first issue is scheduling: how can we get sufficient continuous contact time, where all students are working together? At our university, a normal course is typically taught by the instructor through either three 50 minute or two 75 minute lectures per week. These lectures are augmented by two 50 minute tutorial sessions – often scheduled at odd times of the day so as not to conflict with other lectures – and usually run by the teaching assistant. Following conventional practice would restrict the 'studio' time to just the tutorial sessions, which is clearly insufficient. What I did was schedule a 75 minute and 50 tutorial back to back (with a short 15 minute break in-between), thus forming two 2 ½ hour slots per week as the studio time. While administrators thought it odd that I would request the two together, the primary challenge was to find a 2 ½ hour block of time that did not conflict with other courses that students would likely want to take.

Of course, 5 hours of studio time per week is still far short of the much longer studio times scheduled in design disciplines (e.g., the introductory design studio course in our industrial design department is 4 hours / day where 4 days a week is not unusual). Still, 5 hours is much better than the standard computer science tutorial time offerings.

2.2 The Studio

The second issue is how to create a physical design studio. This is more challenging, for there were no dedicated facilities at my disposal that I could take over for exclusive student use. Instead, I booked one of our instructional computer workrooms (normally open to all computer science students to do their assignments, where a teaching assistant may give an occasional presentation) for use as our pseudo-studio. This again required just a bit of scheduling. Consequently, all contact hours were spent in a computer workroom. Thus during the 2 ½ hour classes, each student had their own computer, and no outside students were allowed in the room. Students also tended to linger after class (or come to it before class) to continue their work.

This room was already instrumented with a screen and projector connected to a single computer. I set up additional wiring (just a second long monitor cable) so that I could quickly plug the projector into any student's workstation as well. This meant that, at any time, any student could project his or her work to the class.

2.3 Numbers

The third issue is student numbers. Normally, class attendance is open-ended, and it would be easy to admit too many students to make a studio-based approach unworkable. I set a cap of 15 students. This sufficed to form a critical mass, but was sufficiently small to encourage cross-student interaction and to make sure that all students could demonstrate their work within the 2 ½ hour studio blocks. A class population of 15 was also considered large enough not to 'raise eyebrows' at the administration level.

2.4 Design Projects

The heart of the course is a series of 4 design projects, each within a different interaction domain (see §3). While students were encouraged to consult with one another, each student had to design, implement and demonstrate their own individual project. Each would be graded independently on their project design.

An issue is how to get students to think about these projects as design exercises. To set the scene, students were given fairly open ended project exercises within new and unfamiliar interaction domains. They were encouraged to be highly imaginative in what they created – artistic, speculative, and entertainment projects were valued as much as a project oriented toward work productivity. For example, the project description below is for designing within the domain of single display groupware.

> You have been hired to create a demonstration of a single display groupware (SDG) system that allows 2 to 4 people to interact over a single display using multiple mice and (optionally) multiple keyboards. Your demonstration should illustrate at least one object that gracefully reacts to multiple people using it simultaneously, which in turn is embedded in an application that exploits it. You have complete freedom of your design, as long as you can show that the SDG object and its containing application are useful for its intended audience, and that its design is somewhat impressive.

2.5 Software as New Media

As will be detailed in §3, each design project required students to work with different specialized media, which in turn allowed them to rapidly develop prototypes in different design domains. This new media came in the form of state of the art software toolkits. I pre-installed all specialized software and toolkits on the computer lab computers, so that students did not have to waste time configuring the system. I also recognized that while many students would continue to work in this studio space outside of class time, many would also work at home. Thus I made sure that all software was available for download, and instructed them on how to transport their own software between home and work. As well, they could optionally install this software on their own laptops and use that as their primary machine.

As a cautionary tale, installing non-conventional software on department computers was no easy matter. At our site, computers are 'locked down' for security purposes, where they restrict how software can be installed and how they are used. Yet much of the new software pushed the security limits of our department computers. Our technical support people had to go out of their way to make the software work, and I had to do extensive testing to make sure the software worked as it should. In contrast, students found it trivial to install the software on their home machines, as these rarely have the same level of security management.

2.6 Project Design Cycle

Students then went through the following cycle for each of the 4 project domains.

1. **Week 1, 1st class:** ***background and example of the design interaction area***. Through a lecture, students were introduced to the interaction area. The lecture typically provided conceptual foundations to the area, and then walked through many, many examples of designs produced within this area. Designs were illustrated by photos, videos, and running demonstrations. The student's homework was to come up with their own ideas of a design within this area, where they had to produce a variety of sketches (see §4 - Sketchbook).
2. **Week 1, 2nd class:** ***sketch presentation and toolkit introduction***. At the beginning of class, I passed around a digital camera. Each student was asked to select and take a photo of their 'best' idea in the sketchbook. I then projected these sketches, and each student was given about five minutes to explain their sketch. Every student had at least two other students critique their idea - we seeded this by asking observing students to say the best and worst thing about the design, and suggestions for improving it. To encourage idea sharing, I told students to copy design features that they liked into their own sketchpad (with attribution), which they could then use in their own designs. The second part of the class was an introduction to the toolkit. This was a hands-on tutorial where we walked students through a simple 'hello world' style example. Students were lock-stepped; all had to master a step before we went on. Thus at the end of the class, students had seen each other's ideas, and had a basic understanding of the toolkit capabilities. The students' homework was to modify their design based on what they had seen others do, and to leverage both the opportunities and limitations of the toolkit.

3. **Week 2, 3rd class:** ***hands-on in-depth toolkit laboratory, and design elaborations.*** The tutorial on the toolkit continued, where more sophisticated examples were demonstrated. Again, all students worked through these examples in lock-step. Afterwards, students were asked to describe their design changes. This sometimes happened in a public setting, or by breaking students up into small discussion groups, or one-on-one with the instructor and/or teaching assistant, or a mix of all approaches. If time allowed, students started programming their design. The students' homework was to work on their design and implementation.
4. **Week 2, 4th class to Week 3, 6th class:** ***design elaboration and implementation***. Students worked on their designs in class. While many were also programming their systems outside the class, they had to have a working copy that they could bring into class to continue their work. During this time, students were encouraged to show their on-going work to the instructor and teaching assistant (who were constantly walking around) and to other students, and more importantly to engage in discussion and critique. They were also encouraged to share technical 'tricks' with one another, i.e., on how they mastered the new media through the toolkit. This usually happened when one person was showing their work to others, who would then ask how they managed to implement a particular feature. When that feature looked generally valuable, we would plug the projector into that student's computer, who would then show the class how to do it. The students' homework was to continue the design, and to begin developing their web-based and paper-based portfolio description of their design (§5 Portfolio).
5. **Week 4, 7th class:** ***demonstrations*****.** Students would demonstrate their final system to the entire class within a design crit structure. Each student would typically start with the portfolio entry on the web site, then move onto a live demonstration. At the same time, the paper-based portfolio entry would be passed around. After the demonstration, all students were expected to participate in the 'crit' of the work, emphasizing its positive aspects and potential flaws, and how to improve the design.

Interspersed between classes were other events to encourage design skills. These included rapid-fire design exercises where students (sometimes in groups) would have to create a design that solved a specific problem. It also included discussions, applications and examples of best practices, e.g., in the use of a sketchbook, in portfolio construction, in lateral thinking, in rapid prototyping methods.

3 Selecting Design Domains

Over the semester, students have to sketch (§4), design and implement, demonstrate and record via a portfolio (§5) four projects, each in quite different areas of interaction design, following a particular design process (§3). I choose project areas around the following criteria, which I believe makes them well-suited for beginning computer science designers.

3.1 Novel interaction paradigms, rather than user-centered problems.

Projects are centered on particular interaction paradigms, rather than user-oriented problems. This may seem at odds with an HCI course. However, recall that this is a second HCI course; in the first course, students had already experienced the pedagogy of developing a system based upon a user-centered requirements analysis [7]. By having projects fashioned around a novel interaction paradigm, I believe that students would be more willing to pursue risky designs as a way of exploring that interaction space. At the same time, students would be introduced to an emerging area of HCI.

3.2 Personally unfamiliar interaction area.

I choose interaction paradigms that are emerging but not yet in every-day common usage. Thus students likely have little prior exposure to commercial systems in the project area. My expectation is that students are much more likely to come up with their own novel design, as they cannot fall back on standard solutions that they see in routine use.

3.3 Engaging and futuristic.

To motivate students, each interaction paradigm has to be in an engaging area, i.e., as an emerging paradigm in HCI and/or as a research area. Motivation is very important: when undergraduate students believe they are working at the frontiers of computer science, they can do amazing things.

3.4 Availability of rapid development tools within that area.

I strongly believe that students are heavily influenced by their development tools. The offerings of software toolkits are akin to media in the arts: glazes in pottery, paints in painting, wood and tools in woodcrafts. Students learn to think in terms of the affordances of the media, where the media suggests solutions to particular problems. I further argue about the importance of toolkits in fostering creativity in greater detail in [5].

Thus I select interaction areas with the proviso that rapid prototyping and development tools are readily available. These tools must be in the form where they are easily and quickly learnt. While computer science students are gifted programmers, I did not want them to spend their time and effort working on low-level implementation details, figuring out complex APIs, or untangling cryptic documentation. Instead, I wanted to give students tools as media, whereby simple design ideas are actually simple to do, and hard design ideas are possible [5]. As a consequence of working with this media, students could spend their time thinking about the design rather than programming, and could also rapidly iterate over their designs as they reflected on it, and as others critiqued its early versions.

3.5 Tools work within a familiar and rich programming environment.

Related to the above, I wanted to select tools that worked (as much as possible) within students' existing programming environments and development paradigms. Again, this is because I wanted students to concentrate on their designs rather than spend time learning a new programming language or IDE. Similarly, I wanted them to develop systems within a commercial quality programming environment so they

would have standard high-quality development tools available, such as debuggers and structured programming editors.

2.6 Example domains

To illustrate how this works in practice, I chose the following four interaction domains last year. Within this domain set, student prototyping and implementation was supported by rapid prototyping tools created in our own laboratory[1].

Single Display Groupware is a domain where multiple people can work together over a single display using multiple mice and keyboards. We gave students the SDG Toolkit [16,17], which makes it extremely easy to capture input from multiple mice and keyboards, to create multi-user aware widgets, and to draw multiple cursors on the display.

Physical User Interfaces is a domain where people interact with computer-controlled devices: tangible media, ambient displays, sensor-driven ubiquitous computing environments, and so on. We gave students Phidgets hardware (available at http://www.phidgets.com/) and our version of the Phidgets Toolkit [8,12]. The Phidgets toolkit makes it very easy to capture data from the input devices, and to control output devices. Phidget hardware at their disposal included servo motors, RFID tag readers, environmental sensors (heat, light, motion…), controls (switches, buttons, sliders), LEDs, and so on.

Distortion-oriented information visualization is a domain where people interactively explore visualizations presented in a distorted space (e.g., fisheye views). We gave students the Elastic Presentation Space (EPS) toolkit, a framework for designing such distorted visualizations [3]. It is a mathematical toolkit: one sets properties of a mathematical space (such as lenses, their shapes, their extents and so on), gives it a point in that space, and gets back where that point is located in the new distorted space. Thus students can experiment with quite different ways of mapping and visualizing information within a distorted space.

Groupware affording small group casual interaction over rich media is a domain where small groups of collaborators engage with one another in a public virtual space. We gave students the Community Bar as a groupware platform [13], and the Media Item Toolkit to actually develop a groupware media item within that platform [14]. Students create interactive multi-media items whose contents are broadcast to others. The toolkit is based on the idea of plug-ins: developers build the multimedia groupware component using a well-defined interface, and that component is then inserted into a pre-existing groupware architecture and system. Thus students can experiment with quite different interfaces of interest to the group, where their items lives in ecology of the Community Bar standard offerings as well as items created by other students.

Of course, these projects are just a vehicle where the goal is to train students on best practices in design. We already described in §2 how students developed their designs as part of the design studio practice. In the next two sections, we turn to two other valuable artifacts supporting design activity: the sketchbook and the portfolio.

[1] All tools are available at http://grouplab.cpsc.ucalgary.ca/cookbook/, including documentation and tutorials, and all work within C# and Microsoft Visual Studio.

4 The Sketchbook

The sketchbook is perhaps the most prevalent best practice artifact found across all design disciplines. Many designers keep a sketchbook with them at all times. They use it to record and elaborate their ideas, to gather other people's ideas or artifacts of interest that may inspire future ideas, to 'doodle' half-formed thoughts, and to share ideas with others by showing [2]. The sketchbook is particularly valuable as it encourages its owners to develop a multitude of ideas and choose between them, rather than to fixate on a single idea. Buxton [2] calls the process of distilling between many ideas as 'getting the right design', whereas the process of developing a particular idea (e.g., through iterative refinement or usability engineering) is 'getting the design right'. The former emphasizes design that chooses between idea alternatives, while the later is the creative engineering that refines a particular idea.

Computer science students do not normally keep sketchbooks, and as a consequence they typically develop the first idea that comes to them. That is, they worry about 'getting the design right' without considering if the basic idea is the best one worthy of pursuit. This is equivalent to the local hill climbing problem in Artificial Intelligence, where local maxima are reached without considering how they would relate to a global maximum. Sketches become a way to investigate other nearby hills (ideas) to see if they can offer better solutions.

To encourage students to develop many ideas, I made the course text an empty sketchbook. I insist they buy a nice one (hard cover, coiled) so they can take pride in it; otherwise (in my experience), they will end up using scraps of paper. Students are expected to fill their sketchbook with their project ideas over the course of the term, and to show these ideas to others on demand. I can ask to see it at any time, where their number of sketches must reflect where they are in particular projects. This stops students from 'cheating' the process by sketch cramming, where they sit down at some late date (e.g., after the project is being done) and just 'brain dump' a bunch of sketches as a single batch.

With the assistance of Coleen Campbell, a teacher and doer of arts and design, and influenced by Bill Buxton's book 'Sketching the User Experiences' [2], I developed a brief instruction manual for the sketchbook; an extract is included in Table 1. In terms of grading, the key deliverables are that students must generate at least ten different sketches demonstrating quite different ideas for a particular project, and then choose one idea and develop ten variations and/or refinements of that idea. Unlike most grading schemes, they are evaluated on quantity, not quality!

Table 1. The Sketchbook 'Instruction Manual'

Why a sketchbook	Real progress in developing yourself as an interaction designer will depend on you frequently and habitually sketching out your ideas and their variations, recording other people's ideas you may see, reflecting and choosing between these ideas, and then further developing those ideas that seem promising. The sketchbook records all these. Carrying the sketchbook with you at all times will help you incorporate sketching and reflection into your daily routines.

What is a sketch?	The following list paraphrases Bill Buxton's properties of sketches [2]. • **Quick** to make. • **Timely** so they can be provided when needed • **Inexpensive**, where cost must not inhibit the ability to explore a concept. • **Disposable** so you can afford to throw it away - the investment is in the concept, not the execution. • **Plentiful,** where its meaning is within the context of a collection or series • **Clear vocabulary** where the rendering style signals that it is a sketch • **Distinct gestures**, where their fluidity gives them a sense of openness and freedom *vs.* engineering precision and tightness. • **Minimal details**, including only what is required to render the concept.
Uses	Sketchbooks are useful in many ways. It is a place where you should: • **Jot down and annotate** your own initial ideas - and there is no such thing as a bad idea! • **Explore and refine** ideas both in the large and in the small • **Develop variations**, alternatives and details • **Refer back to your ideas** and reflect on how your thought processes have changed over time • **Record other good ideas** you see elsewhere e.g., in other systems, in your readings, and in your classmates' work. • **Collect existing material** (e.g., pictures from magazines, screen snapshots) and tape them into the sketchbook. • **Develop your skills**, your accuracy and your confidence in sketching out your ideas through regular use Sketches do not have to be pretty, beautiful, or even immediately understandable by others. However, you should be able to explain your sketches and ideas when anyone asks about them.
Best Practices	• **Always carry your sketchbook** with you everywhere (a 2nd small sketchbook is helpful). Jot down ideas as you think about them. • **Always have a pencil** handy in the coil binder. • **Use it frequently**, e.g., at least several times a day. • **Fill pages** with a series of related drawings about a design idea, or with a single well-composed design idea. • **Consider alternatives.** A series of sketches related to the same interface problem might explore different aspects of the interface. These could include different interface representations, different interaction details, different screens, different levels of details, different contexts of use, and so on. Each page can become a series of studies that will help you develop and reflect on the many ideas you will have. • **Annotate** drawings appropriately, including information such as descriptions for ideas that you cannot draw out well; textual addendums; sources of your ideas (e.g., books, magazines, classmates), creation date, and any other relevant information. • **Do not erase** ideas because they are messy or because you no longer like them. Your sketchbook is a record of all your developing ideas, good and bad, not just of your final work. • **The sketchbook is for design only** - do not use it for other classes just because you do not have any paper.

Sketchbook grading	I and the teaching assistant will be looking for the following evidence of use. • **Idea quantity**, where you develop many ideas: for each project, we expect a minimum of 10 sketches illustrating 10 quite different ideas and a minimum of 10 refinements / variations for a chosen idea; • **Regular use**, where you habitually use the sketchbook to jot down, annotate, and develop ideas over time – at any instance, we expect your sketches to reflect where you are in your project; • **Thoughtfulness**, where you can explain the development of your ideas within particular sketches; • **Attribution**, where you credit other people's ideas that you are using.

5 The Portfolio

The portfolio is another 'best practice' found within design disciplines. A portfolio serves as a living resume, where designers use them to collect and illustrate their (usually completed) projects. Designers show portfolios to others both to highlight individual achievements, and – as a collection – to suggest the scope, breadth, depth and quality of the professional's design proficiency. Thus a good portfolio will summarize the professional's abilities, strengths and styles. Portfolios can vary greatly: they can be stand-alone artifacts for others to review, or can serve as a conversational prop where designers tell stories about the artifact to peers, employers, and clients.

Sadly, computer science students rarely create portfolios. Instead, they typically describe their ability to others through a skills-oriented resume format: the programming languages they know, the courses they have taken, and related employment history. This is somewhat surprising, for most computer science students – like designers – spend immense effort developing projects as part of their course work. In spite of these creative experiences, computer science students portray themselves to their potential employers as skill-oriented technicians rather than as people well-practiced in the art of invention.

To encourage students to capture their work in a way that can be presented to others, students are required to construct a portfolio as the course progresses. Two types of portfolios are required, as described below.

The *project learning portfolio* captures the essence of each student's project through a variety of media. As each project is completed, each student creates a detailed portfolio documenting and archiving what they have done. It takes two forms, both done for every project. First, the student creates a web site with a project portfolio entry: this web site includes a brief executive summary, then a visual summary of what they did (captured as annotated screen snapshots, simple animations and/or storyboards, slide show), downloadable executables so that others can try their system, and an archive of their source code. Second, the student uses more traditional media to create an alternate visual summary. The chosen media varies per project, and covers a range including paper-based posters, booklets, commercial packaging (e.g., a box), and a short self-contained video. The goals of these two portfolio styles is to encourage students to create a stand-alone electronic presence of their work (through the web site), and to give them practice creating

portfolio entries with alternate media that can be quickly shown to others. As a side effect, the project portfolios are an easy way for the course instructor to access and review each project after the demonstrations, where details can be explored.

The *professional portfolio* is created at the end of the term. Its intent is to be used as a living resume, where students can bring it to their job interviews and show potential employees and/or clients about the kinds of work they did as well as the scope of their many achievements. The emphasis is that these portfolios must serve as a conversational prop: students should be able to conveniently carry it to a job interview, where its contents are easy to show, browse and discuss as opportunities appear. Thus paper-based portfolios are encouraged, although I also suggest they include electronic medium (e.g., a CD) that they can quickly load into a computer if time and facilities allow for further elaboration and demonstration. These portfolios can come in quite different form factors, although the two main styles are booklets or modest-sized posters held in a portfolio case. I should note that the professional portfolio is rarely a rebundling of the project portfolio entries: students create or remix entries to exploit the form factor and visual effect of their professional portfolio.

In practice, students take great pride in their project and professional portfolios. Many are quite creative in building the portfolio web site, to the point that the web site itself demonstrates the student's skills. Similarly, many pay considerable attention to their professional portfolios. They construct it out of high quality materials, and extend it to include projects outside the course. A variety of students described how they used the portfolios during actual job interviews, and how they believed it made the difference in setting them apart from other applicants.

As with the sketchbook, I developed a brief instruction manual for the portfolio; an extract is included in Table 2 below.

Table 2**.** The Portfolio 'Instruction Manual'

What is a Portfolio?	A portfolio is a representative or selective collection of one's work. • Design professionals (e.g., architects, industrial designers, artists) often create professional portfolios, and use these to illustrate their work to potential employers or clients. A portfolio is a living resume. They are an expected part of how professionals in many disciplines portray their achievements. A good *professional portfolio* will contain visual samplings that collectively suggest the scope, breadth, depth and quality of the professional's design proficiency. It summarizes the professional's abilities, strengths and styles. • Some educational programs also have students create *learning portfolios*, where students document their work, sometimes over years. These portfolios are used by instructors to evaluate students, and by students to help them reflect on what they have learnt over that time. • Unlike sketchbooks, portfolios are neat, orderly and professional in appearance. You critically select and craft what goes into it. Because this is a design-oriented portfolio, its contents should be highly visual. Each visual summary should tell its own story with only modest labeling and textual descriptions. It should also serve as a conversation piece letting you talk about your work.

Your Portfolio	You will create your own learning and professional portfolio • Your learning portfolio will show your projects. You will document your developing abilities as an interaction designer by creating visual summaries of how you solved your exercises and assignments. As the course progresses, you will see what you have accomplished to date. • Near the end of the course, you will use this learning portfolio to seed your professional portfolio. Feel free to add samplings of any other relevant work you have done outside of this course to your professional portfolio. • After the course, you can maintain and modify this portfolio into something that will help you present yourself to future employers.
Learning Objectives	Your portfolio will help you learn the following. • Develop skills creating visual summaries of individual designs by using screen snapshots, story boards, videos, and other techniques. • Demonstrate in these summaries how you have used particular interaction techniques. • Learn how to effectively archive your code and supporting documents so you can easily install and demonstrate your system on any handy machine. • Develop your skills in creating both a professional-looking learning and professional portfolio • Use the portfolio as a personal reference summarizing your course accomplishments.
Be Organized	A professional portfolio can be packaged in many ways. • Keeping all summaries organized but separate will allow you to selectively rearrange your portfolio to fit your need (e.g., for a job interview). • The simplest form sees it as separate summaries collected in some kind of container e.g., an artist's portfolio case. • You can also paste summaries into a large high-quality (but very good looking, maybe even hand-made) booklet. Ideally, pages are removable so you can add and rearrange items as needed.
Styles	For each project, you will create two versions of a learning portfolio entry. The first is an electronic web summary (including code archive), while the second is crafted out of a media form selected from the list below. • **Paper**. Each visual summary (screen snapshots, storyboards, etc) are pasted onto a high-quality mat or backing (e.g. poster cardboard). Poster sizes of at least 16"x20" will give you enough space to create an effective visual summary. Alternatively, you may want to create it as a flip-book of screen-shots showing how the interaction flows over time. • **Packaging**. The visual summary is created as packaging, e.g., a paper box that would contain your software and other materials. • **Video**. Create a video is a very effective way of showing your work. The best videos are short ones focused on showing your design in action. • **Interactive multimedia.** You can create a project summary using a multimedia presentation tool e.g., Powerpoint or Flash. • **Running software.** You will archive the system in your electronic portfolio so that you can demonstrate it as needed. Be aware that this does not suffice by itself: over time, it will become unlikely that your system will run due to changes in operating systems and expectations of installed software. Thus you should see this as a way to supplement your other portfolio activities.

Best Practices	*Hint. Stress visuals over text. A common error is to include overly long text descriptions.* • Be creative in your portfolio – it also illustrates your design abilities. Search the web (e.g., try the terms Interface Web Design Portfolio) for examples of how other people have created on-line portfolios. • Carefully decide what parts of your projects you want to use in your portfolio summary. • Treat portfolio creation as a design exercise. Prototype a few different approaches for each project. Your first idea may not be your best one. • Label each summary with a descriptive title and its date of completion. You may include short explanatory annotations and paragraphs, but don't go overboard: this should be a visual summary rather than a textual one. • If you are using screen snapshots, make sure they are interesting ones i.e., the screen is populated with meaningful data, and the screens are in an interesting visual state. • Use storyboards to show how an interesting interaction sequence progresses over time. • Print screens in color. • Include screen fragments to embellish your story. For example, a blow-up of a screen portion can show hard-to-see details (and can include annotations); a mini-storyboard of an interaction component can illustrate how a particular interaction technique works. • Emphasize any uses of novel interaction techniques. • Make sure you can 'disassemble' your portfolio summaries or that you have archived electronic copies of images so that you can regenerate them on paper. In the future, you may want to recreate a portion of your portfolio, perhaps in a more expensive book or some other medium, only to find that you cannot unglue your images from the pages.
Portfolio Grading	Every project requires a learning portfolio entry. You will also create a professional portfolio at the end of the course. Portfolios and contents should impress me with both your vision and how well you have mastered the technical aspects of interaction design. Other grading aspects include: • completeness • quality (how well the portfolio captures your work and the techniques we have asked you to include in it); • professional appearance (including overall organization). • effectiveness of your code archive.

6 Experiences and Reflection

The first year I taught this class, I was astounded by the quality and the diversity of the students' projects: they all consistently surpassed my expectations. Successive years proved that this was no accident. Because of this quality, I have featured student's work (with permission, of course) in a variety of publications and web sites, as described below.

Greenberg and Tse [9] is a video that catalogues eight student projects in the area of Single Display Groupware. Examples of what they did include: collaborative activities such as photo selection, viewing and organization; children games for

cooperative fashion, multi-person music playing, action games where people work together toward a goal vs. competing with one another, multi-person drawing and sketching tools, and so on. Source code and executables of several of these projects are available at http://grouplab.cpsc.ucalgary.ca/cookbook/index.php/Toolkits/SDGToolkit.

In other papers [6,5], I feature projects students did on physical user interfaces, concentrating on those that emphasized collaboration. The range of systems developed include a variety of status indicators of people's activities, devices that serve as multimedia communication channels, notification displays of asynchronous message arrival and of meetings, tools supporting information exchange, sensor-based systems safeguarding privacy, collaborative games, and interactive art. A comprehensive web site features a video gallery of almost all Phidget projects developed in this class over the years: http://grouplab.cpsc.ucalgary.ca/phidgets/gallery/.

McEwan et. al. [14] briefly describe various student groupware projects created using the Community Bar media item toolkit. Example projects include a group-editable list of web pages, an awareness tool that displays motion activity at distant sites, a novel visualization of a video media space history over time, a group editable and browsable photo gallery, a group scheduler, a gossip item for teenage girls, a family shopping list, a document status awareness item that tracks documents being worked on by the group, and a group interface to a mobile robot that allows people to view a physical environment and contact people within it.

The main question is, of course, did students gain the rudimentary best practices typically found in a design discipline? The quality of the results above suggest that they did, at least within the constraints of the course. Perhaps a more relevant question is: did students carry theses best practices with them to their jobs, and did they apply them to design problems? This is a more difficult question to answer, as I have not systematically tracked students after they graduated. However, I do know that several students have gone on to careers as interface developers / experience designers. Some still carry a sketchbook with them years after the course. Most said that they believe bringing their portfolio to the job interview helped clinch that position.

The question of whether students continued these best practices is perhaps the difference between a design studio run within computer science constraints *vs.* the way it happens in a professional design program. Professional design programs have multiple design studios that run almost every semester. Students are so habituated in the design process that best practices become engrained. In contrast, there is only so much that one can do in a single computer science course containing an abbreviated design studio. Perhaps the most we can expect is that it gives students a flavor of what design is all about.

The ideal solution, of course, is to transform a good portion of Computer Science into a design discipline using the tricks of the trade found in design programs. This is not that far-fetched. While some courses are predominantly about knowledge transfer, a vast number of courses in Computer Science are project based amenable to structured design. Perhaps the greater problem is that Computer Science faculty members need to retrain themselves as designers before this can happen.

Acknowledgments

I am not a formally trained designer. My early attempts to create a design-studio course relied heavily on advice by others. In particular, I thank my colleague Sheelagh Carpendale at the University of Calgary (a computer scientist and a trained artist who also taught one of the modules in this course), and Colleen Campbell, then a design instructor at Mount Royal College, Calgary. If I've said anything that would make a true designer uncomfortable, I assume all blame in misinterpreting what my colleagues said.

References

1. Blevis, E., Lim, Y., Stolterman, E., Wolf, T. and Sato, K. Supporting design studio culture in HCI. Workshop Overview, *Proc ACM CHI 2007 Conference on Human Factors in Computing Systems* v.2, ACM Press, 2821-2824. (2007)
2. Buxton, B. *Sketching User Experiences: Getting the Design Right and the Right Design.* Morgan-Kaufmann. 2007.
3. Carpendale, M.S.T. and Montagnese, C. A Framework for Unifying Presentation Space. *Proc ACM UIST Symposium on User Interface Software and Technology*. ACM Press, 61-70. (2001)
4. Docherty, M., Sutton, P., Brereton, M., and Kaplan, S. An innovative design and studio-based CS degree. *SIGCSE Bulletin.* 33, 1 (March), 233-237. (2001)
5. Greenberg, S. Toolkits and Interface Creativity. *Journal Multimedia Tools and Applications (JMTA)*, 32(2), February. Springer. 139-159. (2007)
6. Greenberg, S. Collaborative Physical User Interfaces. In K. Okada, T. Hoshi and T. Inoue (Editors) *Communication and Collaboration Support Systems*. IOS Press (Amsterdam, The Netherlands), 24-42. (2005)
7. Greenberg, S. Teaching Human Computer Interaction to Programmers. 3(4), *ACM Interactions,* July-August, ACM Press. 62-76. (1996).
8. Greenberg, S. and Fitchett, C. Phidgets: Easy Development of Physical Interfaces through Physical Widgets. *Proc ACM UIST Symposium on User Interface Software and Technology*, ACM Press, 209-218. Includes video figure. (2001)
9. Greenberg, S. and Tse. E. SDGToolkit in Action. *Video Proc ACM CSCW'06 Conference on Computer Supported Cooperative Work*, November, ACM Press. Video and two-page summary. Duration 7:14 (2006)
10. Klemmer, S., Hartmann, B. and Takayama, L. How Bodies Matter: Five Themes for Interaction Design. Proc *ACM DIS'06 Designing Interactive Systems,* ACM Press. (2006)
11. Lawson, B. *What Designers Know.* Architectural Press, Elsevier. (2004)
12. Marquardt, N. and Greenberg, S. Distributed Physical Interfaces with Shared Phidgets. *Proc. TEI'07 1st International Conference on Tangible and Embedded Interaction.* (Feb 15-17, Baton Rouge, Louisiana, USA). (2007)
13. McEwan, G., and Greenberg, S. Supporting Social Worlds with the Community Bar. *Proc ACM Group 2005 Conference*, ACM Press. (2005)
14. McEwan, G., Greenberg, S., Rounding, M. and Boyle, M. Groupware Plug-ins: A Case Study of Extending Collaboration Functionality through Media Items. Proc CollabTech 2006 2nd International Conference on Collaboration Technologies, Tsukuba, Japan, July 13-14, IPSJ SIG Groupware and Network Services, 42-47. (2006)

15. Reimer, Y. and Douglas, S. Teaching HCI Design with the Studio Approach. *Computer Science Education* 12(3), pp. 191-205. (2003)
16. Tse, E. and Greenberg, S. Rapidly Prototyping Single Display Groupware through the SDGToolkit. *Proc Fifth Australasian User Interface Conference*, Volume 28 in the CRPIT Conferences in Research and Practice in Information Technology Series, (Dunedin, NZ January), Australian Computer Society Inc., p101-110. (2004)
17. Tse, E. and Greenberg. S. SDG Toolkit. *Video Proc ACM CSCW 2004 Conference on Computer Supported Cooperative Work*. (November 6-10, Chicago, Illinois). ACM Press. Video and abstract, duration 3:55. (2004)
18. Winograd, T. What Can We Teach About Human-Computer Interaction. Proc ACM CHI 1990 Conference on Human Factors in Computing Systems, ACM Press, 443–449. (1990)