# CS 362 Course Intro

- In most of your CS courses, you learn about how to *program***,** sometimes in a very general way (like in CS 162 or CS 261) and sometimes in a very specific way (like in CS 290 or CS 492).
    - Some courses also teach CS theory, which is important, too!

- Programming is obviously important in software development. If you don't know how to program, then you can't create software.

- However, because we focus so much on programming in the CS curriculum, it can be tempting to think that to make good software, you just have to be a good programmer.

- In fact, this is not true. Creating good, reliable, production-quality software requires more than just 1337 programming skills for the following, simple reason:

  ***Programmers, no matter how good, are human, and humans are fallible.***

- In other words, no matter how skilled of a programmer you are, you will always make mistakes, and these mistakes will come in many forms, including:
    - Writing code that contains bugs.
    - Mistyping commands, e.g. for building our software.
    - Uploading the wrong version of a file to a distribution server.
    - Clobbering a feature one of your teammates implemented.
    - Etc., etc., etc.

- **In some ways, you could say that CS 362 is about simply accepting the fact that human developers make mistakes.** This is incredibly important because once we accept that we, as developers, make mistakes, we can start to put in place safeguards against our mistakes.

- **Safeguarding against mistakes becomes even more important when we're developing code that users will truly rely upon and as the scale and lifetime of our applications grows** (exactly the kind of development you'll do when you leave the academic environment and start working on real production software).

- In a more precise way, this is really what CS 362 is all about: strategies we can take and processes we can put in place to minimize the number of our mistakes

that impact the users of our software, even as our code scales in complexity and use.

- The topics we study in this course will fall into three broad categories, each of which represents a different aspect of what is needed to create and maintain production-quality software:
    - Software testing
    - Process automation (i.e. continuous integration and continuous delivery/deployment)
    - Strategies for working with other developers on a shared code project

- Before we look at each of these categories in a bit more detail, it's important to note that **CS 362 is mainly a technical course**. In particular, most of what we'll do in this course will involve writing code, configuring processes, and using various technical tools, such as a version control system.

# Software testing

- One of the most important things we can do to help ensure the quality of the software we release is to test that software.

- Of course, as developers, we are (or should be) constantly testing our own software informally and manually, e.g. by manually running our code and noting its results.

- In this course, we'll study software testing that differs from the informal testing we do everyday in two important ways: it will be formalized and it will be automated.

- In the form of formalized, automated software testing we'll study in this course will involve writing code that applies various ***test cases*** to our software. At a high level, this involves running our software with specific combinations of inputs under specific execution conditions and verifying that our software produces the expected results.

- We'll specifically study several different kinds of automated software testing that differ from each other primarily in their scope, i.e. *how much* code is under test:
    - **Unit testing** – Unit tests are designed to validate small, focused pieces of code, e.g. a single function.
    - **Integration testing** – Integration tests are designed to validate the interactions between multiple software components.

    - **End-to-end testing** – End-to-end tests are designed to validate large, connected portions of our code, e.g. an entire screen/page or user flow.

- For each of these different types of software testing, we'll study tools and techniques for implementing and running tests as well as testing best practices.

- In addition, we'll study a popular development strategy known as ***test-driven development***, which places software testing at the core of development.

# Continuous integration and continuous delivery/deployment

- As you have likely experienced, much of software development does not involve directly writing code but rather carrying out various processes, e.g. running commands in the terminal to build our software, run automated software tests, publish files, etc.

- These processes, when carried out manually, become breeding grounds for mistakes. Each command we type (or copy/paste) is another opportunity to make a typo, to choose the wrong file, etc.

- In this course, we'll learn techniques for automating these processes, which can dramatically reduce the number of opportunities we have to make mistakes.

- The automation techniques we'll study fall under a broad umbrella of techniques known as ***continuous integration and continuous delivery/deployment*** (or just ***CI/CD***).

- Under a typical CI/CD setup, developers frequently merge code changes into a central repository, where those changes are automatically built and tests are automatically run against them (this is ***continuous integration***). After these changes are built and tested, they are either ***delivered*** to a staging environment, where developers can test them further, or they are ***deployed*** directly to production, so users can immediately use them.

- Typically, CI/CD pipelines are built around a central version control repository, like a repo on GitHub. Integration and delivery processes are triggered automatically when code is pushed to this repository. Deployment processes are typically triggered automatically when code is pushed to a specific branch of the

repository (e.g. the main branch).

- In this course, we'll learn how to set up a CI/CD pipeline to automatically build, test, and deploy our software when we push code to our project's GitHub repository.

# Working with other developers on a code project

- As you've moved through your CS courses, most of your development has probably been solitary, with you mainly working alone on individual programming assignments.

- If you've had the opportunity to collaborate on a software project with a team of other developers, you've likely noticed that developing with a team is very different from developing on your own.

- One of the greatest challenges of working with a team of other developers is that no single developer any longer has total control over the codebase. Instead, each developer contributes pieces of code to a larger whole, trying their best to make their own code fit into the code developed by the rest of the team.

- As you may have encountered, problems can arise when one or more developers aren't in sync with the rest of the team. For example, a developer can make wrong assumptions about how another developer's code works, or a developer could make changes that undo progress made by another developer.

- In order to protect against these types of issues, it is important for teams to adopt workflows and processes that help keep developers in sync.

- You should have studied some of these workflows and processes in CS 361. In this course, we'll study another that is more directly centered around the code itself called ***code review***.

- Code review is a process in which changes to the code are reviewed by one or more developers other than the author before those changes are accepted into the codebase. These reviewers assess the changes, offer feedback, suggest improvements, and ultimately determine whether and when the code is good enough to be accepted into the project.

- This process typically requires a specific workflow to facilitate easy code review as well as a tool for performing the review itself. In this course, we'll explore a workflow that centers around a GitHub repository, and we'll use GitHub itself as the code review tool. This setup is used commonly across the software development world.

- By ensuring that multiple developers look at every change to the code, this process of code review has the obvious benefit of helping to detect and eliminate bugs before they reach the users of a piece of software. In addition, it offers other, more subtle benefits, such as helping to distribute knowledge about the codebase across the development team.

# CS 362 is a hands-on course

- **CS 362 will be a hands-on course, designed to give you practical experience with tools and techniques that are used widely in the software development world.** We use testing frameworks and write tests in this course. We will set up CI/CD pipelines. We will implement code that is intended for review, and we will perform code review on each other's code.

- The goal of the course is to give you practice with these tools and processes in preparation for using them when you leave college and enter the software development world. In particular, if you get comfortable with these things now, it'll be much easier to use them later in a real production setting.

- With this goal in mind, we will focus on technical/programming assignments and a final technical project in this course rather than on written exercises and quizzes/exams.

- In particular, we will have 4-5 assignments that will all involve some combination of writing code and setting up tools and processes, and we will have a final project in which you work with a small team to apply the skills you learn in this course.

- Because much of the work in this course will involve writing code, we will adopt a common programming language for the course. Specifically, any lectures and assignments that involve coding will use JavaScript, since this is a language you should already have some exposure to.

- Importantly, we will not use any advanced features of JavaScript in this course, and the JavaScript programming we do will be very straightforward, so if your JS skills are a little rusty, or if you haven't used JavaScript before, the syntax should be easy to pick up (again, if needed) and understand.