SOFTWARE ENGINEERING

ENGINEERING | PATTERNS | DEBUGGING

FULL SAIL UNIVERSITY®

In the Software Engineering course, students will learn commonly used design patterns, practices, and principles involved in the process of constructing software. Students will be working inside a prebuilt software solution wherein they will find and fix various kinds of software bugs, add new features to the software, and track the changes made by using version-control tools.

SOFTWARE ENGINEERING

GDD258

THIS COURSE WILL TIE TOGETHER CONCEPTS LEARNED IN EARLIER COURSES AND WILL BEGIN PREPARING STUDENTS FOR GROUP WORK.

In addition to using and expanding on knowledge in such areas as polymorphism, encapsulation, trees, and vectors, students will learn to evaluate the implications of their decisions on their projects as well as their teams.

COURSE OUTCOMES

- To apply key object-oriented programming concepts to code more flexible and robust software
- To use revision-control software
- To describe design patterns and their appropriateness in different situations
- To identify the utility of library modules and their relationship with software applications

COURSE MATERIALS

- *Microsoft Visual Studio*
- *O'Reilly Online Learning*
- Git
- *Unity 3D*

SOFTWARE ENGINEERING

SKILLS DEVELOPMENT

ACADEMICS

Students will utilize and expand knowledge gained in previous courses, including object-oriented concepts such as polymorphism and encapsulation as well as data-structures concepts such as trees and vectors. While forming connections with those building blocks, students will be expected to evaluate the implications of the decisions they make, including their impact on future work for themselves and their teammates.

CAREER

The ability to create a solid foundation for the programs students develop will determine the amount of time and energy they will spend inserting features or fixing problems that occur during the development cycle. Employers depend upon the ability to make effective decisions in this area in order to minimize the costs of making changes to a program.

PORTFOLIO

Students will prepare for their portfolio work by learning how pieces of software are engineered. They will also learn how multiple individual modules merge together once completed to form the software solution.

Note: Your portfolio-related course work will be indicated on this syllabus by the icon above.

ALEJANDRO GARCIA-TUNON

SENIOR SOFTWARE ENGINEER, MAGIC LEAP

Alejandro Garcia-Tunon is a 2005 Full Sail Game Design and Development graduate who has been working in the game industry for many years on titles such as ***Marvel: Ultimate Alliance***, ***Spec Ops: The Line***, and ***BioShock 2***. He has worked for multiple companies including ***Dynamic Animation Systems***, ***Vicarious Visions***, and ***Darkside Studios***. At present, Alejandro holds the position of senior software engineer at ***Magic Leap***.

Alejandro was the multiplayer lead on ***Spec Ops: The Line*** and was lead programmer for ***Avatar: The Last Airbender*** for ***iPhone***, ***Borderlands: Claptrap's New Robot Revolution DLC***, and more.

Alejandro can often be found visiting campus to speak with students and faculty about their work.

SEE HIS WORK

SOFTWARE ENGINEERING

WEEKLY BREAKDOWN

WEEK ONE

The main goal this week is to get students set up and familiar with the appropriate software that will be used throughout the month. In addition, students will start working within a preexisting software solution and begin to make changes to improve the solution in different ways.

THEMES

- Revision Control
- Object-Oriented Principles
- Linking

WEEK TWO

This week, students will focus on solving more difficult problems while continuing to practice skills learned in Week 1.

THEMES

- Design Patterns
- Testing
- Debugging

WEEK THREE

This week, students will work in a different software solution and reinforce and expand upon the principles learned previously.

THEMES

- Interfaces and APIs
- ***Windows Forms***

WEEK FOUR

In the final week, students will merge different components they have worked on to form a larger solution. During this process, they will learn about software conflict resolution and practice the testing and debugging skills they have acquired to solve any issues that arise.

THEMES

- Advanced Revision Control
- Refactoring

SOFTWARE ENGINEERING

GRADE WEIGHTS

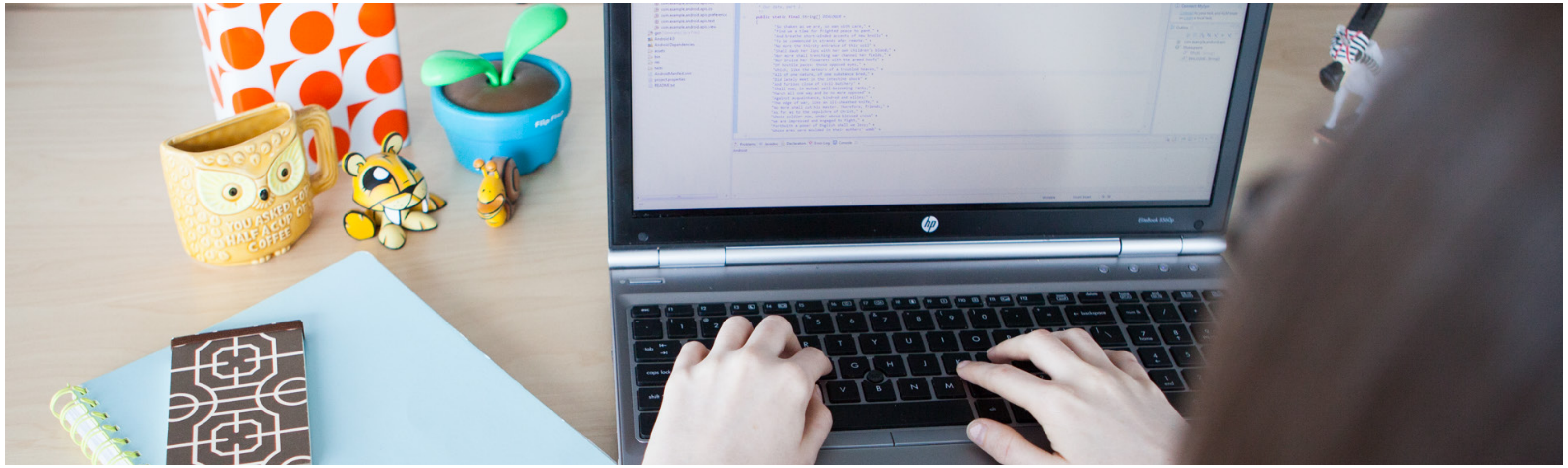
Discussion Board—Source Control (2%), Lab 1—Factory Method (7%) 💼, Lab 2—Command Pattern (7%) 💼, Week 1 Test (12%) 💼	**28%**
Lab 4—Libraries (7%) 💼, Lab 5—Debugging (7%) 💼, Week 2 Test (12%)	**26%**
Labs 6 and 7—Level Editor Project (14%) 💼	**14%**
Lab 8—Merging (15%) 💼, Discussion Board—Research (3%), Week 4 Test (4%)	**22%**
GPS	**10%**

Week 1 Week 2 Week 3 Week 4

Total **100%**

OUT-OF-CLASS WORK

This course requires at least 24 hours of preparation and out-of-class work. Out-of-class activities are documented in this syllabus and include reading assignments, outside research, project development, skills practice, and homework. Consideration has been given to creating out-of-class work that will support your efforts to successfully complete this course while achieving course objectives and program learning outcomes.



RECOMMENDED RESEARCH

EXTRAS

FURTHER RESEARCH | TUTORIALS | COURSE DIRECTOR'S PICKS

WEEK 1

Pro Git, Scott Chacon and Ben Straub, Apress http://git-scm.com/book

Artima Developer: "Pure Virtual Function Called": An Explanation
http://www.artima.com/cppsource/pure_virtual.html

WEEK 2

Game Programming Patterns, Robert Nystrom, Genever Benning
http://www.gameprogrammingpatterns.com

Alex Blekhman: How to Export C++ Classes from a DLL
http://www.codeproject.com/Articles/28969/HowTo-Export-C-classes-from-a-DLL

Michael Haney: Design Patterns in Game Programming
http://www.gamasutra.com/blogs/MichaelHaney/20110920/90250/Design_Patterns_in_Game_Programming.php

WEEK 3

Microsoft Ignite: Build .NET applications with C# https://docs.microsoft.com/en-us/learn/paths/build-dotnet-applications-csharp/

Fundamentals of Computer Programming with C#, Svetlin Nakov, Svetlin Nakov
Chapter 20: Object-Oriented Programming Principles (OOP)
https://introprogramming.info/english-intro-csharp-book/read-online/chapter-20-object-oriented-programming-principles

WEEK 4

Refactoring.com: https://refactoring.com/
Improving the design of existing code

SOFTWARE ENGINEERING

CONNECTIONS

INDUSTRY CONNECTIONS

In this course, students will learn more about core object-oriented principles and practices used every day in the position of a software engineer. This includes making changes to a software solution and keeping track of those changes with revision-control software as well as being able to communicate with teammates about the choices involved in making those changes.

SEMESTER TRENDS

This course builds upon the skills learned in all of the other courses this semester, and the knowledge gained in this course will provide the foundation to enable students to take on more complex course work later in the semester and program. A solid understanding of revision-control software and the knowledge of how to create and integrate software modules are essential for a student to be an effective group member in future classes.

COLLABORATION

Sections of this course combine students from several different degree programs, and while the lab work is not group based, group discussion is encouraged in lecture. Pertinent group exercises will help to facilitate this discussion.

FEEDBACK

A large portion of the lab work is graded in two phases. The first phase will be based on how much students complete within the given deadline, and the second phase will involve polishing the completed work and integrating it into a larger solution. This process simulates exactly what professionals do in the industry.

PERSONAL BRAND

This course covers several different aspects of software engineering, and one of those topics may lead to a specialization within the desired branch of software development. The projects students will work on this month are preexisting solutions created by someone else and are not appropriate for demo reels, but the skills they will learn are ideal for inclusion in their résumés.

COURSE-SPECIFIC RUBRICS

The lab document that will be provided each day will contain a rubric specifically tailored to that day's assignment.

TERMINOLOGY

ABSTRACTION The process of separating out a section of code into a reusable function or class—also an abstract class or interface.

CALLBACK A reference to a function that is stored for future use. It is often passed forward as a function parameter so that something down the line can call it (backward).

COHESION How well a group of things work together. High cohesion is desired in software engineering.

COUPLING How reliant two things are upon one another. Low or loose coupling is desired in software engineering.

ENCAPSULATION Hiding implementation details to reduce dependencies from outside sources and increase robustness.

EXECUTION POINTER In software debugging, the execution pointer points to the line of code instructions that the CPU will run next. In ***Visual Studio***, this pointer is represented graphically with a yellow arrow and will usually point to a line within a code file.

FLEXIBILITY When one thing can be used in different ways or for different purposes.

HIERARCHY Reusing existing functionality by defining a relationship between two classes, typically through inheritance or containment. This also describes the assignment or placement of responsibility.

LAW OF DEMETER The principle of least knowledge; a given object should assume as little as possible about the structure or properties of anything else.

MODULARITY Small pieces of a software solution that are self-sufficient add to its modularity.

PIMPL IDIOM Separation of implementation details from an exposed class to provide additional flexibility and robustness.

POLYMORPHISM Utilizing different objects that exhibit a similar set of behaviors via a shared interface (usually via a virtual-function table in C++); treating an object at the appropriate level of inheritance for a given set of operations.

REVISION CONTROL Keeping track of changes to a file or set of files over time so they can potentially be reviewed or reverted in the future.

ROBUSTNESS Designing software in such a way to facilitate future changes.

TRANSLATION UNIT A preprocessed unit of code that is ready for the compiler to translate or the output of the preprocessor after operating on a single source file.