Experiences with Just-in-Time Teaching in Systems & Design Courses

Janet Davis
Dept. of Computer Science
Grinnell College
Grinnell, Iowa, USA
My Goals for This Talk

- Share my experiences
- Contribute to a small body of literature on Just-in-Time Teaching in Computer Science
- Advertise!
  - It's fun, it's effective, try it!
  - Can work for new instructors as well as experienced.
Outline

• What is Just-in-Time Teaching?
• Prior work in computer science
• Three examples, two approaches
  • Assignment
  • Lesson planning demonstration
  • Student & instructor evaluations
• Conclusion
What is Just-in-Time Teaching?

Learning in the Classroom → Learning outside the Classroom

Learning outside the Classroom ← Learning in the Classroom
What is Just-in-Time Teaching?
[Novak et al. 1999]

Learning in the Classroom

Homework, Projects

Exercises completed before class based on readings or prior knowledge

WarmUp Exercises
What is Just-in-Time Teaching?
[Novak et al. 1999]

Key Idea:
Create or adjust lesson plans “just in time” in response to students' preliminary understandings.
Applications of JiTT to CS

- Bailey & Forbes (SIGCSE 2005) – CS 0
- Astrachan (FIE 2004) – Programming contests
- Fleischer (TLS 2004) – Theory of computation

What I contribute:
- Experience with different types of courses
- Experience of a new instructor teaching courses for the first time
Approach 1: Discussion Questions

- Students are asked to write
  - Questions concerning terms not defined in the reading,
  - Questions to clarify ideas,
  - Questions about the relationships between ideas or approaches,
  - Questions regarding assumptions behind approaches,
  - Questions concerning the motivation or necessity of ideas or protocols.

- Cite a page number from the text

- Operating Systems, 15 students
- Average one question per class for 10% of grade
- Due 5 p.m. for 8 a.m. class
My Class Preparation Process

- Review assigned reading (1-2 days before)
- Gather questions in HTML document (5 p.m.)
- Organize:
  - By topic
  - Basic → detailed → “big picture,” beyond scope of text
- Sketch lesson plan on paper (later that eve)
- Mainly use whiteboard, occasionally overheads and handouts, never PowerPoint!
Discussion Questions for 9/4

Device management
- Who is usually responsible for writing device drivers? The designers of the OS? Or do they rely on the manufacturers of components to create drivers that are compatible with their reconfigurable device drivers?

Memory management - What is virtual memory?
- My understanding of virtual memory is that it is a simulation of physical memory (93), and I would therefore say that virtual memory is not "real" memory. So, how is it possible to store or save information on virtual space, space that does not even exist in the first place?
- The book says 'by 1990, almost all UNIX implementations had evolved from swapping systems to paging systems.' (p. 106) What exactly is virtual memory, how is it used in programming, and how much control over virtual memory does programmer have? Is virtual memory controlled by the operating system or by the programs that are running?
- If virtual memory utilizes storage space in both Physical memory and Storage device, how does it deal with the I/O delay that Storage device need to retrieve/store data? Would this cause complications in the synchronizing of the memory management system?
- What is the difference between swapping and paging systems, as mentioned on page 106?[we'll come back to this later too]

Processor modes
- I'm a bit confused on the mode bit mentioned around page 96. How can the bit be changed from user to protected mode? A protected program cannot change this bit, as to be protected, the bit must already be set. This means a user space process is changing the mode, but wouldn't that defeat the purpose of the protection? (we'll also come back to this more in chapter 4)
- When discussing system calls, the book refers to "stub" functions that use the trap function. (3.2) Are these stub functions similar to husk and kernel functions where the stub simply calls the trap function with some additional parameters or is there more to the stub functions?
- Why is it that the trap mechanism and related code of kernel-implemented functions take relatively longer time to execute compared to normal function calls? (pg 98-99)
- In supervisor mode the processor only executes instructions from the OS kernel, since that is the only trusted software (at least that's mentioned in the book). How would an app programmer "trick" the processor into thinking the mode bit is set to supervisor, or how would an OS programmer prevent that from ever happening?
- Is it possible for end users to write a trusted software? If not, does it imply the number of trusted softwares has been fixed since the OS is released?
- Is it possible for an application programmer to write programs for the kernel of an OS (i.e. that will be executed in the kernel of an OS)? I know that one of the major advantages of Linux is that the user can alter the kernel and recompile to truly customize the OS. How does this fit in with the trusted aspect of kernel software?

Requesting services from the OS
- The author states that "as a rule of thumb, operating systems based on a system call interface are more efficient than those requiring messages to be exchanged between distinct processes" (pg 100). However, are there special cases when message passing is more efficient than system call?
- A daemon is a user space thread that seems to implement time or space multiplexing that the OS usually handles. Shouldn't the OS simply do what a daemon does? (pg 101) [What is a daemon?]

UNIX Architecture - went over diagram did not get to
- Could you explain what is meant by a portable timesharing OS (104)? The definition of timesharing in the glossary is not very clear, and the concept of portability is not explained.
- Originally, Unix served as the example of a pure microkernel (pg 104), but more modern versions have moved to a larger more complex kernel (pg. 106) due to device complexity. Why did the operating system (how Solaris) change its basic philosophy? Is the microkernel no longer valid? Is the NT kernel considered a "microkernel"?

Windows NT Architecture
Student Evaluations – Approach 1 (OS)

Concerns:
• Frustration about time questions are due, reading to assignment

Kudos:
• “helps me think carefully”
• helps to have a forum to ask and discuss questions
• “helps everyone to contribute”
• “entertaining”
Instructor's Evaluation – Approach 1

Advantages:

- Self-bootstrapping.
- Class prep in brief regular sessions [Boice 2000].
- Students prepare.
- Address questions & misconceptions.
- Use class time effectively.
- Engaging, improvised, conversational, fun!

Disadvantages:

- Prep takes longer than reusing notes.
- Prep not reusable.
- Need a readable text.
- Less structured.
- May need to work to break out of lecture mode.
Approach 2: WarmUp Exercises

- “Pre-homework” - problems to be completed or questions to be answered before class
- Used in
  - Software Design, 14 students, 15% of grade
  - Human-Computer Interaction, 17 students, 20%
- Completed several hours before each class
- Graded on effort, not correctness
Example assignment: Week 1

0. What, in your experience, makes really BAD software?

1. Brain Power Question, p. 35: Can you think of three specific ways that well-designed software is easier to change than software that has duplicate code? [Head First Object-Oriented Design & Analysis]

2. What's the most important thing you learned from this chapter?

3. What's one question you still need to answer?

4. About how long did you spend doing the reading and answering these questions?
Bad Software (vs. Good SW)
- Copy-paste
- Hard to use
- Lacks useful functionality
- Wrong tools (QBasic)
- Doesn't work correctly
- Inflexible
- Not having a plan, rushing
- Too many features
- User-hostile
- Sloppy coding
- Hard to understand

What I learned
- Saw examples of encapsulation
- Three steps:
  - Not just functionality
  - No BOF
  - OO used well allows more flexible, robust code than OO used badly
- Fundamental design that changes don't necessarily change results in code
- Importance of keeping things hidden, keeping code that does different things separate

Questions
- What if you don't care about the type of wood?
- What if they care about the price?
- What's type safety?
CSE 223 2007F
"Well designed apps rock!"
Wed, Sept. 5

1. What makes BAD software?
   - Go around class
   - Fill in gaps

2. Three steps to great software
   - What are they?
   - How do we use them
      - to avoid bad software?
      - what about good software?
      - what about SEF?

3. Encapsulation
   - what did we learn?
   - Keep things hidden
   - Keeping different things separate & related things together
   - explain guitar spec example

Questions:
- What if you didn't care about the type of wood?
- What if they care about the price?

4. Other questions
   - Is there a limit to how far you should go when making things object-oriented?
   - Why don't software developers use these steps?
   - Just write up if nothing else is specified.

- What is type safety?
- Variables & parameters
- What is an interpreter?
Example assignment: Week 2

1. Super Brain Power Question, p. 85: Can you come up with at least one more alternate path for Todd & Gina's dog door? Write out the use case & update the requirements list for your new alternate path, too. [HF-OOA&D]

2. What's the most important thing you learned from this chapter?

3. What's one question you still need to answer?

4. We'll (probably) practice developing use cases and requirements in class. Can you suggest a small application that would be interesting to think about?
CSC 223 2007F
Monday, Sept. 10

0. Admin
   - Lab 1 due by 5:15 pm
   - Office hours after class.

1. Overview
   - Why do we need both:
     - Use cases & requirements?
     - Most important thing

2. Super Brain Power, p. 85

3. The use case
   Role-playing exercise
   - 10 clients,
   - 10 designers (pairs)
   - 4 groups of 3

   Jigsaw: Client discuss
   Designer discuss
   Share lessons w/class
   (at least one)

   Designer: talk to client, write requirements & use cases
   Use computer or index cards
   Scenario taken from
   Student comments
   Make sure complex
   Make enough
   Thumbs up!

4. Discussion
   - What about HW?
   - Malfunctions?
   - How far to go w/alt. paths?
   - What if a client gives a "bad" requirement?
   - Simulator vs. real world
Student Evaluations – Approach 2 (HCI class)

Concerns:
• Disliked readings
• Seem to not always address key issues
• Some took longer than others

Kudos:
• Made me do the readings & think critically about them
• Helped know where to focus effort
Instructor's Evaluation – Approach 2

Disadvantages:
- Need to generate WarmUps.
- Helps to have a text with good exercises.
- More work to decide how to use student responses.
- Need to set student expectations carefully.

Advantages:
- Can reuse WarmUps.
- Class prep in brief regular sessions.
- Give more direction.
- Feed in-class problem solving as well as discussion.
- Engaging, improvised, conversational, fun!
Conclusion

- Try it!
  - It's fun.
  - It's effective.
  - It takes less time than you might think, especially when preparing a new course.

- “They are like nothing I've encountered in any of my other classes, but I think it's a great idea.”

- “I wish every class used them.”
Questions?

Janet Davis
Grinnell College
<davisjan@cs.grinnell.edu>
Discussion Questions for 9/4

Device management

- Who is usually responsible for writing device drivers? The designers of the OS? Or do they rely on the manufacturers of components to create drivers that are compatible with their reconfigurable device driver?

Memory management

- My understanding of virtual memory is that it is a simulation of physical memory (93), and I would therefore say that virtual memory is not "real" memory. So, how is it possible to store or save information on virtual space, space that does not even exist in the first place? (please]
- The book says "by 1990, almost all UNIX implementations had evolved from swapping systems to paging systems."(p. 106) What exactly is virtual memory, how is it used in programming, and how much control over virtual memory does the programmer have? Is virtual memory controlled by the operating system or by the programs that are running? (please]
- If virtual memory utilizes storage space in both primary memory and secondary device, how does it deal with the I/O delay that secondary device need to retrieve/store data? Would this cause complications in the synchronization of the memory management system? [we'll come back to this later when we talk about memory in more detail]
- What is the difference between swapping and paging systems, as mentioned on page 106? [we'll come back to this later too]

Processor modes

- I'm a bit confused on the mode bit mentioned around page 96. How can the bit be changed from user to protected mode? A protected program cannot change this bit, as to be protected, the bit must already be set. This means a user space process is changing the mode, but wouldn't that defeat the purpose of the protection? [we'll also come back to this more in chapter 4]
- When discussing system calls, the book refers to "stub" functions that use the trap function. (3.2) Are these stub functions similar to hash and kernel functions where the stub simply calls the trap function with some additional parameters or is there more to the stub functions? (please]
- Why is it that the trap mechanism and related code of kernel-implemented functions take relatively longer time to execute compared to normal function calls? (pg 98-99)
- In supervisor mode the processor only executes instructions from the OS kernel, since that is the only trusted software (at least that's mentioned in the book). How would an application programmer "trick" the processor into thinking the mode bit is set to supervisor, or how would an OS programmer prevent that from happening?
- Is it possible for end users to write a trusted softwares? If not, does it imply the number of trusted softwares has been fixed since the OS is released?
- Is it possible for an application programmer to write programs for the kernel of an OS (i.e. that will be executed in the kernel of an OS)?
- I know that one of the major advantages of Linux is that the user can alter the kernel and recompile to truly customize the OS. How does this fit in with the trusted aspect of kernel software?

Requesting services from the OS

- The author states "as a rule of thumb, operating systems based on a system call interface are more efficient than those requiring messages to be exchanged between distinct processes" (pg 100). However, are there special cases when message passing is more efficient than system call?
- A daemon is a user space thread that seems to implement time or space multiplexing that the OS usually handles. Shouldn't the OS simply do what a daemon does? (pg 101) [What is a daemon?]
6.4 The Process Abstraction (from last time)

- What's in a process descriptor? (Ms. Davis)
- What is the advantage of using handles? (pg 213)
- At the very end of the main text of 6.4, on page 214, the author mentions that "various other parts of the OS" can change process descriptors, even though a better programming practice would be to ensure only the process manager can change them. What are these 'various other parts' and why do they need to be able to change process descriptors?

6.5 The Thread Abstraction

- When something is implemented by "the library" (top of pg. 218) what does that mean? What is "the library"?
- Java has one class and one interface: Thread and Runnable. Does the "Thread" class in Java have the same concept as our textbook? How about more details?
- I'm not sure I understand the relationship between fork, sys_fork, do_fork, clone, sys_clone, and do_clone [in Linux]. Is clone used for threads rather than processes? What's the difference between sys_ and do_? (p. 218)
  - According to man clone: "Unlike fork(2), these calls allow the child process to share parts of its execution context with the calling process, such as the memory space, the table of file descriptors, and the table of signal handlers. The main use of clone(void) is to implement threads: multiple threads of control in a program that run concurrently in a shared memory space."

6.6 State Diagrams

- In UNIX state diagram, the "zombie" child is only terminated when the parent calls wait(). What happen if the parent is terminated before the child or the parent never call wait()? Will this cause resource leakage by keeping the process descriptor of zombie processes? Does this suggest that a child should not stay alive longer than the parent? (p. 221)

6.7 Resource Managers

- Why is a resource defined as "anything that can potentially block a process from executing" (p. 225) when a more useful definition could be used (such as something to do with data transmission)?
- The book mentions, on page 224, that a computer has RResources and CResources and each resource requires an example of what an example of a process would need more than one CD drive at the same time or any other resource for that matter? (Lu) Li
- On pg.223, the author mentions that there are various "types" of object of resource and each type has multiple "units" of resources. Could explain with examples what he means by "types" and "units" of resources.
- Just out of curiosity, what are few other consumable resources apart from input data and messages that the author mentions? (Saugar)
- If a USB device is unplugged, how does its resource manager let processes know that the resource is no longer available? (Or more generally, can a resource be taken away once it is allocated? (Luis)

6.8 Process Management Policies

- From page 226, it seems that a parent process knows all its child processes, and that a child knows its parent. However, does a child process know its "siblings"? What type of sharing occurs between child processes, explicit or transparent? (Saugar)
- Why do some OS's ignore the hierarchical relationship described in Figure 6.13? I thought that the parent process automatically had the right to block/activate the child process. How would the parent manage the child otherwise? (Luis)
- What is the advantage of giving a parent control over child processes?
- Are there any advantages or disadvantages to using the process hierarchy to control aspects of processes instead of leaving it all to separate resource managers? Why would programmers choose one design over the other?
- What are the advantages/disadvantages of having a parent process control its child processes for an app programmer or end user?
- The OS for the RC 4000 (pg. 228) sounds very interesting. What makes this setup so difficult and why isn't it in use today?

Janet Davis (davisj@cog.gmu.edu)

Created September 14, 2006
Last revised September 14, 2006
Example assignment: Week 10

1. Which XP practices are already being used by your team? In what ways?

2. Which practice would you most like to see your team start using (or significantly improve)? Why?

3. Which practice seems the least relevant to your project for this class? Why?

4. What's the most important thing you learned from this reading?

5. What's one question you still need to answer?
XP Primary Practices

Sit together (7)
Whole team (6)
Informative workspace (1)
* Is this practice relevant? Can we do it?

Energized work (2)
* Is this relevant when programming is not your full-time job?

Pair programming (9)
* The section on Pair Programming notes that "most programmers spend more than five or six hours in a day". With this in mind, how do companies that use pair programming have their employees when the remainder of the workday? Two to three hours of need for more than five or six hours in a day. As does two to three hours of incredibly excessive code. How many programmers actually write code?

Stories (1)
* What is this practice?

Weekly cycle (2)
* Do we have a weekly cycle?
* On page 66, in the second item under "Weekly Cycle": "Have the customers track a week’s worth of stories or to do list. Who are the customers in this case? Are the customers and developers?

Quarterly cycle
* This is difficult to do in a one-semester course!

Slack (2)
Ten-minute build
* What is it?
* Is it relevant to this practice?

Continuous integration (3)
Test-first programming (2)
* Is this appropriate for our projects?

Incremental design (4)
* What is the heart of this practice?

---

Wed. Nov 14

0. Admin

- Project Workshop
  Friday

- Next week:
  More XP or learn SQL?

1. Practice Q&A

2. Teams & Practices
   - Handed in your emails
   - Identify one or two practices your team would like to improve
   - Map that practice

3. Share
   - Share

3L: Share
Example Student Questions

My understanding of virtual memory is that it is a simulation of physical memory (93), and I would therefore say that virtual memory is not "real" memory. So, how is it possible to store or save information on virtual space, space that does not even exist in the first place?

I'm a bit confused on the mode bit mentioned around page 96. How can the bit be changed from user to protected mode? A protected program cannot change this bit, as to be protected, the bit must already be set. This means a user space process is changing the mode, but wouldn't that defeat the purpose of the protection?

I know that one of the major advantages of Linux is that the user can alter the kernel and recompile to truly customize the OS. How does this fit in with the trusted aspect of kernel software?

A daemon is a user space thread that seems to implement time or space multiplexing that the OS usually handles. Shouldn't the OS simply do what a daemon does?