CS373S Software Design
Spring 2015

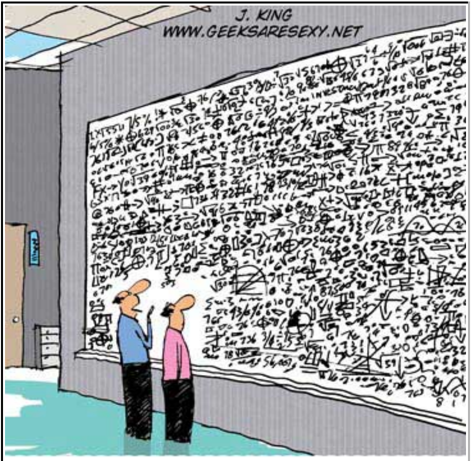
Announcements

H2: Models and their Instances
due Tues Feb 3, 9pm

Program P2 Due Friday, 9pm

Tolouse Latrec Info

piazza Discussion Group



"...And that, in simple terms, is what's wrong with your software design."

Professor	Don Batory	batory@cs.utexas.edu
	office hours	GDC 5.826 Tues 3:30-4:30
TA	Steve Rutherford	ruthst@utexas.edu
	office hours	Monday, 330 - 530.
Room	GDC 1.304	
Days and Time	Tues & Thurs 9:30am-11:00am	
Unique #	52005	
Ground Rules	UTCS Rules to Live By	
Your Grades	canvas LOG IN	
Final	Monday, May 18, 2-5 pm	

Prerequisites, Overview, and Goals

Prerequisites: Minimum C- grade in CS 429 or 429H.

The UT course catalog says CS 373 (software engineering), CS 347 (databases), and CS 375 (compilers) are required for this course --- this is ***wrong****. It would be great if you took these courses concurrently or later, but software engineering, databases, and compilers are NOT required.*

Check this page of student comments on prerequisites.

Bottom line: You must know how to program in Java; if Java is new to you, do not take this course! The more experience you have in programming, the more you will learn and appreciate this course.

Here is a letter that you can print and take to your advisor or CS undergrad councilor to verify the above for registration.

Overview. Software design and construction is not hacking; it is governed by fundamental concepts. We see these concepts at work in **Model Driven Engineering (MDE)**, which gives us a general way to think about software design and construction -- not as a mass of spaghetti code, but as a principled collection of ideas architected in a thoughtful way. MDE deals with the creation of models and transformations of these models to other models, eventually yielding executables (yet other models).

This is rather abstract, but if you think about it, we express a program in the Java language (that's our concrete model of this program). The **javac** compiler transforms source code to bytecode (it maps a Java representation of a program to a corresponding representation of that program in bytecode). **javadoc**, as another example, maps Java source to html (both are models). Object oriented refactorings map source code to refactored source code. There are even refactorings that map bytecodes to bytecodes.

These are just a few examples of program representations and the tools that transform them from one representation to another. Virtually all of software design deals with the creation of models (program representations) and their transformation into other representations. Most work in Engineering (with a capital "E") deals with model development -- ex. Boeing creates and analyzes models of aircraft long before they actually build a physical plane. Similarly, the software next generation Space Shuttle is being built solely from models; no one is writing code! (These models are transformed to ugly C++ by commercial tools and then commercial C++ compilers transform source into executables).

Goals. The goals of this course are to give students the basics for what we as programmers and system architects do, and how to reason about program design and construction. This course will:

- teach you basic skills in reasoning about and expressing software designs,
- give you experience in writing programs to exercise these skills,
- expose you to the fundamental ideas of software design,
- provide an insight on where software design technology is headed, and
- give you a framework to understand what Software Design will become in the future.

The course includes lectures on the following topics: UML models and metamodels, metamodel constraints, categories, refactorings, object oriented design patterns, architectural patterns, parallel architectures, and service oriented architectures (SOA). And of course, concepts from MDE which all of the above are special cases. The course covers examples from compilers and databases , so if you have any experience writing compilers or databases, you're that much further ahead than others. A course on compilers or databases or software engineering is **not** a prerequisite, but it is good to have before you complete your undergraduate education.

As this is gateway course to further undergraduate courses in CS on Software Development, there will be many small programming assignments and written homework assignments.

Course Materials

The PPTX file and its PDF for each lecture will be posted after class. (I change the notes every time I teach the course). Links to these files are listed in the syllabus below. PDF with audio descriptions of figures are given names ending with "Audio".

Should you choose to create a hard copy of these files, **do not use CS printers** -- take the files to Kinkos (or whatever Kinkos is called now). I have created several instructional web pages with accompanying .AVI or .MOV videos. Note: The .AVI videos work only in a Windows-based environment. The .MOV were converted from .AVI files and (unfortunately) are not as sharp.

- NetBeans Instructional Page
- Information on PDF Generation, which you'll need for submitting homework assignments
  - Windows info: http://www.howtogeek.com/howto/windows-vista/how-to-print-to-a-pdf-file-on-windows-vista/
  - More Windows info: http://www.dopdf.com/

Lecture material is taken from the following texts, which are **NOT** required for this course. All of them are worth having in a library, but often, much of their content is posted for free on the web.

- Booch, Rumbaugh, Jacobson. "The Unified Modeling Language User Guide", 2nd Edition.
- Fowler, "Refactoring: Improving the Design of Existing Code"
- Kerievsky, "Refactoring to Patterns"
- Gamma, Helm, Johnson, Vlissides. "Design Patterns: Elements of Reusable Object-Oriented Software"
- Buschmann, et al. "Pattern-Oriented Software Architecture: A System of Patterns"
- Pierce. "Basic Category Theory for Computer Scientists"

A useful list of web pages are collected below:

- SourceMaking.com -- for design patterns, anti-patterns, refactorings, and UML
- YUML -- this is a neat, free, web-based tool for drawing UML diagrams.

Software

All programming assignments (except one) are in Java. We will be using the following software, all of which is free to UTCS students, and all of which has been installed in the UTCS Microlab. The following is for Windows Platforms. If you use Apple machines, well, I will try to help as much as I can, but no promises!

- NetBeans EE IDE 8+ -- I prefer NetBeans to Eclipse for program development as it is a simpler environment. Click here to see the option to download.
- Microsoft Visio 2010/2013 -- for drawing UML diagrams. If you are a CS student, you can gain access to Visio via Software Downloads, read the MSDN/AA agreement, click the secure download site link, and follow the instructions. For ECE students, do the same at the ECE-MSDN/AA web page.
- VLC Media Player -- to view .MOV videos shown in class. (You can use Windows Media Player, but its video resolution is awful). No need for this software if you are a Mac user.
- SWI Prolog -- for writing metamodel constraints.

CS Accounts

If you need an account, follow this link to create a CS account and to find the names of public machines to which you can log in.

Quizzes

There may be unannounced in-class quizzes on course lecture material, readings from the course texts, and/or homework. Missed quizzes will be given a grade of zero unless there are extenuating circumstances.
.

Homework, Programming Assignments, and Submissions

Submissions are via Canvas. **A PDF file must be submitted on all assignments** -- it lists your name and email address -- the address must be hyperlinked so that I can easily send my comments of your assignment back to you. Click here for an example of what is expected.

Issues with Netbeans and Eclipse. Programming assignments will require the writing of clear documentation and Unit tests. There are some problems/issues in writing javadoc documentation and running regression tests with JUnit, which are described below:

- Images in JavaDoc-Generated Files in NetBeans
- JUnit Regression Tests with System.out

You should read these postings carefully.

Examinations

There are two midterms and a final. These exams must be taken on the specified date and at the specified time. If you miss an exam due to extenuating circumstances, a grade will be negotiated for the exam based on a percentage of your homework, quizzes, and other exam scores. Otherwise missed exams will be given a grade of zero.

Class Grades

Final grades will be determined approximately by the following scheme:

1. Your accumulative programming assignment grade will determine the maximum final grade for the course. Ex: if you get a "B" average across all of your projects, your final grade will be no greater than a "B".
2. Final counts 40%, each midterm 30%, approximately.

Homework grades and class participation will be used to decide final grades in grade-borderline cases.

Extenuating Circumstances

If you have difficulty meeting the requirements of this course, fail to hand in an assignment, or miss an exam because of an unforeseen situation, please advise the instructor in writing at the earliest possible date so that your situation can be discussed. If you encounter an unexpected medical or family emergency or a random act of Nature that causes you to miss the due date for homework or miss a quiz or exam, you must present suitable documentation in writing to the instructor before special consideration will be given. A file of all written correspondence will be kept by the instructor and decisions regarding them will be made at the end of the semester.

Schedule

Numbers in [brackets] indicates the estimated number of lectures per topic. The number indicated is a lower-bound, as there will be class room discussions to work on problems and review of homework assignments. PPTX and PDF copies of the lectures are hyperlinked below. Given this, the exact dates of a lecture are unknown. The order in which topics are presented is below.

Topic [# of lectures + days of discussion]	Written Assignments	Programming Assigments
1. Introduction [1] • design, model driven engineering, models, transformations, automated software development	Complete Course Survey (1/23,5pm)	P1: Java Reflection (1/27,9pm)
2. Unified Modeling Language [4] • class diagrams, cardinalities, associations object diagrams, constraints, meta-models and meta-modeling, writing constraints in Prolog, in class examples	H1: UML Models (1/28,5pm) H2: Models and their Instances due Tues Feb 3, 9pm H3: Model Constraints due	P2: Writing Metamodel Constraints in Prolog due Fri, Fed 6th, 9pm P3: Class Diagram Visualization and Model-to-Text Transformations due Wed, Feb 11th, 9pm H3: Model Constraints due
3. Categories and Fundamental Modeling Abstractions [2] • domains, instances, arrows and model transformations, arrow composition, • recursion, cones of instances, Meta Object Facility (MOF) architecture	H4: Meta Grammars due	P4: Programming Graphs due
4. Object Oriented Refactorings [2] • rename, extract method, substitute, move member, factory method, pull-up, • push-down, singleton, extract class (normalization, partition), extract superclass, extract interface	H5: Refactorings due	
Midterm #1		
5. Design Patterns [7] • basic patterns: facade, adapter, flyweight • separation of concerns, observer, model-view-controller, mediator • frameworks, template, factory, abstract factory • grammar patterns, composite, interpreter, decorator [2] • delegate patterns: visitors, command, memento	H6: Refactoring Scripts due	P6: HP Calculator Interpreter due P7: Decorator HP Calculator due P8: Creating Visitors
Midterm #2		

6. Architectual Patterns [2] • layers, multi-tiered and 3-tiered architectures, object-oriented virtual machines, symmetric vs. asymmetric layers • pipe and filter architectures, components and connectors, Java pipes		P9: Gamma Joins due
7. Parallel Architectures [2] • program derivation, refinements, optimizations, map reduce, substitution principles, architectural optimization, architecture refinement and correctness, parallel architectures, parallel hash join architectures • design of byzantine fault tolerant servers, program extensions		
8. Service Oriented Architectures (time permitting) [2] • historical development (CORBA, COM) and evolution to present-day concepts • JAX web servers		
Course Recap [1]		