

# Master of Fine Arts in Software

Richard P. Gabriel

Software education today is embodied in Computer Science and Software Engineering programs, supplemented by informal mentoring on the job. I find this approach unsatisfactory. Software development is a performance exhibiting skills developed by an individual—often in groups of teams in order to achieve the scale of software required. In this way, software development is like putting on a play, which requires the skills and performances of a number of people working in tandem on stage and behind the scenes. Such skills can be developed in isolation through practice with other amateurs or even by putting on plays in public without any training at all. But how much faster could talent be developed in an educational program that recognized that writing software has enough of an arts-like performance component that the program was tailored to it?

Another apt comparison can be found in the creative writing arts. It is entirely possible to become an extraordinary writer by one's self, by simply writing and reading, and many excellent writers progress this way. A faster way to gain competence is through a Master of Fine Arts program, which is designed to rapidly increase one's skills and to get one prepared to bring to bear critical thinking to the process of continuing improvement. Some believe that all aspects of software design and development are really engineering or scientific disciplines where the models of engineering and science apply, and I will not quarrel with them nor try to convince them otherwise.

My observation is that software design and development encompass more than pure engineering and science: The code and design of a system are communications with other people as well as with machines, and any part of a design that touches people—such as user interface design and what is generally meant by the term “user-centered design”—involves understanding perception, storytelling, and the needs, comforts, and predilections of people, which is as much the stuff of art as it is of engineering. Therefore, to my way of thinking, the same sorts of educational practices as are used in the writing arts should be used to augment what we already have that are engineering and science based.

Many degree programs for Computer Science and Software Engineering work by shovelling information into the student along with presenting opportunities for developing critical thinking. Despite the tone of this description, I believe it's a valid way to teach certain students certain things, to prepare them in a particular way for their work or research careers. In the writing arts there are similar approaches in the forms of Master of Arts and PhD degree programs. But both in software as in writing, there are people whose work is “doing the thing”—writing and designing programs—and such people do this work every day. They hope to be good at it and to be able to improve over time. They have pride in what they do and are satisfied or not with each project they do. To them what they do feels more than craft, includes engineering and science, but still feels like more. This proposal is aimed at educating those people.



This is a proposal for an Master of Fine Arts in Software program. For purposes of simplicity I am locating within the semantic range of the term “software” not only software code and coding, but software design, user interface design, user-centered design, humane design, and in general the practice which is creating the entirety of a software system including its human interactions, what to design in the first place, and the techniques for constructing such a thing. Some will feel most comfortable limiting their imagination of the program to design, and it is true that no one will be able to graduate from the program without displaying excellent high-level design skills. But in my view, a person may reasonably enroll in the program to become quite simply an extraordinary cod-

er—but one with very good design skills. Others may wish to focus on the graphic parts of user interface design, for example, and these folks will need to end up with a good grounding in implementation techniques and skills.

This proposal is predicated on the belief that being a good software designer and developer requires talent, and that talent can be developed. We explicitly liken the practice of software to the practice of fine art.

The structure and constitution of this program is such that we can begin it immediately upon an educational institution being given the right to grant the degree called “MFA in Software.” There is no curriculum to develop, no books, no semester-long classes. You can see this by realizing that the MFA is primarily a performance degree not a knowledge degree—though every student and teacher will gain knowledge along the way, and knowledge will be generated by the program. Education in this MFA programs operates by providing a context in which students are constantly designing, writing, and working with software under supervision, with critique, and with explicit thought being given by the student to what the student is doing.

One way to look at it is that the student is writing, designing, and working with software while paying attention.

“Writing, designing, and working with software” is achieved by the student working on a variety of his or her own projects under the serial supervision of a variety of faculty. This is a low-residency program, which means that these projects are done at home or at work. In this way, the program resembles a mentorship program, and the work done by the student can be projects of interest to his or her employer.

“While paying attention” is achieved principally through writing short essays called *annotations*. An annotation is a vehicle for the student to pay attention to one *craft element* at a time. A craft element is something that makes software effective and beautiful. Craft elements can range from the smallest coding details such as the choice of variable, function, class, or method names through the design of interfaces and protocols through the layout of information on a screen to the largest concerns such as the architecture of a system and descriptions of what to design. In essence, anything created that is a hallmark of what sets the extraordinary apart from the ordinary in software and software design is a craft element.

Here are the parts of the MFA for Software program:

- The MFA is a two-year, low-residency program: There are 10 days of residency twice a year (semester system), July 1–11 and January 1–11 (roughly) at the start of each semester. Each semester is about 5 months long, and during the majority of the semester the student is working at home or at work. The workload is gauged so that a student can also hold a full-time job while in the program. The choice of when residencies take place is opportunistic: We hold our residencies when the school’s regular students are on vacation between semesters.
- Admission is based on letters of recommendation, successful graduation from a CS-related undergrad or grad school program or equivalent experience, submitted work (for example, a user interface), a critical essay on an existing piece of software, and a statement of purpose covering what the student hopes to gain from the program and the student’s ability to get something from such a program. Students are admitted every semester. Student tuition is about \$15,000–\$20,000 per semester including room and board during the residencies. This should cover expenses at the college, some additional funds to the college, and pay for the faculty, who nevertheless teach mostly for the good of it not the pay of it.
- Students specialize in genres, for example, software implementation, user interface design and implementation, user-centered design, and system design. There certainly will be others.

- There is no permanently hired faculty; each semester the director of the program locates enough faculty to accommodate a student/faculty ratio of around 5/1. I expect there will be a core faculty who are relatively frequently on the faculty semester to semester. Faculty are selected from industry and academia (to some extent). Faculty are chosen from among extraordinary software practitioners and teachers.
- It's a 4–5 semester program. The first 2 semesters are general reading and project work (described below), the third semester is devoted to a longer critical essay on a substantial system, and the final semester is devoted to producing a larger project which can be the accumulation of projects done to that point in the program (as long as it forms a coherent system).
- During each residency the following happens:
  - There are lecture and discussion classes taught by faculty and graduating students on topics appropriate to the fine art of software. These are the only classes during the semester. In general these classes are about a particular craft element, software system, design issue, or user interface. A typical class is about an hour with some being 2 hours long.
  - There are presentations of work by faculty and graduating students (every evening). This is the equivalent of a reading.
  - Students are assigned to advisors by committees of all the faculty in each genre.
  - Every day there are workshops of 10 or so students and 2 faculty members (rotating)—student work is workshopped as in a writers' workshop.
  - Students and advisors plan the semester project and select a reading list. A semester project could be a series of programs, a user interface design, a larger system project, or really anything that makes sense to the student and adviser. A student may use as a project something he or she is doing at work. There is also a reading and critical component in each semester. Typically a student will have a coordinated semester plan with the software work, reading, and annotation writing all on the same sorts of topics. For example, a student might decide to try his or her hand at functional programming and plan a series of programs to explore that paradigm. Readings would be of the FP literature and “published” functional programs (stuff on the Web, in the faculty member's collection, etc). Annotations could be on time/space tradeoffs, laziness, monads, and the like.
  - Thesis interviews (described below) take place.
  - Students graduate.
- During the at-home portion of each semester, every 3 weeks or so each student sends to his or her advisor the following:
  - new work, typically in the form of code, code designs, UI fragments, UI designs, system designs, etc.
  - revision of work reviewed already by the advisor. This and the annotations are the most important aspects of the program. The student will produce a design or some code in small enough chunks to be manageable in 3 weeks. The faculty adviser will read, test, and respond in writing to the work, particularly with respect to the goals of the semester. Based on this critique, the student will revise the work, perhaps many times. This is unlike simply throwing off a piece of code if it works as often happens in the real world. As the student gains more experience with how to see the work and how it is put together explicitly, the better will become the student's first-draft work. It is a process of doing and reflection.
  - 3 or so annotations—short critical essays on a software craft element; these annotations address published or existing work, not student work.
- Each student is expected to spend about 20 hours each week to support this workload.

- Creating software is not just a solo activity—it involves collaboration. A student must select one of the first two semesters or an extra semester as a collaboration semester in which the student works of a common project with his of her adviser and the adviser’s other students. For the first half of the semester, the entire group resides at a common location, which is agreed to at the residency—this could be on campus; for the second half of the semester, the group works together remotely in an open-source-like way or any way the group decides on at the residency. The collaboration is considered a work, and so is the subject of at least half the annotations that semester.
- Annotations are based on student reading. Students are expected to read a book, read the code for a program or part of a system, or examine some other part of an existing program or system every week. Annotations are on craft elements as described in books or exhibited by code and interfaces. Code read by the student and not available to the faculty advisor must be made available to the faculty member.
- Advisors respond in writing to students’ work within about a week or two of receiving it.
- During the essay semester, the annotations can be used to lead up to the essay, which is a longer critical piece (50–100 pages) on an entire system or significant piece of software. The essay should demonstrate depth of analysis and engagement with a larger topic than the annotation.
- During the final semester, the student completes the final project which is an entire system, a significant piece of software, or a significant design with enough code written to show it is feasible to implement and is usable. This final project is often the culmination of the work done during the entire program.
- To graduate a student needs to do the following:
  - receive full credit for at least 4 semesters
  - participate fully in at least 5 residencies, which includes writing commentaries on each class attended, attending a workshop, and producing a semester project plan
  - successfully complete about 12 annotations per semester (excluding perhaps the essay semester) and a total of at least 36 annotations for the entire program
  - complete a substantial body of software work each semester
  - teach a 1 or 2 hour class on any software-craft-related topic
  - complete a significant software project
  - present the software project to the student body and faculty as a whole
  - discuss the software project at the thesis interview, which is made up of 1 faculty member and 2 students. A month before the interview, the faculty member and 2 students receive the software project for review; at the interview, the system is discussed as among peers with the goal of seeing what additional work or direction is needed for the project or how it will mature.
  - participate in 2 thesis interviews as a student
- Before the final semester, the student petitions for graduation by providing a synopsis and fragments of the final system, which is judged both by a committee of faculty and an outside software expert (anonymously) who is familiar with the requirements and quality standards of the program.
- There are no grades. Instead there is a narrative report done incrementally each semester on the student’s progress, strengths, and weaknesses.

Those are the important points. The way that the program works is for each student to spend a lot of time explicitly thinking about the craft elements in any piece of software, design, or user interface. This is accomplished by the annotation. Attention is paid to the smallest details first and the focus expands as time goes on (at a rate that depends on the student). The faculty advisor is chosen from software practitioners and teachers, but always the faculty member is an extraordinary practicing software person. The software each student produces during the semester is carefully critiqued in

writing by the faculty member. Work on a piece of software or a design is continued until it is completed; this might mean that a single 1-page program or the design of a dialog box may be revised 5 or 6 times or even more over the course of the program.

It is this explicit attention to matters of craft that matures each student into an excellent software practitioner.



This program can be quickly set up assuming there is a friendly host school and a group of 5 or 6 dedicated core faculty. A Board of Directors also needs to be formed to resolve various issues such as recruiting faculty, settling disputes, and establishing rules for the program.

There is no preset curriculum, though I anticipate that themes and trends will emerge which form a sort of “school” of design thought which will be passed on to all students. This is a performance degree, which means that students who graduate will be extraordinary practitioners. Though the MFA in Software is a terminal degree, it is not suitable as a teaching credential.

The host school would need to be able to house students during the residencies as well as support one permanent faculty member on its faculty who would serve as the director of the program. Further, a full-time administrative assistant and perhaps a lower level assistant would be needed particularly during admissions, which occurs twice a year.



There is no other program like this in the world today. This is because the traditions of Computer Science and Software Engineering have tried to turn all aspects of software creation into a pure engineering discipline when they clearly are not. The MFA in Software would begin to repair this error.