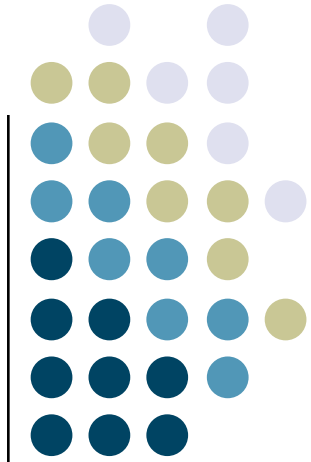
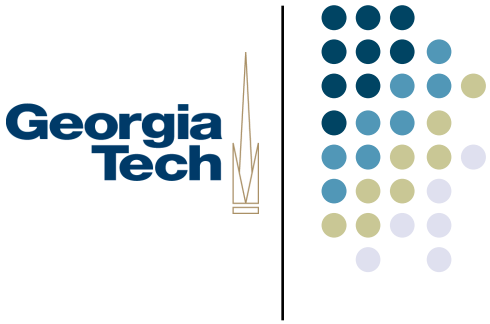
Welcome to CS6452!



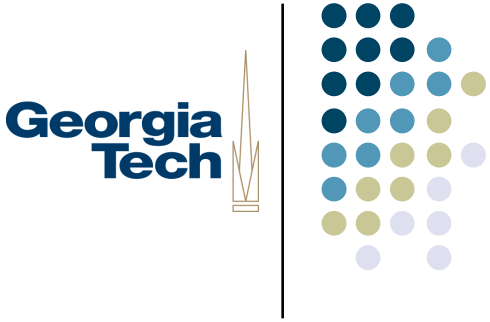
Keith Edwards
keith@cc.gatech.edu



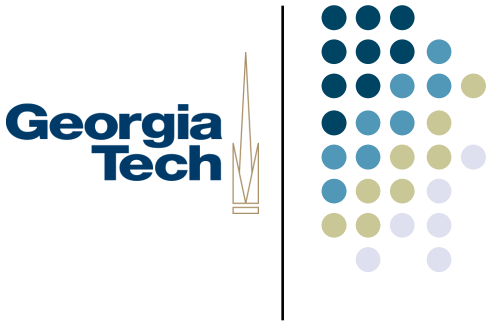
Introductions!

- Name
- What program
- Why this class?

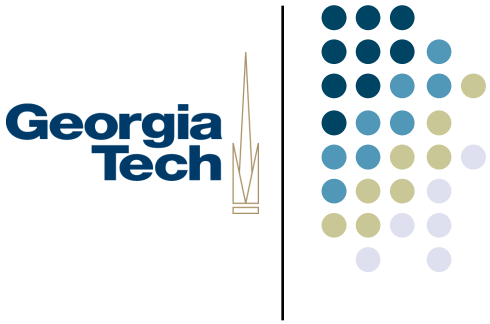
Some Preliminaries



Nuts and Bolts

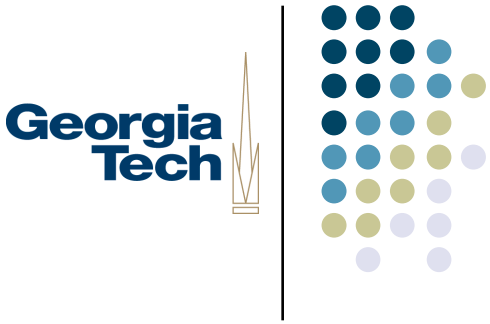


- This is the second required class in the HCC Ph.D. program
  - Designed to ensure a basic level of competency in building medium-scale programs
  - Understanding of software architectural design considerations
  - Best thought of as the second part of CS4452 (which is CS1315++)
  - In HCC terms, should give you the skills needed to do your computation portfolio requirement
    - Technical reading
    - Technical writing
    - Technical doing
    - Technical talking
- Also substantial HCI MS representation



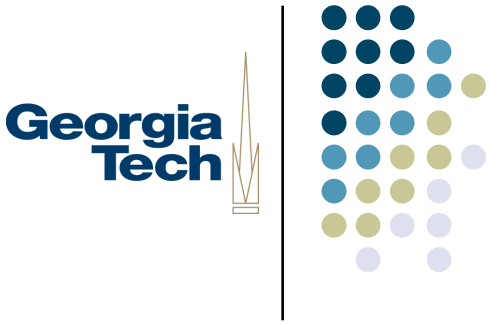
Setting Expectations

- What does "Prototyping Interactive Systems" mean, anyway?
- The course title has caused a lot of confusion:
  - *Not* about using prototyping tools (e.g., Director)
  - *Not* about evaluating prototypes (take the HCI class for this)
  - Instead, about the *rapid creation* of *interactive systems* through *programming*
- Emphasis on scripting languages and common technical idioms that are useful across a breadth of CS
- Covers both *theory* and *practice* of pragmatic systems building...
- ... as well as skills in describing/arguing/defending your design choices



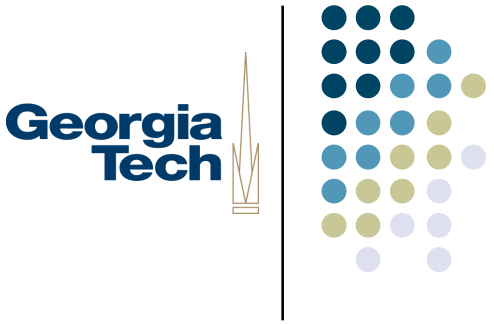
Programming and Prototyping

- What does programming have to do with prototypes?
  - It's the final (and most time consuming) stage of the prototyping lifecycle
  - Gives you the most high-fidelity approximation of a "real" system
  - Useful for communicating with end-users, other developers, etc.
- How is prototype programming different than other programming?
  - Focus on rapid creation of basic functionality, appearance, behavior
  - Less on dealing with errors, boundary conditions, performance, etc.



Focus on Practice

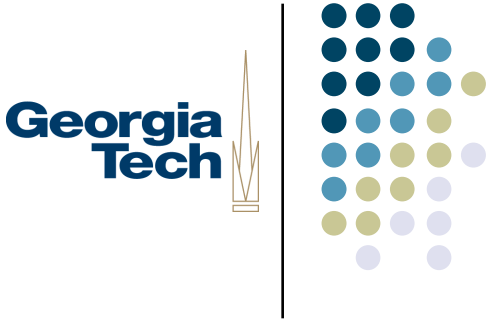
- Software development with a focus on breadth, not depth
- Skills to produce high-fidelity interactive prototypes
- Skills to produce code that *makes an argument*: demonstration of concepts
- HCC: skills to complete the computation portfolio requirement
- Skills in *talking* and *writing* about code
- Pragmatic development:
  - Scripting languages (Jython)
  - Integration with non-scripting languages (Java)
  - Multi-file development
  - Command line tools
  - GUIs, networking, threads, databases, web services, security, ...



What Do We Mean By Theory?

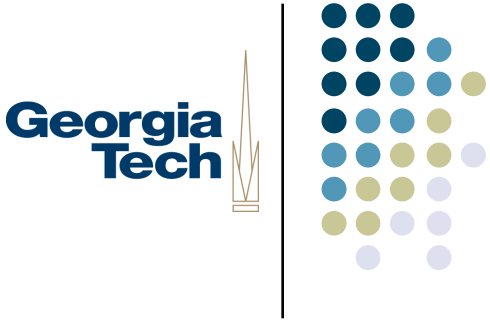
- Understanding why things work the way they do
- Understanding competing architectures and approaches
  - E.g., client-server versus peer-to-peer
  - E.g., different models for GUI programming
- Not just building systems for you to evaluate...
- ... but understanding the design choices embedded in systems, and what those implications are for HCC
- Reading and understanding technical papers for their (often implicit) design choices

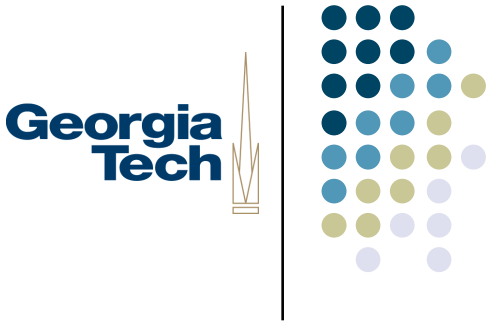
My Goals for this Class: HCC and HCI students



- Hone your programming chops to the point where a medium-sized project (say, 5000 lines of code) is not a terrifying prospect
  - Learn how to decompose a problem into manageable chunks
  - Learn enough of the "idioms" of programming to be able to do more than just simple, straight-line programs
- Impart a few "meta skills" in the process
  - Communicating *about* software
  - Communicating *through* software
  - How to appropriate (read: steal) others' code and adapt it
  - Basic software project management
- Basic understanding of a range of systems architectural choices

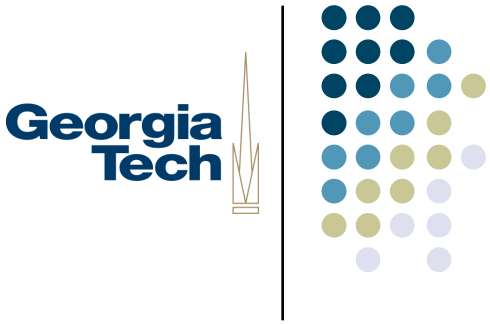
Course Structure





Course Structure

- Course is structured as a set of "modules"
  - Each module covers a subject area in CS
  - Modules align with topics needed to complete a part of the project
  - Readings cover advanced topics related to each module
- Each module is roughly 2-3 weeks, but we'll adapt as needed
- Roughly:
  - First half of class is lecture, mostly focused on practical concepts
  - Second half is either paper discussion, or problem solving/lab
  - Occasional: invited guest lectures on topics of interest
- Everybody works individually, but we'll share experiences
  - Short in-class presentations toward the end of each module
  - Describe the architecture of a portion of your prototype, how you solved a problem, what design choices were available, etc.



Modules

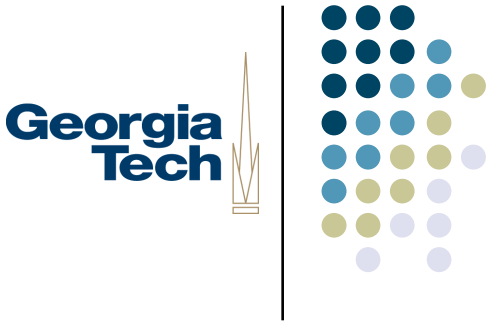
I. Asynchronous Programming
   - Event-based programming, callbacks, polling

II. Distributed Applications
   - Idioms of networking, client-server, peer-to-peer

III. Web Services
   - XML, SOAP, using web services in practice, integration with Java code

IV. Data Management
   - Logging, instrumentation, data storage and querying, databases

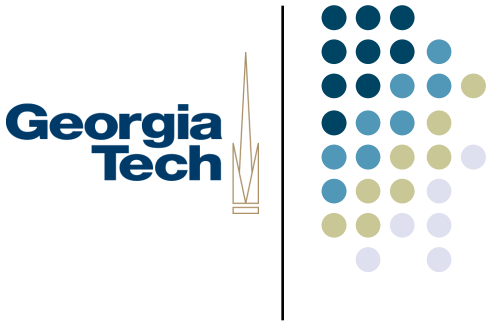
V. Advanced Topics (if time)
   - TBD, but candidates include: security, hardware, research in prototyping



The Project

- This is a project class
- We will do one project that lasts the duration of the semester
  - IM/Chat program, probably 2000-3000 lines of code
  - Single-person "teams"
- Assumes Jython knowledge at about the level of CS4452
  - Good mastery of control flow, variables, scoping
  - Basic object-oriented programming concepts
  - How to use JES (or another development environment)

Readings and Homeworks



- We'll have a number of readings through the semester
- Papers selected to build on topics covered in each module
- Technical papers: UI software, networking, applications, etc.

- Homework: written, one-page summaries of each paper
  - I'll provide a list of criteria I'd like you to touch on in your summaries

- In-class discussion of each paper

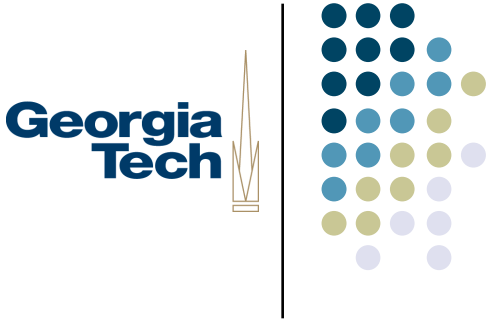
Exams

- Exact number TBD
- Longer written assignments based on either the readings or the project
  - Possible examples:
    - Write an "implementation section" describing the design choices inherent in your project
    - Take three of the assigned papers and contrast/critique the technical assumptions made in each
- Will likely be take-home

Grading Criteria

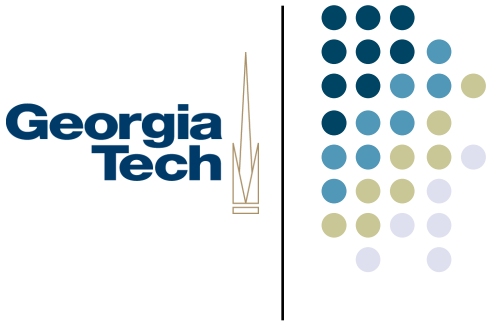
Project Implementation: how functional is your prototype? How well does it work? How well does it demonstrate the concepts taught in class?	40%
Homeworks (paper summaries)	20%
Exams (longer written assignments)	20%
In-class presentations	10%
Class participation and attendance	10%

Today's Class

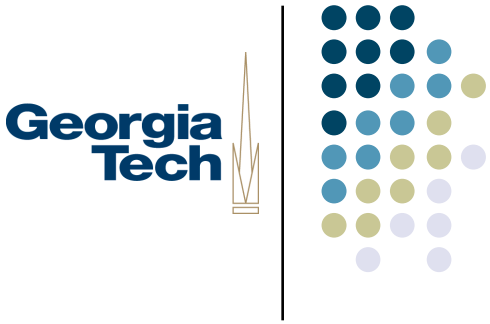


- Outline for the remainder of today's class:
  - What is prototyping?
  - Why prototype?
  - The kinds of prototyping
  - The first project assignment
  - Practicum: getting started

What is Prototyping?



- The creation of *artifacts* that can be used to:
  - Assess the utility and usability of a proposed system, through *evaluation*
  - Communicate design alternatives with various stakeholders
    - The "customer"
    - Engineers/builders
    - Management
- Ideally, a prototype should
  - ... be quick enough to build to allow easy experimentation
  - ... have fidelity *appropriate* to demonstrate the desired concepts



Why Prototype?

- In two words: **risk mitigation**
- From an evaluation perspective, allows you to get feedback on designs before there's a huge investment in it
- From a design perspective, allows you to quickly experiment with alternatives, cheaply

An Example

- When interfaces go bad...

- What's wrong with this?

An Example

- When interfaces go bad...

- What's wrong with this?
  - The "From" field is editable, but doesn't do anything!
  - Let's you change the file extension without warning
  - Is modal!
- Could this have been saved by prototyping?

Another Example

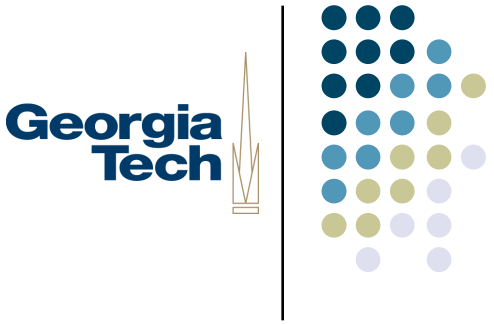
- Not just restricted to applications...

"If you are seated in an exit row and you cannot understand this card or cannot see well enough to follow these instructions, please tell a crew member."

One more...

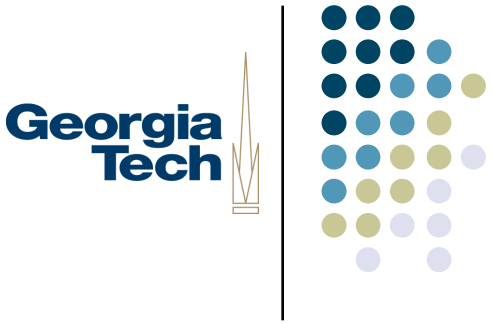
- Alarm Clock, a la Terry Gilliam's *Brazil*

Kinds of Prototypes



- There are a range of prototyping techniques, for a range of goals
- Ideally:
  - Start with lightweight prototypes to communicate the "big picture"
  - Move to more realistic ones as risk factors are mitigated and you need to communicate about the details
- Fidelity in prototyping
  - Fidelity is the level of detail in a prototype
  - Low-fidelity: many details missing, maybe "sketchy" apperance
  - High-fidelity: prototype looks like the final system on the surface

Low-fidelity Prototyping



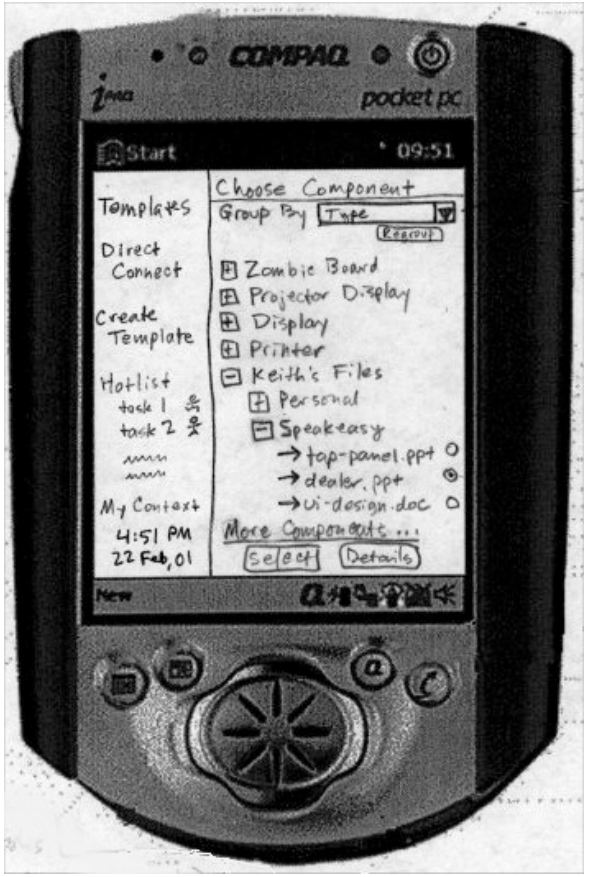
- The lowest of the lo-fi: paper prototyping
  - If you've ever designed a UI, this is probably something you've done informally
  - Capture overall layout
- Storyboards
  - From the film and animation arts
  - Capture *behavior,* not just *appearance*

- Goal: keep the design/implement/evaluate cycle as tight as possible
- These techniques do it by keeping the implementation phase small

Example: Simple Paper Prototype

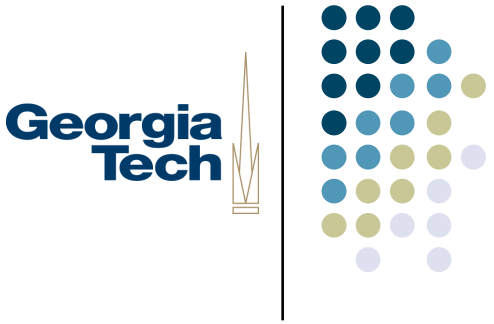
1. Get image of iPaq
2. Cut out screen area
3. Make lots of copies
4. Fill in copies as needed

- Can be turned into storyboard
  - Annotate controls with numbers
  - Numbers lead to other sheets



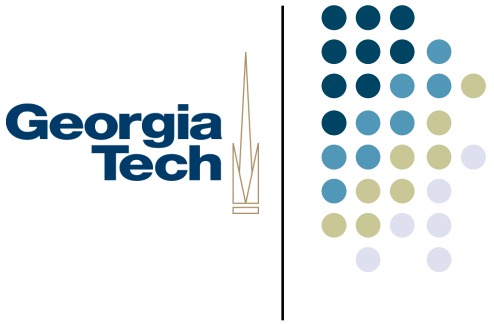
A Few More Examples

Developing and Evaluating Low-fidelity Prototypes



- Basic tools of the trade:
  - Sketch large window areas on paper
  - Put different screen regions (anything that changes) on cards
  - Overlay cards on paper
- The copier is your friend:
  - Can easily produce many design alternatives
- Evaluation: You can "run" your paper prototype
  - The designer "simulates" the computer in front of a user
  - Need to be ready for any user action (drop-down menus, etc.)

High-fidelity Prototyping

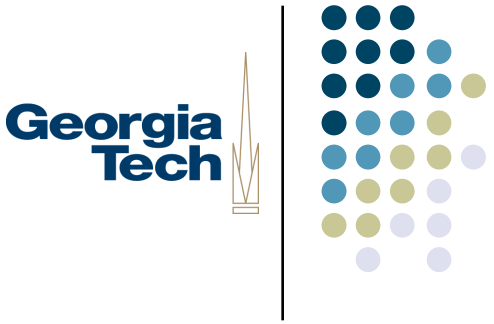


- Once again, a range of practices that give you higher fidelity in exchange for higher implementation time

- Tool-based approaches
- GUI builders
- Code-based approaches

- Downsides:
  - Cost is the obvious one
  - Also:
    - Warp perceptions of the customer: elict more comments on color, fonts, etc.
    - Attending to details can lose the big picture

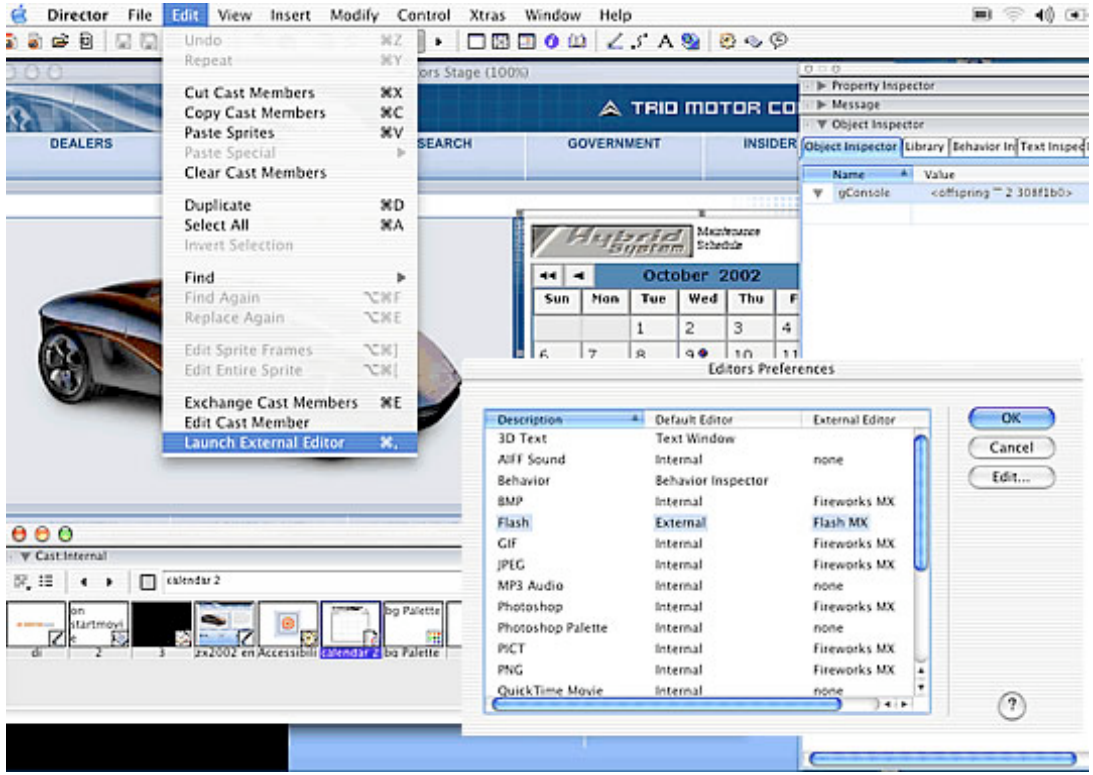
Tool-based Prototyping



- Examples: Director, Flash, the Web

- Pros:
  - Faster than writing code
  - Easier to incorporate changes
  - Often more reliable (hit the back button, rather than program crash)
- Cons:
  - No easy way to transition to a finished product
  - May not allow access to the full range of features available to the finished product (e.g., may not be able to prototype networking, or certain platform-specific features)

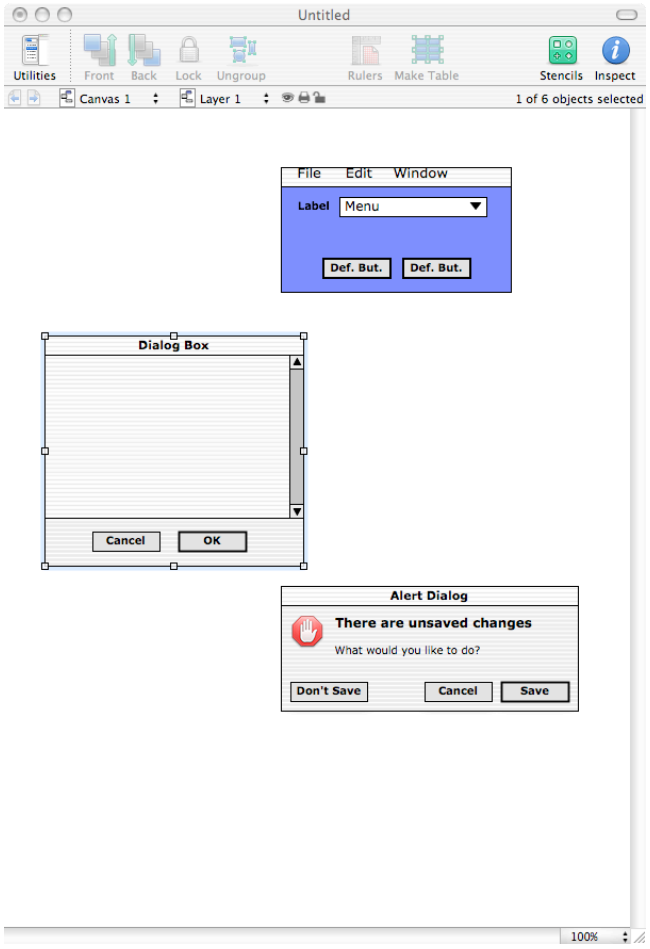
Example: Director



- Timeline editing, palettes of graphical widgets, etc.
- Emits a file that can be executed on any program that has the required runtime engine

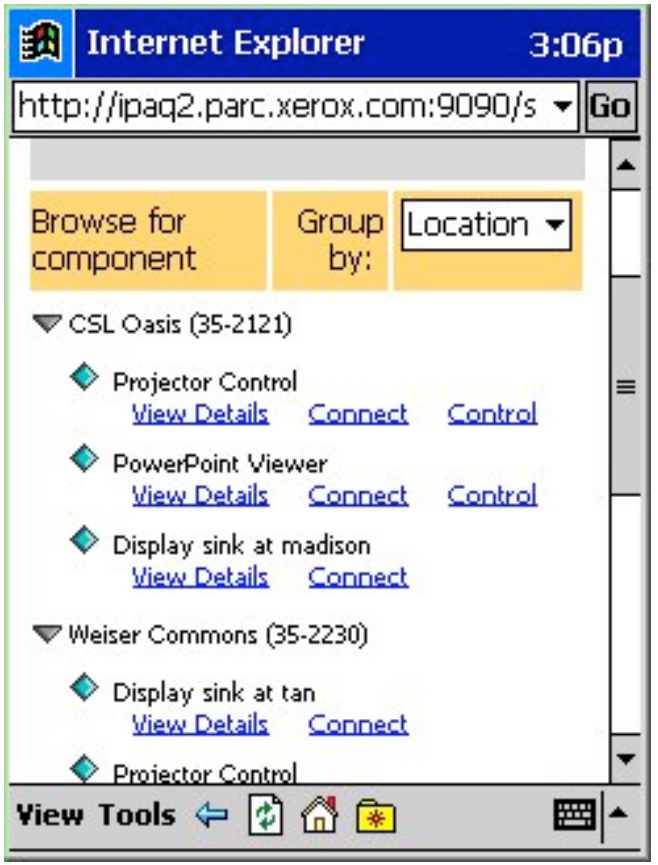
Example: OmniGraffle

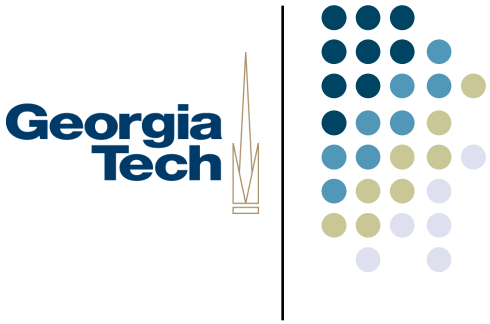
- Drag graphics that depict GUI elements onto canvases
- Canvases can be linked
  - Example: Click on element A on canvas 3 goes to canvas 4
- Can emit an interactive set of web pages
- Mac only, unfortunately



Example: Web Prototyping

- Web-based version of lo-fi prototype shown earlier
- "Controls" simply link to another page
- Allows fine-tuning of text, graphic size, after behavior has been tested on paper
- Can be done by hand or by web development tools



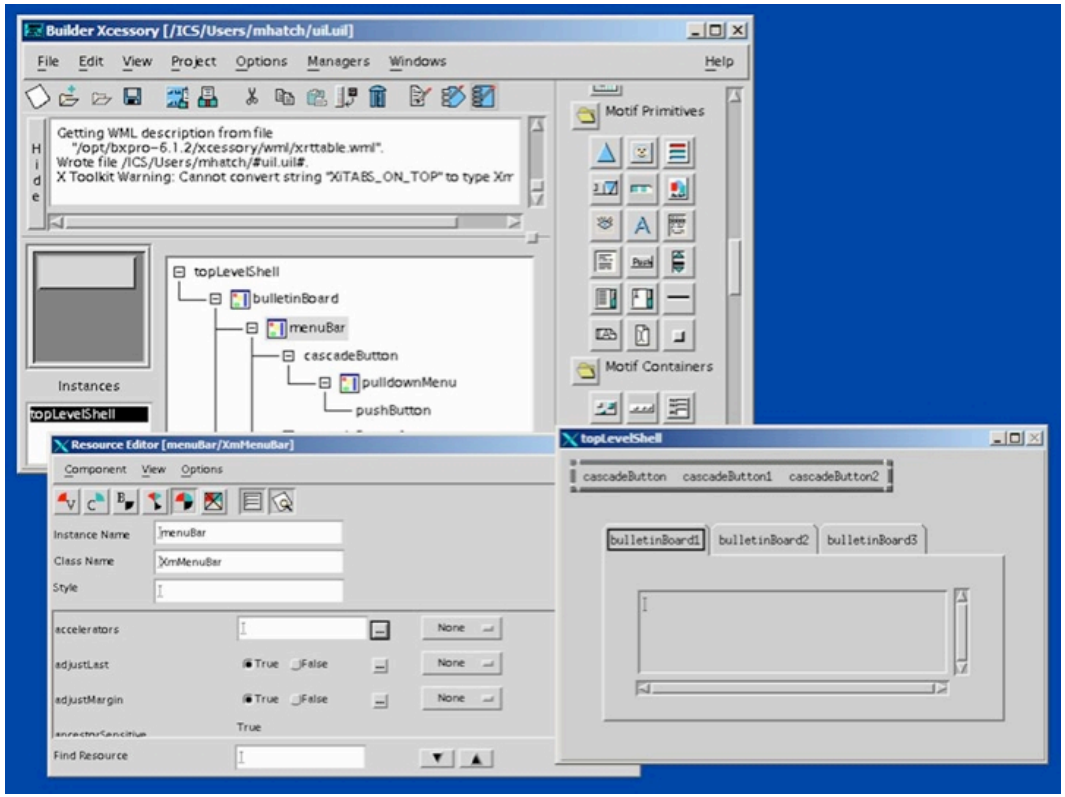


GUI Builders

- A special class of tool for creating GUI systems
  - Drag-and-drop "widgets" from a palette
  - Emit code that you then edit: fill in the blanks

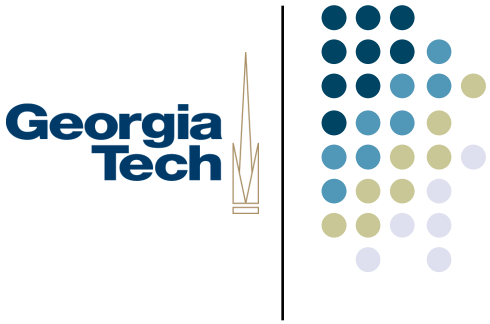
- Pros:
  - Facilitate reasonably good transition to the final product
  - What you get looks exactly like what the finished product will look like
- Cons:
  - Still have to know a lot about programming
  - *AND* have to know about programming peculiarities in the GUI builder itself (can be very opaque)

Example: BX Pro



- Drag and drop graphical "widgets" onto a screen canvas
- Set properties of widgets

Code-based Prototypes



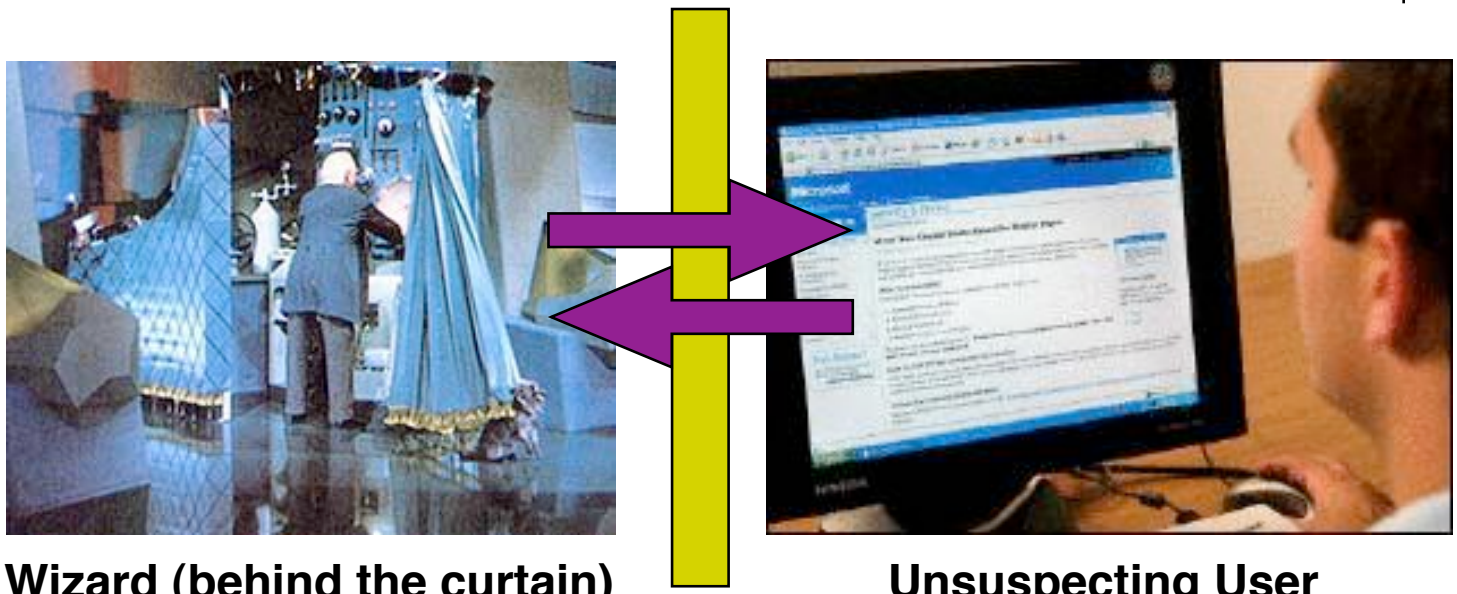
- This is what we'll be focusing on, after this week
- Many approaches:
  - Production languages (Java, C++, etc.)
  - Scripting languages (Jython, Python, Visual Basic, AppleScript, TCL)
- There is often a fuzzy line between code and the use of tools
  - Can often "drop down" to code to augment behavior

- Pros:
  - Very high fidelity
  - True interactivity
  - Good transition to final system
- Cons:
  - Cost, learning curve

Evaluating Hi-Fi Prototypes

- Some hi-fi prototypes are hi-fi-*enough* that standard HCI-style analyses work fine
- But what if you don't have all the necessary behaviors implemented?
- Answer: ***fake it!***
- *Wizard of Oz* technique
  - You are the person "behind the curtain"
  - Provide simulation of missing implementation details as necessary
  - Especially important for features that are hard to implement
    - E.g., speech or handwriting recognition, activity sensing, intelligent interfaces, etc.

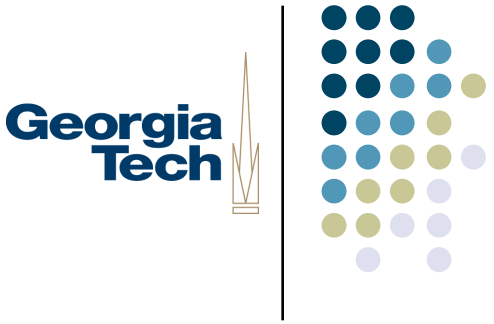
Example: WoZ



Wizard (behind the curtain) **Unsuspecting User**

- Wizard watches human input and explicitly controls the computer

This Week's Assignment

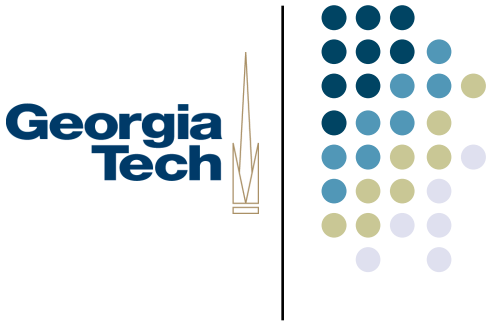


- Create a lo-fi paper or web prototype of the UI for the project
- This prototype will serve as the basis for the *interactive* UI we will create in the first module
- *Prototyping as a **design** tool, not an **evaluation** tool*

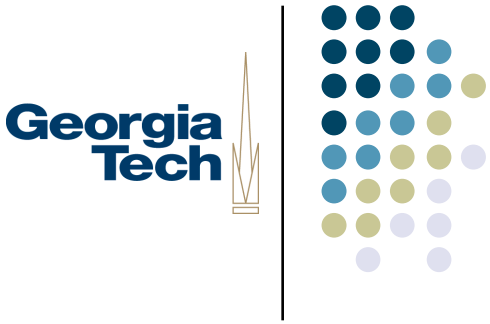
- Requirements:
  - Should show every screen/window that is reachable in the UI
  - Identify all graphical elements
  - Identify transitions between elements
  - Should be sufficiently detailed that you could "run" a user through it, by playing computer

- Submit to me by next Monday

Requirements for IM GUI



- Provide list of all online users
  - Allow selection of one (or optionally, more) users
  - Provide some control to initiate a chat
- Requested users should receive an invitation window
  - Allow them to accept or reject the invitation to chat
- For each chat a user is engaged in, one chat window
  - Text area that shows chat transcript of all parties
  - Area to enter your text
  - Provide some control for disconnection
- Other members of chat should receive notification upon disconnect of another chat member

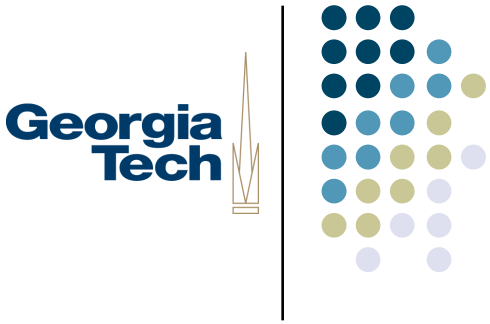


Practicum

Getting set up for development

- Install Java, *if you don't already have it*
  - Macs: comes with OS X
  - Windows, Linux: http://java.sun.com/j2se/1.5.0/download.jsp
    - I'll be using Java 1.5.0; you're welcome to use earlier versions at your own risk
    - Either the full Java Software Development Kit (JDK) or Java Runtime Environment (JRE) should be sufficient
- Downloading Jython
  - http://www.jython.org
  - Jython 2.1or 2.2a1
  - Should run on any platform that supports Java 1.2 or later

Practicum, cont'd



- Development environment
  - I'm agnostic about which (if any) development environment you use
  - JES: simple, single-window environment for Jython
    - http://coweb.cc.gatech.edu/mediaComp-plan/94
  - Eclipse: *much* more complicated, but more "real"
    - http://www.eclipse.org
  - JEdit
    - http://www.jedit.org -- used with some success last time
  - Others:
    - You're more than welcome to use a simple text editor and command-line Jython
- If you're unsure what to use, or new to programming, my suggestion is to use JEdit