How I Teach Fluency

Lawrence Snyder

Note to Reader: This description has been written for both UW and non-UW audiences, with the result that it contains detail not needed by one or the other audience. Please excuse the inclusion of this extra information and ignore it. Thank you.

Contents

Fluency with Information Technology is a course designed to teach the "Fluency content" as set forth in the National Research Council's report, *Being Fluent with Information Technology*[*]. The course that I will describe is jointly offered by the Computer Science and Engineering Department (CSE100) and the Information School (Info100) at the University of Washington, and is known as FIT100. The course has undergone considerable evolution since I first offered it Spring Quarter, 1999. (See Acknowledgments.)

FIT100, Fluency with Information Technology, is a challenging class to teach not only because the content is extremely broad—far broader than the experience of even "Renaissance" faculty—but also because the population is composed largely of non-technical students, a new constituency, for me at least.

There is no claim or warranty that this is the best or even the preferred way to teach Fluency. It is simply the way I have taught the material. It is offered to the extent that it is useful. I have taken the trouble to write down my thinking on this class in the spirit of a posting in Rick Reis's Tomorrow's Professor:

> "When all the careful, difficult, intentional, and scholarly work of planning and teaching a course is undocumented, it is lost for further use. Not only is it unavailable for the teacher's own reflection, but it is not there for aspiring teachers and colleagues to learn from. It is also unavailable to those making important decisions about hiring, promotion, and tenure, and to those mentoring colleagues who are being considered in those processes." – Dan Bernstein and Ellen Wert in MAKING VISIBLE THE INTELLECTUAL WORK IN TEACHING, Stanford Center for Teaching and Learning, http://ctl.stanford.edu

Since there are many ways to deliver Fluency, I assume that readers will be also be documenting their efforts at creating courses. I encourage both the documentation and its circulation to a wider community. It would be ideal if this document could serve to focus a discussion of "best practices" for teaching Fluency. The more we know about successful ways to teach Fluency, the more effective it will be.

Structure of this document

Though this document has been structured to be read front-to-back, I believe it is also useful as a reference for some of the individual topics. Accordingly, I have made every effort to make the sections standalone.

[*] *Being Fluent with Information Technology*, National Academy Press, 1999

Brief Review of Recent History

In 1997 the National Science Foundation asked the National Research Council to conduct a study on computer literacy. Officials noted that many Americans were becoming literate computer users—they could send email, browse the Web, use basic word processing tools, etc.—but would that be it? That is, would the impact of computers, the Internet and the information resources of the World Wide Web be limited to simple "clicking" for the average person in society? Clearly, for the digerati the information age was transformational, but would it be for everyone? Stated briefly, the NSF wanted to know, "What should *everyone* know about Information Technology?"

That simple question, the driving force behind the study, immediately begs the question, "For what purpose?" What benefits could accrue to a population more knowledgeable about IT? NSF listed among its anticipated benefits:

- *Job Preparation* IT is a critical productivity tool for businesses, and the better workers are prepared the more effectively can they apply all IT. Naturally, businesses will train their employees to use the firm's IT tools, but a better-prepared workforce makes America more competitive.

- *Citizenship* The so called information revolution brings problems to society that it has never before treated, such as judging the limits of strong encryption, evaluating electronic voting, assessing the wisdom of owning intellectual property such as music, etc. An informed citizenship capable of weighing benefits and risks promotes better governance.

- *Personally Relevant Goals* Many people realize that they under-use IT. They want to benefit much more completely from it. More knowledge promotes the citizens' personal goals.

Numerous other opportunities accrue from an IT-knowledgeable population.

The committee—Alfred Aho, Marcia Linn, Arnold Packer, Lawrence Snyder (Chair), Alan Tucker, Jeffrey Ullman and Andries van Dam—concluded after two years of study that a new level of expertise about IT was needed.[*] The new goal, dubbed Fluency, was composed of three kinds of knowledge:

> Contemporary Skills—using IT resources today,
> Fundamental Concepts—understanding the basic ideas underpinning IT, and
> Intellectual Capabilities—applying higher level thinking to IT problems.

[*] *Being Fluent*, Ibid.

The committee listed its top 10 topics in each of the three areas. Further, they recommended students integrate their new Skills, Concepts and Capabilities know-how using projects, multi-week exercises that result in an IT product such as a Web page, database, etc. The report—presented at forums like the Snowbird Computer Science Department Chairs meeting—was well received.

I decided to try to teach a Fluency course.

The principal motivation for a course was that the report did not provide a college level curriculum. Indeed, there wasn't much attention paid to implementation at all. Further, the top 10 topic areas were very generally specified, and quite uneven. That is, topics like "sustained reasoning" [Capability Topic #1] are extremely broad and might be treated repeatedly throughout a course, while topics like "universality" [Concept Topic #8] can probably be taught in fifteen minutes. Having the advantage of sitting through all of the committee's deliberations, I believed I had a reasonable understanding of the depth and emphasis that the committee intended, which may not have been evident in the report. Accordingly, it seemed that an "example" course would both benefit students at UW and elaborate on the report.

With the endorsement of CSE and the iSchool, I developed a course to be offered in Spring 1999. Forty students signed up, half freshmen and half seniors. There were numerous handicaps: no text, little infrastructure, a very broad set of topics, etc. Further, I had never tried to teach computer topics to an audience that didn't have a consuming interest in computation. The course was evaluated, [*] and though it may have been successful from the students' point of view, I thought it fell short of what could be done. So, I signed up to teach it again in Autumn Quarter 1999.

CSE100/Info100
Fluency with Information Technology
5 Credits
Fulfills QSR requirement
3 Lectures (50 minutes) MWF
2 Closed Labs (50 min., w/TA) TTh
Classroom limit: 150
Offered each quarter

The first task was to produce notes to ease students through the programming component of the course. (These notes, enhanced through several iterations became Addison Wesley's *Fluency with Information Technology* [2004].) Even the few programming concepts recommended by the NRC report strike fear into many students and maybe even some potential instructors. I had learned a lot from the students in the first offering of the course concerning (a) what they found difficult and (b) the point of view they brought to the topic. For example, variables are difficult—they are confused with unknowns in algebra—while event-based program execution is usually very intuitive. The goal was to produce resources that would simplify teaching. Autumn 1999 went much better, especially the programming, than had the first offering. I offer some of my tips in a later section.

[*] http://www.washington.edu/oea/9915.htm

Another challenge in teaching the class is to produce interesting projects that are achievable by the students in a limited amount of time. This is not as easy as it might appear. Some projects have worked out better than others. (I offer guidelines below.)

As other faculty members taught the class, it relaxed and improved. I attribute much of this progress to my iSchool colleague, Grace Whiteaker. For example, my original offering had assigned four 2-week projects. This may have been ideal, and might work well in a semester, but for a quarter it was simply too much. We have moved to a busy-but-achievable pattern of two 2-week projects and one 3-week project.

Eventually, it was necessary to return to NSF's original goal of defining "What everyone should know about Information Technology." The report had listed the topics and FIT100 had made that list into a concrete curriculum. *Being Fluent*'s recommendations, that people become effective computer users and prepare for lifelong learning, was being realized at UW and other campuses across the world. Students were taking the class, and were becoming sophisticated, confident users, at least anecdotally. But Fluency was reaching only a few thousand students per year, and even when it is taught at every college (or better every high school) on the planet, it would still reach only a few million per year. The *everyone* goal for NSF was not even close to being achieved.

Though there are a variety of ways to reach the larger population—from TV programs to hints on the backs of breakfast cereal boxes—there was a need for some mechanism that facilitated people using the resources already developed. So, I applied for and was awarded an NSF grant[*] to put the Fluency course online. The course BeneFIT100 would be hosted by UW and be free to the public.[*] The project was implemented largely by UW's Distance Learning group with Grace Whiteaker as the primary author. The course was launched in March 2004.

The importance of the BeneFIT100 experience from the point of view of teaching Fluency is that BeneFIT100 was piloted by two groups and assessed for its effectiveness. The results have informed this the following material.

[*] BeneFIT100 …
[*] `http://www.fit.washington.edu`

Meta Concepts

This section—perhaps the most important in the document—deals with the topics that transcend the particular way I have taught FIT100. That is, no matter how Fluency is offered topics such as teaching programming or writing project descriptions must be addressed because they are the essence of Fluency and must be a part of any class that uses the name.

The section covers the following topics:

- Class Goals and Structures
  - Alternatives
  - Goals
- Classifying Material
  - Skills
  - Concepts
  - Capabilities
- Teaching Capabilities and Class Demonstrations
  - Specify the Guidelines
  - Set Up a Demonstration
  - Multiple Examples
- Designing Projects
  - Choosing a Project
  - Assignment Structure
  - Plan for Grading
- Teaching Programming
  - The Language
  - Programming Instruction
  - JavaScript Specific Issues
  - Conceptually Hardest

Class Goals and Structures

Fluency can be a component of a curriculum in several ways. At UW FIT100 is a service course offered to “all” students. The Information School requires it for undergraduate Informatics majors and Navy ROTC requires it of all corpsmen. It fulfills UW’s QSR (quantitative and symbolic reasoning) graduation requirement, which motivates some students to take it as an elective. CSE prohibits students from receiving credit for FIT100 once they have taken the first programming course. Anecdotally, students not taking it as a requirement sign up without a clear goal in mind—they just think it’d be good material to know.

Alternatives

Permit a brief digression mentioning the other ways in which Fluency content has been applied at 2- and 4-year colleges:

> *CSE-0* Fluency has been used as a preliminary course towards a computer science degree on the theory that it gives a broad coverage of material some of which students will eventually study deeply, but all of which will be useful before then. (And since Fluency is broader than any of its defining fields, some of the material will never be taught to the majors.) The approach gives students a chance to try out programming in a small way, since Fluency includes introduction to a few programming concepts. Further, for those students who decide CS isn't for them, they come away from Fluency with valuable knowledge useful throughout the rest of their lives.
>
> *Disciplinary Emphasis* Fluency has been taught in the context of specific majors, e.g. business or nursing, as part of a general preparation for the major. Such an approach is desirable—it's how the committee expected Fluency to be taught—because the projects and examples can be customized to the discipline, greatly improving its relevance to the student. But the approach assumes the students know what they will major in, and at UW and many schools, that decision isn't usually made until the junior year. In my view, it's better to teach Fluency earlier than wait, even if it's generic.
>
> *Computing Minor* Some fields use IT intensively, and all fields are adding to their applications of IT. Several schools that have the appropriate courses have pushed the "Disciplinary Emphasis" idea even further by formulating a small course sequence that starts out with Fluency and adds discipline-specific courses that use IT intensively.
>
> *Critical Thinking* Schools that offer courses in critical thinking skills classes as well as basic computer literacy have combined them into one two-term course that merges Fluency's problem solving, information assessment, reasoning, etc. components with the critical thinking goals. A well-developed version of this approach could produce highly effective college students that could incorporate IT into their college life about as seamlessly as they incorporate reading.

There are a variety of other strategies, but these seem to be the most popular.

Goals

The goal of any Fluency course is to teach the Skills, Concepts and Capabilities that students need to understand and use IT today, and for the rest of their lives. (The fact that IT will change dramatically during a student's lifetime explains why the topic list is concept and capability heavy.) The recommended content includes the top 10 topics in each area. These are listed in the accompanying table. As a general goal, it seems that to teach a class with "Fluency" in the title, it is necessary to teach those or equivalent topics, but there is some leeway:

Fluency with Information Technology
Skills
Set-up a personal computer
Use basic operating system facilities
Use a word processor to create a document
Use a graphics or artwork package to manipulate an image
Connect a computer to the Internet
Use the Internet to locate information
Use a computer to communicate with others
Use a spreadsheet to model a simple process
Use a database to access information
Use on-line help and instructional materials
Concepts
Fundamentals of computers
Organization of information systems
Fundamentals of networks
Digital representation of information
Structuring information
Modeling and abstraction
Algorithmic thinking and programming
Universality
Limitations of Information Technology
Social impact of computers and technology
Capabilities
Engage in sustained reasoning
Manage complexity
Test a solution
Find problems in a faulty use of IT
Navigate a collection and assess quality of the information
Collaborate using IT
Communicate using IT about IT
Expect the unexpected
Anticipate technological change
Think abstractly about Information Technology

- Skills—a sufficient body of knowledge to permit people to use a computer effectively today—will change over time and is sensitive to people's personal needs. So, the recommended list is generic and applies to the 1999 timeframe. The committee expected the generic list to change as time passes. For example, "assembling a computer" [Skill Topic #1] seems less important today, and "virus protection" [not on the Skills list] should probably replace it.

- The Top 10 were recommended under the proviso that if subsequent discussion identified worthy topics that had not been included, it should *replace* something

on the list, i.e. the Top 10 was only a priority listing and material not on the list was also important. Subsequent discussion of the Concepts and Capabilities lists has not changed the original recommendations.

- Adding Topics to a Fluency class is possible if time permits.

Since the topics have received general acceptance, implying that with unlimited time they would all be taught, the issue comes down to how to teach them in the time available. I try to teach each to the greatest extent possible in the available time.

Classifying Material

The committee's intent was that Fluency instruction should integrate the ten Skills, Concepts and Capabilities, not teach them as 30 separate lessons. Accordingly, structuring the class is essential.

Skills

As noted, the skills required to be a competent computer user change with the person and change over time. They will certainly change with respect to a college's or a department's goals. In my view a generic course like FIT100 should assure that students are prepared to be effective college students and know how to learn more applications. In that way they will maximize the application of IT in their education, and be prepared to learn new tools as their interests change and develop over their college years. Obviously, a discipline-specific approach will add more applications from the discipline. It seems equally obvious that it's impossible to remove any of the generic applications, since people are general users when they are not at work.

The good news regarding skills is that college students generally begin with considerable knowledge. Most, but not all, students arrive at FIT100 knowing how to send email, browse the Web, make a Google query and use a word processor. Many know considerably more, but this slim list constitutes my operational definition of "literate." Since there are no prerequisites for FIT100, it cannot be guaranteed that students will possess this knowledge. So, each term that I teach the class, I schedule a "Welcome" session for two hours on the first two evenings of class for students who are not literate so that they can acquire the needed skills. This allows me to begin class assuming more knowledge, and thereby avoids boring the other students. In the early offerings, students came to the Welcome, but in recent years they have not; they believe they are literate. (The "digital divide" is a reality, but I conclude that it doesn't affect many students who decide to take FIT100 at UW.)

Because students come to class knowing at least half of the Skills recommendations, it is possible to greatly reduce the class time that must be devoted to learning Skills. Specifically, I emphasize the process of learning a new application with the explicit request that students who need to learn a new application pick it up on their own. The process relies on fundamental concepts such as consistent interfaces that are described in

Chapter 2 of *Fluency with Information Technology*. Teaching how to learn a new application handles all but the "most advanced" applications like Photoshop, Excel and Access, which we teach in labs.

Concepts

The Concepts are the "book learning" part of Fluency, and when I teach it, I rely heavily on the book. For a basic course like FIT100, a book is essential because students are not well trained to read diverse sources or to take effective notes, either from those sources or in class.[*] Further, students are generally well trained at learning from a book. For a typical lecture I assign an entire chapter's worth of reading. The lecture then briefly highlights the most important material from the chapter, usually with other examples or demonstrations, and that's that.

Because most professors will have a set of concepts that are near and dear to their hearts that they would like to include in the course, the book concentrates only on the "core" topics recommended by the NRC report. Moreover, there are topics that simply cannot be covered well in a medium as slowly evolving as a textbook, and therefore instructors will have to augment the material in the book. For example, in the 2003 timeframe when the book was being finalized, MP3 file sharing, Napster and Kazaa, iPod, Recording Industry Association of America lawsuits, etc. were all too fast moving to be included in the text. The text does include discussions on intellectual property and copyright that underpin the topic, but instructors must augment the lectures each term with the latest situation.

I like to include additional topics, often chosen from the news. For example, I finished a lecture with a short discussion of Google Bombing on the Monday following the New York Times's first coverage of it. Such topics, which are of intense interest, relate the course's material to contemporary usage.

Capabilities

The NRC report describes Capabilities as forms of higher-level thinking applied to IT problems. They include everything from sustained reasoning to thinking technologically. On the one hand teaching capabilities is difficult—it is a challenge to teach someone to reason better. On the other hand, teaching Capabilities, which is often done by example, can be a lot of fun. Most of my teaching of Capabilities is done through lecture demonstrations. So, for example, after presenting the six guidelines for debugging, I go through a demonstration in which the HTML of a busted Web page is debugged. Typically, such demonstrations are very engaging for many students, as they try to make progress on the demonstration while it is progressing. There is apparently no effective way to demonstrate that teaching Capabilities by example works, but I like to think that it is a way for me to add value to the class.

The bottom line on teaching Capabilities is this: Teaching Capabilities applies the instructor's teaching abilities most completely, and is the greatest fun. Of course, learning

[*] The book hasn't always existed, of course, and teaching the material in lecture form is sufficiently onerous, that being able to rely on the book has greatly improved the course from earlier offerings.

the Capabilities is of greatest value to the students, too, because the knowledge generally applies widely beyond the limits of IT. The next section gives a discussion of the appropriate teaching techniques.

Wrap-Up. In summary, students arrive having many skills, they are taught how to learn new skills on their own, and a few more "advanced" skills are taught in lab. Only one lecture is devoted to the Skills component of Fluency. Concepts rely heavily on the reading with lectures used to reinforce the material learned. Capabilities emphasize the dynamic, discussion-rich teaching through real time application of IT.

Teaching Capabilities and Class Demonstrations

An essential teaching aid for Fluency is a video projector displaying the screen of the instructor's laptop or a classroom computer, because it allows for interactive demonstrations. (The physical setup of video projected computers is sometimes very unsatisfactory for keyboard interaction and programming, but the value to the students is so great in my opinion, that it is worth whatever contortions and inconvenience the instructor must endure.)

The capabilities component of the Fluency curriculum includes a series of the higher-level thinking abilities such as logical reasoning, debugging, problem solving, complexity management, critical thinking, thinking "technologically," etc. All of the capabilities transcend IT, applying to a broad range of situations. Indeed, the capabilities are a goal for much of education generally, and if students acquired them, there would be much less to teach because students would be independent learners. And that's the reason the capabilities are included in the Fluency curriculum, to ensure that students can learn on their own the IT they need to know after they leave the Fluency class.

But capabilities are difficult to teach. First of all, they are by definition non-algorithmic: It is not possible to give an algorithm for reasoning, or debugging or any of the other capabilities. Since there is no step-by-step process, how does one do it, and how does one teach it? Second, the capabilities are difficult even for the accomplished. Reasoning, debugging, etc. are challenging intellectual activities. They give us "brain fry" and many of us avoid them when possible. Students have to work hard to learn them and work hard to apply them, and naturally they'd prefer to avoid them, too.

But there are techniques that simplify capability instruction.

Specify the Guidelines

There is no algorithm for any of the capabilities, but they have structure. These take the form of guidelines—rules that can be applied as appropriate. In *Fluency with Information Technology*, for example, the following guidelines are given for debugging:

- Make sure that you can reproduce the error.
- Determine exactly what the problem is.
- Eliminate the "obvious" causes.

- Divide the process, separating out the parts that work from the part that does not.
- When you reach a dead end, reassess your information, asking where you may be making wrong assumptions or conclusions; then step through the process again.
- As you work through the process from start to finish, make predictions about what should happen and verify that the predictions are fulfilled.

Obviously these are not algorithmic, and must be applied with a certain amount of judgment and creativity, but they do provide a place to begin and a schema for working through the process. Since all capabilities can be abstracted into a set of guidelines, present the guidelines first. (I advise students to memorize the debugging guidelines so they are ready when they get stuck and wonder "What do I do now?")

Set Up a Demonstration

An online demo is perhaps the best way to present capabilities. Because there is no cut-and-dried procedure to follow, it is perhaps easiest to show the process as a context for talking about it. There are several considerations.

- *Careful Planning*. To be successful the demonstration must be well planned. Typically this means that the problem is completely solved ahead of time, and to expedite parts of the show, portions may have to be prepared in advance to avoid long, tedious, uninteresting typing. For a debugging demo, for example, bugs are typically planted in a working system, and then they are "discovered." If a conjectured fix requires quite a bit of typing, it could be prepared ahead of time.

- *Stumbling Around*. One characteristic of the guidelines is that they are not guaranteed, so it is natural to have false starts, and draw wrong conclusions, etc. So, although the demo must be well prepared to work right, it cannot be so well prepared that it loses the spontaneity, false starts and fumbling. The students need to see those things just as much as seeing the successes.

- *Correlate with Guidelines*. The best demos correlate to some degree with the guidelines introduced initially. It is not usually possible to do a perfect job of this; first because problems typically don't require the use of all guidelines, and second it may be that the solution needs key reasoning not codified into guidelines. Still, it is best to connect the specifics of the demo with the abstract ideas that are being illustrated.

- *Interactivity*. Perhaps the most challenging part of doing an effective capability demo is managing class interactivity. The students should be participating in the activity, but by the nature of a capability there will likely be many more ideas and suggestions than anyone can follow at one time. An engaged class will bombard the instructor with input. The instructor has to winnow the ideas to develop the (planned) thread of the solution. More importantly he or she has to bring the members of the class along *who are not thinking about the task the same way*. If these students don't understand what is happening then the demo will have largely been wasted for them.

- *Rate of Presentation*. After planning the demo carefully and noticing that some students are thinking about it in the same way you are, it is very easy to assume "we're all thinking about this the same way," and go too fast. Carrying the rest of class along is essential, and that may take time. But it is possible to go too slowly, too, especially if there is a very "thick" student trying to follow along. Such students may need further instruction after class.

- *Too Many Eyes*. Another problem is that collectively, the class homes in on a solution much faster than intended. For example, someone might notice a bug in a program almost immediately, making it difficult to "find" if it's been announced in a loud voice. Such rapid discovery is more likely in reasoning, testing and debugging situations than in design situations, but it is always a risk.

- *Post-class Summary*. In general, teaching capabilities by demos is too unpredictable to prepare a summary of the solution for students before the actual demo. It usually unfolds in unpredictable ways. However, a useful aid to students making a sincere effort to learn the capability is to post a Web page that recapitulates the logic that the demo exhibited. (It may be even better to have the students produce the description.) This is a good time to bind the guidelines, where possible, to the steps illustrated in the demo.

See Lecture #7 below for further commentary on the debugging demo specifically.

Multiple Examples

The capabilities are not learned in one example or one demo. The best situation is for students both to have seen many examples and to have tried the capability themselves. (The labs are ideal for this latter goal.) Usually, there will be many chances to apply the capability to problems that arise in the process of learning IT, but those are usually not "controlled." So, it is best if there can be exercises to practice. For example, after one or more debugging demos, ask students to take a working Web page, make some subtle changes to it to "mess it up" and ask their partner to debug it, perhaps coaching on the use of the guidelines.

Designing Projects

Fluency projects are multi-week exercises that are designed to give students the chance to apply the three kinds of knowledge taught in Fluency class: Skills, Concepts and Capabilities. Effective projects push students, especially in the use of

FIT100 Projects
- Build a bogus Web page in HTML
- Program a Concentration Game as a JavaScript application
- Design Database AIDS Testing Clinic

higher level thinking abilities (Capabilities). But experience shows that writing the assignment is a significant task that requires considerable care, or the students will be floundering, not pushed. But there are guidelines for structuring and specifying a project.

Choosing a Project

Certain guidelines should be kept in mind when picking the topic and the task for the project.

- *Assess Difficulty*. Projects are multi-week homework assignments, so they must be substantial. But, it is extremely easy to be too ambitious, both in terms of conceptual difficulty and effort. Caution must be exercised. The most dangerous situation is formulating a project on a topic of interest to the instructor, since familiarity obscures difficulty very easily. Let me repeat that remark: *It is most dangerous to formulate a project based on a topic of interest to the instructor!*

- *Produce a "Product."* Projects should produce an IT "product" at the end so the students have a clear goal to work towards, and for which there is a clear test for completion. Examples are a set of Web pages, a database, a program, a presentation, etc. (This correlates with how IT is used much of the time.) The product doesn't eliminate the possibility of including exercises along the path to the end. Nor should it be the only thing that the students turn in for grading. Asking for assessments or other writing exercises related to the project test whether students understand what they have done.

- *Emphasize Interesting Content*. We all have different interests, and a professor's interests will likely be different than those of 19-year old students, but the students will have to devote a lot of energy to the project, so the more interesting the better. Avoid "math" or "technological" topics—non-techies find them dull and often intimidating. Emphasize "glitz" or operations that the students' friends might not know how to achieve. For example, in a binary search for a birthday starting with a zodiac sign, the computer asks questions that "switch month" like "Is your birthday after August 3?" followed by "Is your birthday after July 29?" This is easy to accomplish, but most students don't immediately see how to do it. You will teach it, the student learns the trick, and then later the student will look smart, too. It goes without saying that the project cannot be all glitz!

Assignment Structure

There are several considerations in formulating the assignment.

- *Clear Series of High-Level Steps*. A task like building a database is too daunting for students at this level, even if they know how to use the tools. A good project strategy is to formulate a clear sequence of steps to achieve the goal, and then present (or better, derive) them in class—perhaps with a discussion of problem solving techniques and student participation—prior to handing out the assignment. That way the top-level structure will be commonly understood. A teaching goal would be to make the logic of the steps plain enough that students

could formulate an analogous top-level strategy when confronted with a similar task.

- *Control Step Level*. The student's idea of a good project is one in which they are told *exactly* what to type at each step, especially if it involves using lots of easy wizards. (If you gave them such an assignment, however, they'd complain bitterly that it is not teaching them anything!) The teacher's ideal project is one in which the student implement's each step without any explanation beyond what the step is to achieve. The project's steps must fall somewhere in between, and formulating them to have the right level of challenge is the hardest part of creating a project. A typical step in a project describes the inputs, the goal, and how it contributes to the overall solution. But they are subtle to get right. Clearly "Declare a variable with a name of your choosing and initialize it to the number of inputs at the start of the computation" is better than "type **`var numInputs = 0;`**" But if the directions are too vague, students can easily get lost. Perhaps the most useful techniques are

    - Define terminology (even naming parts of the project that wouldn't usually be named) and use it consistently
    - Be very explicit about the continuity of the solution at the start and end of steps, reviewing how far we've gotten and where we're going at each step
    - Limit the generality of the solution when doing so simplifies the student's effort

- *Multipart Turn-in*. Many of us put off working on difficult problems, and so it is natural for students to wait until the last minute, not appreciating that the whole point of a project is that it is so large it can't be done in one sitting. Plan on one or more intermediate milestones at which students turn-in work for grading. As a safety net for students who fail to achieve a milestone, it is helpful to distribute a solution for the milestone after it has been turned in.

- *Copy-free Components*. In IT it I easy to copy homework, and students pressed for time are tempted to do so when the crunch comes. It is especially easy to copy and get away with it when the steps of the assignment can be done in only one-way: How can you distinguish between solutions? So, make project components as "personalizable" as possible. Commenting database designs or programs is one way to personalize work that has a limited number of options. Requiring student thought-up names also helps. Good projects maximize the number of features that exhibit the personal touch.

- *Extra Credit*. Opportunities for getting extra points on projects are usually appreciated by students, especially those who are worried about doing well. There are typically many places to add credit—for example, more general solutions, fancier solutions, etc.

Plan For Grading

There are a variety of ways in which the grading effort can be simplified.

- *Commentary*. An excellent approach is for the student to show the grader the solution in the lab and then answer verbal questions about it. (Unfortunately this is suitable only for small classes.) The technique virtually eliminates any advantages of copying since students asked to explain work they copied are invariably unable to do so.

- *Attend to Naming*. It is often handy if all projects have a common name such as `.../<studentID>/proj1.html` as an aid to grading. Deciding how naming conventions can help should be planned out. (Note that such uniformity can also be dangerous, as when the consistent name requirement makes it easier for someone to copy a solution like a published Web page.)

- *Test Inputs*. It is often helpful both to the instructor and the student if there is a standard set of test inputs or criteria that the solution is expected to pass. The test cases provide a sharp definition of what is a complete solution.

- *Solution + Standard Grading Form*. Since grading online is easy for evaluating computer execution of a solution, but difficult when giving students comments and feedback, it is helpful to have both a hardcopy (for comments) and a softcopy (for executing). The overall scoring can be done on standard document that lists the criteria for grading, the number of possible points, and options for extra credit.

Teaching Programming

Programming is the act of specifying a process precisely enough that some other agent (usually a computer) can execute it. Though this standard definition is correct, it misses most of what is hard about programming, namely the abstract and logical thinking that goes into formulating the specification. Add to that the fact that we usually use a programming language that students are learning at the same time, and there are complications galore. But there are techniques that can help.

The Language

There is a certain interdependence with learning programming languages—the first one is hard to learn without knowing how to program, but learning programming requires knowing the language. Breaking the dependence means emphasizing concepts.

- *Names/Values*. The first difficult concept is appreciating the fact that names in programming can change values. A good heuristic is to avoid referring to common nouns, and use offices or roles such as president, mayor, poet-laureate or Juliet. These are names that change values as different people serve or play the parts. Variables in programming languages work this way. Also, students will

already be aware that file names work this way when they Save in applications: The file name now has a different (updated) value.

- *Assignment*. Unfortunately, because variables look like unknowns in algebra, students will often listen to the explanation of changing values but continue to think of them as unknowns. This makes it probable that they will listen to the discussion of assignment statements, and interpret it as describing equations. So, teaching assignment must forcefully emphasize the right-to-left value flow, and that the variable is changing values. It is best to pronounce the '=' as "becomes," "is assigned" or "gets." And it is helpful to point out that x=x+1 in programming tells the computer to increase the value of x by 1, but interpreted as an algebraic equation is a contradiction; no number is equal to itself plus 1.

- *Syntax*. When introducing language constructs, it is best to begin with the syntax of the statements in general form. For example

```
if (<predicate>)
    <then-statement>;
```

  Then *immediately* give a series of examples. Illustrating the construct through examples teaches the idea quickly, and avoiding all of the usage rules allows students to concentrate on what the construct does. Later, after students are familiar with the construct, it is a simple matter to cover all of the usage rules.

- *Examples*. It's probably impossible to show too many programming examples.

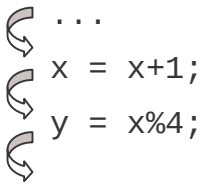
- *Milestones*. Because there are so many concepts to learn (even with the modest goals of Fluency), a good strategy is to introduce only enough concepts to do an interesting program, to illustrate the ideas fully with one or more example programs, and then to iteratively add concepts followed by more example programs.

See lectures 11-14 below for further discussion on teaching programming.

Programming Instruction

There are several techniques for explaining how programming constructs work.

- *Flow of Control*. The statements of a programming language are commands to the computer, and the sequence in which the commands are processed is called the *flow of control*. Students need to know that the statements are executed first to last, and how conditional and iterative statements work to break this sequence. Arrows are a common way to exhibit flow of control:



```
...

if (x==y)
    area=pi*d;
x = x + 1;

for (i=0; i<n; i++) {
    ...
}
```

Functions invocations cause control-flow change as well. Finally, within statements, there is an order-of-execution. For example, in an assignment statement the right hand side expression is evaluated in its entirety and the value is then assigned to the variable on the left hand side.

```
x = 2 * x - 1;
```

Suppose x has value 5 before the statement

In executing the statement, the 5 is multiplied by 2 resulting in 10; then 1 is subtracted from 10 resulting in 9; finally, 9 becomes the new value of x

So, after executing the statement x has value 9

The sequence of these operations is important.

- *Data Transformation*. Programs change the values of their variables to produce the computation. Students need to see this progressive change of values. Thus, a good teaching technique is make a list the variables, and then as the program is executed by hand, to show the successive values of each of the variables.

x:	~~5~~	Value before statements execute
	~~9~~	After executing `x = 2*x-1;`
	3	After executing `x = x/3;`

This is literally what is happening inside the computer and it is a good image for the students to carry in their minds. (Tools animate this process, too.)

- *Incremental Revision*. A good programming technique, apart from teaching, is to use a program-and-check iteration, where each small change to the program is tested before moving on. Students should be encouraged to program this way, and it is a natural way to teach the concepts in a class demonstration. The benefit is that a well-designed example can show a substantial amount of a computation in a way that doesn't require a great effort for any one part. Indeed, the demonstration might even print the successful values described in the last bullet.

- *Program Modification*. An easy way to get students started—especially in terms of giving them the satisfaction of producing a result with a program—is to have them modify an existing program. This is very effective initially, especially if the program they are given is easy enough for them to understand entirely. (They won't try to understand it initially, but usually try when they're working on their own.) Though it is a good crutch to get started with, there are two risks: First, it is easy for students simply to copy text from a previously working program with no idea what it's for except that they need it; second if the program to be modified is too large or contains content they cannot understand, the task ceases to be a crutch and becomes overwhelming.

JavaScript-specific Issues

JavaScript is at once easier than many programming languages and in other respects harder. It's easy because it is interpreted inside a browser, which makes both syntax and execution quite forgiving. The problem is that JavaScript is mixed with HTML, which blurs the boundaries of the language.

- *Generic Host*. Since JavaScript must run in HTML, a good way to get started running programs is to use a generic HTML file in which the JavaScript is placed. To avoid having to introduce forms and all of the complexity of input/output, hard code the input as initial values for the variables and use the `alert()` function for output:

```
<html><head><title>EZ</title></head>
  <body>
   <script language='JavaScript'>
       program specification goes here
     alert(result of computation goes here);
  </script>
 </body>
</html>
```

- *Forms*. Unfortunately, forms are a large topic that must be taught in the midst of teaching JavaScript. Teaching forms before JavaScript is tough because they are crippled without JavaScript event handlers. Delaying teaching them until later handicaps the JavaScript programming since there is no proper input/output without forms. So, forms get in the way, but there isn't much that can be done about it. (Input/Output has always gotten in the way of teaching programming!) One strategy is to introduce forms and ignore the fact that the event handlers require programming … just set up a form and compute on its values. This gives an intuitive introduction to a few programming concepts and gets past forms.

- *Constructing On-The-Fly Pages*. A conceptually difficult problem for students is to understand when text is added to an HTML file from a JavaScript program using `document.write()`. The answer is, of course, "at the next position in the HTML file," but it is difficult for students to follow the processing of an HTML/JavaScript page. The following diagram might help:

```
<html>
<head><title>Explain</title></head>
 <body><p> The browser reads the
     HTML before it creates the page.
     When it comes to a script tag it
     processes it immediately. There
     may be document.write()s and
     if so, it writes the argument
     <script language="JavaScript">
         document.write("into the file");
     </script>
     at the point of the script tags.
  </body>
</html>
```

HTML Source Text

```
<html>
<head><title>Explain</title></head>
<body><p> The browser reads the
     HTML before it creates the page.
     When it comes to a script tag it
     processes it immediately. There
     may be document.write()s and
     if so it writes the argument

     into the file

     at the point of the script tags.
  </body>
</html>
```

HTML File after preprocessing

Conceptually Hardest

Overall experience indicates that the greatest conceptual difficulties in programming stem from the following sources:

- *Names Changing Values*. Variables are names that change values. The “offices and roles” formulation helps illustrate this, but it doesn’t sink in right away.

- *Assignment As Value Change*. A value flows “into” a variable from the right hand side in assignment statement. Again, this idea doesn’t sink in fast and confusion with algebraic unknowns can persist.

- *Indirection*. Students often miss the indirection implicit in using variables, i.e. that `a=b+c;` can compute any sum as long as `b` and `c` are assigned the appropriate numbers. This is probably due to interpreting the text as if it were algebra, missing a variable’s value change and assignment’s value flow.

- *Functions*. Because there is so much to the concept of functions, students find the idea difficult because there is so much to comprehend before any of it can be applied. There seems to be no slow-and-easy treatment.

- *Declaration/Call*. The declaration of a function and it’s call are easily confused. The “textual replacement” explanation—a call behaves as if the text of the declaration were inserted at the sight of the call and the arguments are assigned to the formal parameters—seems not to help so much.

- *Parameters/Arguments*. The indirect reference of arguments via parameters is difficult.

- *JavaScript Interpretation*. Students find the processing of the HTML file and the evaluation of JavaScript—sometimes interpreted immediately, sometimes part of a function or event handler interpreted later—as confusing.

Course Structure

Teaching FIT100 in ten weeks is a challenge because there is so much information. The key is to be well organized, arrange topics so that they flow smoothly, and make your expectations explicit to the students. Achieving these goals will make the class reasonably successful even if other things go wrong, which they surely will.

The principal considerations for structuring the Fluency course:

- *Labs Interleave with Lectures.* There are three lectures and two labs per week. They should be arranged so that all students have seen the same number of lectures when they attend a given lab. The best way to achieve this is to schedule labs on Tuesdays and Thursdays if the lectures are Monday-Wednesday-Friday. This may seem obvious, but when scheduling pressure for labs becomes an issue, invariably someone will want to schedule a lab before the Monday lecture or after the Friday lecture. (Of course, other scheduling plans should also achieve the "same number of lectures" goal.)

- *Fluency is a project-based course.* Recall that the projects integrate the Skills, Concepts and Capabilities that the students are learning. Projects not only integrate the main ideas, they also represent typical applications of IT. That is, most of our uses of IT tend to apply several of the constituents of the Fluency curriculum, just as projects do. Accordingly, the projects dictate the organization of the Fluency class.

- *Projects Require (Some) Preparation.* In order for a project to integrate material, the material must have already been taught when the student is asked to do the project. In order to complete the three projects in a quarter, it is helpful for the students to learn the material (in lecture) for the next project as they finish up the current project. This implies a pipeline structure in which the course begins with preliminaries and then prepares for the first project. Once the first project is assigned, preparation for the next project begins, etc. This rigid form is not *literally* followed because it usually doesn't take two-three weeks to prepare for the next project. So, typically, after a project is assigned, the next several lectures are devoted to "non-project content." Then as the next project assignment date looms, project-related instruction begins again.

With those considerations the course can be structured as shown in the table.

Num	Title	Reading	Assigned	Due
1	Welcome	Syllabus	Scavenger Hunt	
2	*Le Mot Juste*	Chapter 1		Scavenger Hunt
3	Digerati	Chapter 2	Mac (PC) Attack	
4	Networking	Chapter 3		Mac (PC) Attack
5	HTML	Chapter 4		
6	Mis-Information	Chapter 5	HTML Project	
	ML King Holiday			
7	Debugging	Chapter 7		
8	Digital Representation	Chapter 8		Project 1, Part A
	Midterm 1			
9	Computer Basics	Chapter 9		
10	Algorithms	Chapter 10		Project 1, Part B
11	Programming Basics	Chapter 18	JavaScript Proj	
12	Making GUIs	Chapter 19		
13	Functions	Chapter 20		
14	Iteration	Chapter 21		
15	Animation	Chapter 22		Project 2, Part A
16	Big Picture	Review 18-22		
	Presidents Day			
17	JavaScript Summary			
18	Digital Media	Chapter 11		Project 2, Part B
	Midterm 2			
19	Data Base Principles	Chapter 13		
20	Views on Tables	Chapter 14		Project 2, Part C
21	Creating a DB	Skim Chapt 15	Data Base Proj	
22	Polite Users	Chapter 12		
23	Privacy	Chapt 17, first		Project 3, Part A
24	Encryption	Chapter 17 last		
25	Do Computers Think?	Chapter 23		
26	It's a Wrap	Chapter 24		Project 3, Part B
	Final Exam			

Notice that the chapters of the book are not all covered and that those that are covered are not covered in order.

Assignments

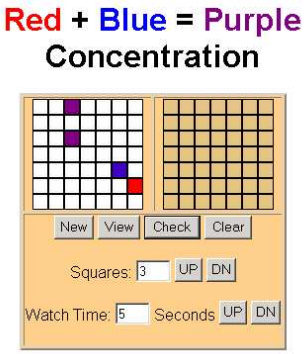
The assignments are

- *Scavenger Hunt*: Students must answer questions based on contents from the Syllabus and the class Web page. The purposes are (a) to assure that they've read various class rules, e.g. the late homework policy, and (b) assure that they become familiar with the structure of the Web page, which they will use all the time.

- *Mac (PC) Attack*: Students are to work on either a Mac or a PC, selected as the type of machine they do not usually use, i.e. PC users work on a Mac. The purposes are (a) to emphasize the "consistent interfaces" feature of GUIs, and (b) to break down the barriers to trying other machines.*

- *HTML Project* [Appendix A]: Students are to use HTML (not authoring software) to develop a bogus Web page, that is, a page that presents wrong or misleading information. The page must include a photograph that has been altered to change its meaning. The pages are clearly labeled with a "bogus icon," to avoid misleading readers who happen onto the sites. The purposes are (a) learn HTML, (b) learn Photoshop, (c) understand copyright limitations, since any photo used must be used legally, (d) appreciate how easy it is to create unreliable content, motivating skeptical browsing. (Authoring tools can be used after this assignment.)

- *JavaScript Project* [Appendix B]: Students use JavaScript to program Concentration Game in which randomly chosen squares are displayed in a 7x7 grid briefly, and when they have disappeared, the player must click on the buttons (right grid) in positions that had formerly contained squares. It is possible to control the delay and the number of squares. The purposes are (a) learn programming concepts as expressed in JavaScript, (b) practice reasoning, (c) understand event-based programming, (d) understand the basics of animation, (e) practice managing complexity, (f) practice debugging, etc.



- *Data Base Project* [Appendix C]: Students create a data base to help a small Washington community's police department keep records on Driving While Intoxicated arrests. Students begin by finding out what happens in a DWI arrest. They design and construct (in MS Access) the physical data base tables, produce queries for the logical tables, and construct GUIs to display the tables. The purposes are (a) learn the principles of data bases, (b) experience designing a data base, (c) practice constructing tables, SQL queries and forms, (d) think about usability and convenience of user interfaces.

Other offerings of the course have used a somewhat different set of assignments and projects, but these are representative.

To illustrate the "pipelined" structure of the course, notice that prior to the assignment of Project 1 at the end of lecture, the antecedent material is taught in lectures 3-6, namely, Networks, HTML and Mis-Information.

* Sadly, most students, being PC users, think the Mac was bizarre and unintuitive!

Tests

In addition to the assigned reading and homework, the class has pop-quizzes, two midterms and a final.

- *Pop-quizzes*. From time to time these have been scheduled quizzes in an effort to improve the students' performance and lower the anxiety. I don't think either goal was achieved, so they're returned to "unannounced" status. The questions are very simple, selected from the day's reading. (Being Freshmen mostly, the students tend to do poorly until the last quiz when they have the idea of how things will go.) The quizzes are allotted only a few minutes (10 maximum), are collected and then the solutions are given.

- *Midterms*. The point of the midterms is to keep students current with the reading, distill the content of the projects, and assess how the class is progressing. As a rule the midterms are a lightening rod for student complaints, even when the class is going well or perhaps especially when the class is going well.

- *Final*. The final is mostly a cross check to make sure students worked hard enough on the projects that they can answer specific questions about the implementations. There is a limited check that content has been learned, but considering that there is an enormous amount of detail that is not worth committing to memory (and should be looked up when needed), it seems unnecessary to conduct a very elaborate check.

Obviously, the large class format implies less essay and more "objective" type questions.

Logistics
Ordering activities; pointer and microphone batteries; getting pages up; turn-ins and grading, scheduling collisions and sharing labs, TA assignments and meetings

Sampling student opinion

Anonymous Email and other stuff of the web page

Resources including Catalyst

Labs

Because the class has a large "learn through doing" component, the labs are extremely important.

Each FIT100 lab is closed; one TA and 25 students are assigned, making six sections for a class of 150. (The labs originally had only 25 computers, which defined the lab size, but now they have 40. Nevertheless, it is difficult for a teacher to give personalized lab instruction to many more than 25 students, so we have resisted the inclination to expand

the size.) In addition to the student machines, the lab has an instructor's machine and a video display device so that TAs can display the material on their screens. The lab session runs 50 minutes. Each machine is a (reasonably new) Windows PC with Internet connection, standard software, and external drives. In addition each student has file storage space on a campus file server and Web server space.

During the term labs change character as the students' needs change. I would identify three different formats for labs in FIT100:

Skills Labs
Explore a Concept/Capability
Project Work

In all cases there is a document available to support the lab activities, though in the last instance it is the project specification. Lab documents are usually in `.pdf` and posted prior to class, so students can see what is coming. Students are expected to have a copy displayed on the screens in lab.

Consider each format separately.

Skills Labs

Some information must be taught in a detailed, click-here, click-there, way. For example, file transfer between the student's lab machine and the file storage on the campus file server is an essential skill.[*] An early lab teaches the FTP application by explaining what the idea is, demonstrating how the application works and giving students experience applying the tool.

The structure of a Skills lab is as follows:

Introduction and Overview. The TA explains the day's agenda and gives a description of the overall concept. It is not assumed that the students have encountered the information before, and so the explanation must be reasonably complete.
Demonstration. The TA then demonstrates the application, explaining what is happening as the demo proceeds. It is advisable to illustrate all components of the day's lab at once, so that there is only one transition between demo and exercise.
Exercise. The students then try out the application on their own. The initial exercise usually has a common basis—everyone is trying to achieve the same result—so that the questions and comments will be generally understood by all. There may be several exercises.
Independent Work. Labs usually finish with a self-directed task that students do on their own. This is often checked by the TA on completion.

Lab projects are not usually graded, but a 0-2 grading scale is frequently used:

[*] FTP is surprisingly difficult and requires further review after the lab before the concept is firmly set.

0, *no score*
1, *incomplete or inadequate solution*
2, *full solution*

Mostly, the purpose of checking is to assure that students complete the work and don't get far off track.

Concept/Capability Lab

The Concept/Capability lab is used to reinforce previously taught material. Such labs are used frequently in the early stages of learning programming. For example, after the animation lecture, students program their first animation in the lab. (See Lecture 15.) The reinforcement is achieved by means of an exercise related to the lecture being reinforced.

The principal differences between the Skills labs and a Concept/Capability lab are that in the latter the material has been taught previously and there is less focus on a specific application. This means that the TA's main goal is to give a quick review of the material, and then turn the students loose on a reinforcing exercise. Because of their independent work on a common task, the student's questions will be a mix of individual queries and general questions of interest to the whole class. The TA has to decide whether a particular question has an answer of general interest or not.

It is not expected that the Concept/Capability labs will be completed within the allotted 50 minutes. So if the work must be checked off, students who don't finish are expected to get it checked at the next lab. Thus, students are strongly motivated to get the work done during the lab period to avoid outside of class work, but those that are struggling can get individualized help at office hours and recover on their own.

Project Work

In a typical two-week project the labs of the second week are often used as an opportunity for students to work on completing the project. There will be little new-project related material that must be covered, and more drill may be redundant. Unquestionably, the main advantage is that the project carries a significant time burden, and any chance for the students to work on it "during class time" is usually appreciated. Further, working on it in the lab means that not only can the TA give help, but it is possible that other students can help explain questions, too.

There is little formal structure to a Project lab.

Because of the independent-study nature of the Project labs, some students figure out that they can get additional "lab time" by attending another section's lab. This is possible because not all of the machines are being used in the larger labs. Because having additional students in the lab spreads the TA's attention more thinly across the class, we do not encourage students to "crash" the labs. On the other hand, we don't discourage it, either.

Logistics of FIT100

Because FIT100 is a 150-student class, it is a fact that the greatest determinant of success is organization. If students know what is expected of them, if their learning path has been carefully prepared, and if the materials they need are available on time and suffice, they will be successful and the class will be successful. That may be depressing for those of us who think of ourselves as gifted pedagogues—shouldn't our great teaching techniques most determine the learning? Sadly, no. It's not possible to wing it and succeed.[*]

This section describes components of the course that contribute to its organization.

Timeline

The issues surrounding logistics are perhaps best organized as a class timetable for class preparation. The main problem with presenting a timetable is that we all have different comfort levels regarding how far in advance things have to get done: Some of us can wait to the very last minute, others of us need to have everything in the can a semester in advance. Everyone will have to adjust the following schedule to his or her comfort level. My comfort level has no apparent pattern and the following times must be adapted to holidays and school practices.

In the following table grading is not mentioned because it is a reality of every week of the term. Further, specific postings are not mentioned, such as announcements, extra materials from lectures, etc.

FIT100 Week-by-Week

Time	Task
-1 year	Verify that the schedule shows the lecture times and the lab times interleaving, i.e. all students at their n^{th} lab section meeting will have had the same number of lectures.
-2 months	Verify that the textbook has been ordered and will be available before the start of class.
-1.5 months	Verify that the labs will have the necessary software installed. Confirm all mechanisms relative to student accounts and student access to labs.
-1 month	Meet with the TAs to (a) choose a TA meeting time for the term, (b) assign lab sections, and (c) select office hour times. (The meeting is a great time to describe the class and explain your expectations for the TA's.) If there will be a Welcome Session on the first two evenings of the term for students who have never touched a computer (see Guidelines below), schedule the lab time and identify the TA to cover it.

[*] This is not the smug assertion of an organization freak trying to convince his colleagues to become better organized, but rather an appeal from a disorganized researcher who has (finally) learned from his betters.

-1.5 weeks	Build the class Web page. The page doesn't have to be fully populated, but certain critical items must be defined, some of which will require further work: • Syllabus, describeing class content and policies, including cheating policy • Room locations and scheduled times for class, labs, staff and, if applicable, Welcome • Textbook and any other materials students must buy • Calendar showing a day-by-day description of the term including all dates with holidays and links for assignments, lecture slides, auxiliary materials (e.g. Web pages), and due dates; prior to the start of class complete detail is needed for at least one full month, though full detail will be needed quickly thereafter. • Anonymous Email link, an anonymized email facility for allowing students to complain
-1 week	Prepare Assignment 1, *Scavenger Hunt*. (Since it requires students to answer questions about the Web page and the Syllabus, it must wait until those parts are in hand.) Prepare Week 1 slides Prepare Week 1 labs
-2 days	Make sufficient copies of the Syllabus handout for the first day of class (there will be a first-day-of-class bottleneck at the copy machine). Prepare the *Mac (PC) Attack* Assignment 2
-1 day	Visit the lecture room and check the size of batteries on the wireless mic; buy spares for it and your laser pointer; store with computer bag
0	Kiss your spouse and family goodbye; promise to write
Week 1	Post this week's assignments on the Web page on the day each is assigned Complete a draft of the Project 1 description – vet with TAs Send email to class email list to verify it is complete – see Assignment 1 Prepare Week 2 slides Prepare Week 2 labs
Week 2	Post Project 1 on Web page on Friday Prepare Week 3 slides Prepare Week 3 labs
Week 3	Complete a draft of Mid-term 1 – vet with TAs Set-up master spreadsheet for class grades using 10-day class list Prepare Week 4 slides Prepare Week 4 labs
Week 4	Complete a draft of Project 2 – vet with TAs Prepare Week 5 slides Prepare Week 5 labs
Week 5	Post Project 2 on Web page on Monday Prepare Week 6 slides Prepare Week 6 labs
Week 6	Prepare Week 7 slides Prepare Week 7 labs

Week 7	Complete a draft of Midterm 2 – vet with TAs Prepare Week 8 slides (Prepare Week 8 labs if they are not reserved for project work.)
Week 8	Complete a draft of Project 3 – vet with TAs Prepare Week 9 slides Prepare Week 9 labs
Week 9	Post Project 3 on Monday Prepare Week 10 slides (Prepare Week 10 labs if they are not reserved for project work.)
Week 10	Complete a draft of Final Exam – vet with TAs Write and post a list of study questions Perform Course Evaluations in both lab sections and lecture
Final Week	Complete grading. Break out the whiskey.

Web Page

I prefer to build and maintain my own Web page for the class. The front page, shown in the accompanying figure,[*] illustrates a fairly convenient structure that I've used throughout the development of FIT100. Find it at

http://www.cs.washington.edu/education/courses/100/04wi/

The content is reasonably self-explanatory, but I'll comment on the following items:

- Image—An archive picture of the campus on a sunny day, rare in winter term, and linked to the CamBot showing the current view across UW's Red Square.
- Announcements—They're on the top of the page as is the date of the most recent posting, so students can remember whether they've read them or not; all announcements that are still current remain, in descending order; the archive of past announcements is at the bottom of this page
- Calendar—The most important link on the page, since it gives the structure of the whole class. See below.
- Computing—Explains where to compute on campus and how to compute at home
- LecsLabsHW—Explanation of how to find this information on the Calendar page
- Projects—These are linked as they are assigned
- Project 3 Turn-in—Online submission facilities for the database; most homework is "posted" to student Web space, and the TAs simply access it.
- Contacts—These three entries have to do with class communication
    - How you doin'?: An anonymous gripe link, posted at midterm to allow students to vent to me about the stress in their lives
    - Bulletin Board: The bulletin board for student discussion and occasional announcements

[*] This is the final form of the page from Winter term 04.

- Anonymous Email: A standard anonymous email facility for students to grip or gossip to me and/or the TAs
- More Info, Files—Repository of files used during the term

Consult the Resources section below for information on Catalyst tools to support some of the above links.

FIT100
Fluency with Information Technology

University of Washington CSE100/Info100 Winter 2004

Most Recent Announcement
24: 14 Mar 2004

Next Turn-in
15 March 2004

Calendar

Computing Lecs, Labs, HWs
Projects
Proj 1
Proj 2
Proj 3

Project 3 Turn-in

Contacts
How you doin'?
Class Bulletin Bd
Anonymous Email

More Info
Syllabus
Vision
Files

Computer Sci
Info School

ANNOUNCEMENTS: Check this space daily for the latest news about FIT100

A. 24: The Turn-in for Project 3B is fixed. Thanks for your patience. Also, there is a correction to the decimal-to-binary conversion algorithm in the book. Find it here and note the red text.

A. 23: The following review questions touch the highlights of the class.

Instructor: Larry Snyder, snyder@cs.washington.edu
Office: 584 Paul Allen Center for CSE
Office Hours: Monday 4:00-5:00

Teaching Assistants:

Name	Pic	Email	Office hrs
Jason Dang		jaydang@cs.washington.edu	10:30-11:30 Tuesday
Justin Perron		waluv1@u.washington.edu	11:30-12:30 Wednesday
Phong Phan		puphan@cs.washington.edu	12:30-1:30 Wednesday
Arpi Shaverdian		arpi@cs.washington.edu	11:30-12:30 Wednesday
Kasia Wilamowska		kasiaw@cs.washington.edu	2:30-3:30 Tuesday

Class: Mary Gates Hall 389, MWF, 10:30-11:20 ... arrive on time

Labs:

AA	MW 12:30-1:20	Justin	MGH 030
AB	MW 1:30-2:20	Kasia	MGH 030
AC	TTh 8:30-9:20	Jason	MGH 044
AD	TTh 9:30-10:20	Arpi	MGH 044
AE	TTh 1:30-2:20	Kasia	OGUL CoL 1
AF	MW 3:30-4:20	Phong	MGH 044

Office Hours [AC abbreviates Paul Allen Center for CSE]

12:30-1:30 Wednesday -- 218 AC	Phong
11:30-12:00 Monday -- 220 AC 2:30-3:30 Tuesday -- 220 AC 1:00-1:30 Thursday -- 220 AC	Kasia
2:30-3:30 Friday -- 220 AC	Arpi
11:30-12:30 Wednesday -- 218 AC	Justin
10:30-11:30 Tuesday -- 218 AC	Jason

Announcement Archive ... outdated announcements are listed here.

Figure: Screen shot of final content of the FIT100 Web Page for Winter 2004 term.

Calendar

The Calendar page, shown below, gives the overview of the class, and in my opinion is an effective form for both the students and the staff to see the course.

FIT100 Calendar

Winter 2004

Note the links to lecture slides and other information. The following color coding is used: Reading for this lecture; Assigned homework; Announcements, due dates and exams.

Sun	Monday	Tuesday	Wednesday	Thursday	Friday	Sat
4	Jan 5 Welcome Printable Lecture Syllabus M/W Labs listed on Tu/Th	Jan 6 Be FIT Browsing Assignment 1	Jan 7 *Le Mot Juste* Printable Lecture FIT Chapt 1 Assignment 1 Due	Jan 8 Directories	Jan 9 Digerati Printable Lecture FIT Chapt 2 Assignment 2	10
11	Jan 12 Networking Printable Lecture FIT Chapt 3 Assignment 2 Due	Jan 13 Command Line	Jan 14 HTML Printable Lecture FIT Chapt 4	Jan 15 HTML	Jan 16 Misinformation Printable Lecture FIT Chapt 5 Project 1	17
18	Jan 19 Holiday (M. L. King)	Jan 20 M/Tu Labs Canceled	Jan 21 Debugging Printable Lecture FIT Chapt 7	Jan 22 Photo Manipulation	Jan 23 Digital Representation Printable Lecture FIT Chapt 8 Project 1A Due	24
25	Jan 26 **Midterm 1** Bring Photo ID (On material thru 1/23)	Jan 27 HTML Project	Jan 28 Computer Basics Printable Lecture FIT Chapt 9	Jan 29 HTML Project	Jan 30 Algorithms Printable Lecture FIT Chapt 10 Project 1B Due	31
1	Feb 2 Programming Basics Printable Lecture FIT Chapt 18 Project 2	Feb 3 Programming	Feb 4 Making GUIs Printable Lecture FIT Chapt 19	Feb 5 Programming	Feb 6 Functions Printable Lecture FIT Chapt 20	7
8	Feb 9 Iteration Printable Lecture FIT Chapt 21	Feb 10 JS Project	Feb 11 Animation Printable Lecture FIT Chapt 22 Project 2A Due	Feb 12 JS Project	Feb 13 The Big Picture Printable Lecture Review FIT 18-22	14
15	Feb 16 Holiday (Presidents Day)	Feb 17 M/Tu Labs Canceled... but Tuesday's labs are open for drop-ins	Feb 18 JS Summary Printable Lecture	Feb 19 Project Work	Feb 20 Digital Media Printable Lecture FIT Chapt 11 Project 2B Due	21
22	Feb 23 **Midterm 2** Bring Photo ID (On material thru 2/20)	Feb 24 Project Work	Feb 25 DB Principles Printable Lecture FIT Chapt 13	Feb 26 Project Work	Feb 27 Views on Tables Printable Lecture FIT Chapt 14 Project 2C Due	28
29	Mar 1 Creating a DB Skim FIT Chapt 15 Classroom Design, no slides Project 3	Mar 2 Access	Mar 3 Polite Users Printable Lecture FIT Chapt 12	Mar 4 DB Design	Mar 5 Privacy Printable Lecture FIT Chapt 17, pp. 456-469 + this Project 3A Due	6
7	Mar 8 Encryption Printable Lecture FIT Chapt 17, last	Mar 9 DB Queries	Mar 10 Do Computers Think? Printable Lecture FIT Chapt 23	Mar 11 Queries+Forms	Mar 12 It's a Wrap Printable Lecture FIT Chapt 24 Project 3B Due	13
14	Mar 15 **Final Exam** 8:30-10:20	Mar 16 Survivor's Party	Mar 17 Survivor's Party	Mar 18 Survivor's Party	Mar 19 Survivor's Party	20

Figure. The Calendar for FIT100 for Winter Quarter 2004

As the heading explains the Calendar is color-coded. Readings are shown in fuscia, live links are in blue (or if followed, lavender), comments are in orange and important information is in red. Aqua is used for assignments BEFORE they are assigned, but once they are linked in, they become blue because the links are live.

I have experimented with placing the current week at the top of the page so students need not repeatedly navigate downward as the term progresses. But we are accustomed to seeing calendars in a certain form, and I'm skeptical that breaking out one week was really useful.

Also, it is easier for the instructor to bind file names to all of the text that will become links during the term even if they are not initially available. Then, the link becomes live simply by depositing the file onto the server. (I typically link in assignments on the day they are assigned, labs a few days in advance, and lecture slides after the fact. See Policy.) However, if the inactive links remain as text, they remain black and the student can notice by the change to blue when links become active. Though convenient, this scheme requires more editing, and I usually forego it.

Resources

At UW the Catalyst Project provides an enormous amount of useful teaching software and instructional resources for faculty and TAs. (Many of these resources are provided to others through commercial products such as Blackboard, etc.) The specific tools that were used in FIT100 Winter 2004 were:

- Anonymous email—used both for the main link and for the How ya doin'? link
- Bulletin Board
- File submission—used for submitting the data base project files
- Dynamic class email list—technically provided by the C&C office
- Student Web Publishing—also provided by the C&C office

In addition, the student-accessible grade archive hosted by Computer Science and Engineering was implemented late in the term. As a result it only provided a convenience to the TAs during final grade assignment.

Consult Catalyst for further information on all of their resources.

Policies

There are a variety of policies required in FIT100. Some, such has how to handle academic misconduct are dictated by the school. Other policies such as handling late assignments are entirely up to the instructor's whim. In this section I give a few policies that have emerged specifically for FIT100 and their rationale.

Posting Lecture Slides

FIT100's large lecture format requires slides and given that they are digital and prepared in advance, students request that they be made available online before class. The purpose is to allow students time to print copies, which they can then annotate in lecture. It may be a good idea in theory, but I don't think it works in practice. My observation is that students don't annotate them. Rather, they look ahead, see what's coming and sleep, missing the commentary and discussion not present in the slides.

So, I refuse to post the slides ahead of time. I encourage students to take original notes, believing that much learning takes place as the lecture goes through the brain and out through the pen. I have no hard data on this point, but my experience is students are more engaged and attentive when they are expected to create notes rather than nap on them.

I post the slides no earlier than 24 hours after the lecture. (One exception is if the lecture and a subsequent lab are tightly coupled. Then, it is posted right after the lecture because it might be needed in the lab.) Typically, a lecture will be posted before the next one is delivered.

This policy is explained on the second day of class. Students will ask to have the policy reversed well into the term.

Turn-in Policies

All homework submissions in FIT100 have a hardcopy component; typically there is also a substantial online component as well. Students include the address of the online component on the hardcopy. The primary reasons for the hardcopy are (a) to provide a medium for such things as student reflection on the assignment, and (b) to have a document to "return." (See Grading Forms below.)

The hardcopy is due in lecture; the online component must be time-stamped by a specified time, often late on the evening before the hardcopy deadline. Students have met the deadlines if the hardcopy has been turned in by the end of class, and if the time-stamp is earlier than the deadline, a condition that TAs check when grading. To assist TAs for Web-posted assignments, students are given a few lines of JavaScript that captures the create time, and incorporates it onto the page. For databases, the "last modified" date suffices.

Late Assignments

FIT100 covers a lot of material in a short time by orchestrating the class carefully. This means that the pace of the class becomes relentless. If a student falls behind by as little as a week, it is essentially impossible to recover. (Students engaged in sports or military activities must study on the road to keep up.) Therefore, late assignments are, with two exceptions, never allowed. (Papers that are slightly late—3-4 hours after class, say—are always accepted, though that fact is not advertised to avoid encouraging the behavior.) The rationale for not accepting late papers is that time spent doing the last assignment

delays the start of the next assignment, making it late, etc. Better to "declare defeat" on the current problem and return to the schedule. To reduce the impact of students "writing off" a bad experience, assignments with multipart turn-ins such as the projects always have intermediate solutions provided so that it is possible to solve the last part(s) without having solved the earlier part(s).

The first exceptions to the "no late assignments" policy are the two assignments of the first week. Class registration is in flux, and students who enter late are allowed to turn in the assignments late. After all, the purpose of the first assignment is simply to become familiar with the class and its resources—obviously everyone should do that. And the second assignment is to reduce phobias between PC and Mac users, also a good thing. Accordingly, the hard deadline is Wednesday of the second week.

The second exception to the "no late assignments" policy involves the concept of a "free late day." For one part of any project turn-in—that would be one of seven parts for Winter 2004—students can "declare" their intent to use the free late day simply by sending email to their TA prior to the deadline for the e-component. This allows them 24 hours after the time-stamp deadline to complete the online portion of the assignment, and it extends the hardcopy deadline either to the next lecture or their next lab. Students most often use the late day during the crunch at the end of term.[*] The most common question regarding the late day policy is whether it can extend over the weekend, i.e. if a Thursday evening timestamp for a Friday turn-in can become a Sunday evening timestamp for a Monday turn-in. The most common answer, actually the only answer ever given, is "no."

Quiz Make-ups

Quizzes are unannounced. They take about 10 minutes at the beginning of class for everyone to answer a few trivial questions about the day's readings. They are used mostly to encourage students to keep up with the reading and to attend class. They cannot be made up. For students with legitimate reasons to be absent—hospitalization, military travel, etc.—the quizzes can be excused. That is, they are not graded, but the number of quizzes used to compute the quiz average is reduced by one. To make the "no make up" policy less brutal, the lowest quiz score is tossed.

Working Together

My goal for students in FIT100 is for them to achieve the goals of Fluency described above. This is often best achieved when the students teach each other. (There are no group assignments in FIT100, because of the usual concerns about clunker partners, difficulties scheduling collaboration times, etc. If more able instructors could resolve these, I think group assignments would be good.) Students are encouraged to work together subject to the requirement that the work they turn in for grading is theirs and theirs alone. The inevitable concern about collaboration leading to inappropriate copying

[*] Once a student who hadn't needed the late day asked if he could get extra credit points for it!

persists, but is lessened by requiring in the assignments a significant amount of "personalization."

Guidelines/Experiences

This section touches on peculiarities of the way I teach FIT100. I wouldn't claim that these are essential to teaching FIT100, or even good teaching practices, for that matter, but they're things I do.

Don't Take This Course

Like "Steal This Book," this section heading is intended to attract attention. FIT100 is not required at UW, many students are eager to take it, and it is taught to only 150 per term. As a result there is, essentially, an infinite supply of students wishing to take it, causing it to be filled quickly. Further, FIT100 is a lot of work and students must be prepared for it. So, on the first day of class I tell them what my expectations are and emphasize that being a five-credit class, I can expect ten hours per week of outside class work. (Student surveys indicate that the actual load is about 8 hours per week.) I remind students that the class is offered every term and tell them that if they think they will be too busy this term to commit to that much work, withdraw and take it in some future term.

A second reason not to take the course, I tell the students, is to get an easy grade. If they've taken computing courses in the past, and think they know all the material, I point out that the course is unique and not presently offered in high schools. Further, though it overlaps with other computing classes, it is a broad introduction that overlaps only a little with any specific course.

Finally, I point out that students expecting to glide through on someone else's work should reassess taking FIT100. It is easy to cheat in IT and it is equally easy to catch cheating in IT. I promise the honest students that we will be vigilant in looking for cheating, and remind everyone that we are obligated to report instances of academic misconduct.

During this discussion, usually as a result of the first item, students get up and leave. There are always other students attending with hopes of getting in, and many do so.

One Handout Only

I pass out the syllabus on the first day and tell the students it is the only paper handout of the term, implying that they must use the Web as their primary resource for course materials. Though verging on excessive given the nature of the course—why not have all materials online?—this is a good crutch for the many students for whom the Web is not (yet) the first resource to come to mind. Assignment 1 builds familiarity with the Web sight.

Post No Assignments In Advance

Part of the orchestration of my FIT100 is to have students prepared for assignments just in time. To avoid their starting on them early—and therefore possibly not benefiting from the preparation or wasting time floundering around on the mistaken idea they can "brain it out"—I do not post assignments in advance. This seems to not be a problem for any students but those with athletic, military, etc. travel constraints.

Questions in Class

Often there is a moment early in the class—before the crush of project work—when a student asks a question with a somewhat complicated answer. Since we're usually doing Web searching at the time, I tend to turn the question into a Web searching assignment. This has been successful in the past, though it didn't happen in Winter 2004, and so isn't in the archive for this class. For example, in an earlier term students wondered why computers use RGB when they were taught red-yellow-blue as the primary colors. The assignment is to find the answer on the Web and turn in a paragraph explaining it to the TA. Since it is usually spontaneous, it is good to have thought ahead of lecture about how it would be handled, e.g. deadline, if such an opportunity comes up.

Anonymous email and complaining

For all of the offerings of FIT100 I have set up anonymous email so students can complain without fear that it will jeopardize their grades. The first thing I discovered is that anonymous email is quite often used by students to praise the course. Since the mail is truly anonymous, this is unquestionably satisfying, since there is no possibility that the student is "kissing up." (Though, it is possible that the student *thinks* I have some way to find out, that I will, and therefore I will give the student some sort of consideration.)

The real reason to use anonymous email is for students to complain when things go wrong. Unclear or too burdensome assignments will always elicit anonymous email. The midterms are also popular targets for complaints. The amount of complaining is a good gauge of how bad the situation is. Though I don't tell the class that will I try to respond when a complaint can be remedied, I do try to fix them whenever I can.

Students will complain about their TAs and this is extremely useful data. (Washington's anon-email has the option for students to cc the TAs or only mail the instructor.) They cut the TAs a lot of slack in my observation, so a complaint is usually something that will require action. A complaint usually means there's fire, not just smoke.

Flame-a-thons and Bulletin Boards

Students commonly get into predicaments involving email or bulletin boards. Flaming can happen, and the wise instructor nips it as quickly (and as positively) as possible. It's hard to generalize on what sets students off—is this deterministic?—but it is certain that they will stay with it if it is not contained. It is a good time to read the flame-a-thon section of Chapter 12 of Fluency.

A worry for bulletin boards—I use them as an electronic forum for students to answer each other's questions—is for some students to answer the questions outright. The

ground rules are spelled out, i.e. the b-board is used only for telling each other *how* to figure out the answer, but it's easy to step across the line. There is the pressure from the struggling student "just wanting the answer," and the tendency from the good student to want to raise his—it's virtually always a man—popularity. Again, this is a problem that should be nipped in the bud quickly and positively.

Summary

FIT100 is an intensive class—check out the timeline! In my opinion it's satisfying to teach. The students who do the work to complete the course are better computer users, and they know it. Some students also change the way they think. Sometimes students learn deep intellectual capabilities, such as reasoning and problem solving. These not only make them more accomplished at IT now and in the future (Fluency's goals), but sharper minds apply across all of the their intellectual experiences.