Curriculum Overview

Our Goal

The goal of the WeThinkCode_ Curriculum is to facilitate learning that will transform students into competent, entry-level software developers who:

- can quickly add value to the software development teams and organisations they join; and
- are equipped with the technical and behavioural skills to self-direct their learning beyond their training at WeThinkCode_.

This model is built from a holistic view of the software developer's capabilities and skills to be effective in their daily work.

Our Pedagogy

We train using a peer-to-peer methodology which means there are no lecturers. Learning material is delivered digitally, and students support one another in their learning. The style of instruction supports self-directed and peer-to-peer learning, which is a necessary skill set of an effective professional software developer.

We built a proprietary Learning Management System (LMS) that directly implements our pedagogy and essential software development practices expected by the industry. In addition, the LMS seamlessly integrates with tools widely used globally, such as git and PGP. This combination promotes the fundamentals of programming and provides valuable experience using prevalent tools.

Students are introduced to concepts in bite-sized chunks in the coursework and subsequently required to demonstrate their understanding by submitting exercise solutions that are functional and working at every step.

We leverage some additional constructs to supplement the coursework, such as the Technical Mentor Program and Code Clinics.

The Learning Management System

The LMS client is a command-line application used to deliver curriculum content to students and track their submissions and progress. All exercises come with a set of automated tests that must execute successfully for the student to submit. Additionally, the LMS pairs students with each other so that their submissions undergo peer reviews. The reviews encourage peer- learning, sharing healthy feedback and enhancing code comprehension.

In-Person Support

While there are no lecturers on campus to augment the peer-to-peer learning environment, we also provide in-person support in the form of Technical Mentors and Code Clinics.

WeThinkCode_ has a team of software developers that use the LMS data to track the progress and performance of students. This team provides further support for students as needed.

- Code Clinics

  Stronger students may volunteer their time to help students who are otherwise struggling. Code Clinics encourages cross-pollination and peer-to-peer teaching and learning.

- Technical Mentorship

  Technical Mentors are a nominated set of second-year students with good performance to support groups of first-year students. Technical mentors act as the interface between WTC_ staff and first-year students. In addition to being a source of technical support, they also serve as a channel to drive non-technical outcomes by facilitating activities such as daily/weekly updates, task management and reviewing presentations designed to build confidence in speaking in front of groups.

Coursework Format

Software development practitioners who are respected and highly recognised and have a collective experience spanning several decades design the curriculum. The WeThinkCode_ curriculum authors continually refresh the curriculum based on feedback from students, partners and industry trends.

Individual and team projects make up the coursework of every module. Students design and deliver on the project requirements using programming languages, development tools, technologies and practices adopted by high performing software delivery companies.

Every semester focuses on the holistic learning of:

- **programming** as the act of designing and writing code for a system;
- **engineering** as the set of techniques for automated testing, deployment and running of programmed systems;
- **communication** which is the foundation for analysis, design and collaboration; and
- **teamwork** because software developers inevitably work in teams to deliver software products.

Block 1 (8 months)

101 Programming Fundamentals

This module introduces programming constructs that are the foundation of any kind of programming. For many students, this is their first time programming. To overcome the technical vocabulary of computer science, we present the fundamentals of programming in plain language. Once the conceptual understanding is embedded, then only is the computer science terminology introduced.

In this module, the language of instruction used is Python. We chose Python because it is one of the more accessible programming languages to understand and is widespread globally.

By the end of this module, students:

- will know the programming fundamentals, including variables, branching, loops, data structures, procedures and functions, error and exception handling, packaging their code into modules and using external libraries;
- are comfortable using unit tests to prove their programs' functionality and write elementary unit tests of their own;
- have practised the basics of git and branching;
- will have learned and implemented OAuth using the Google OAuth API;
- have written code that integrates with an external HTTP RESTAPI, including JSON;
- can write programs that use configuration files to control the parameters of a system; and
- know how to write programs that read and write files.

Programming

- **Making Decisions:** Make a program do different things based on the data it receives.
- **Repeating Instructions:** Get a program to do the same thing several times over.
- **Structuring Data:** Combine data into meaningful structures.
- **Combining Instructions:** Combine several instructions and reuse the combined instructions as a single instruction.
- **Processing Collections of Data:** Work with multiple instances of the same kind of data in various ways.
- **Modules and Packages:** Use code from other developers (including open source) to construct a more extensive program from smaller modules.

Engineering

- How to structure code files and package them to be run.
- How to write a program and test that the core program is functioning as intended.
- How to run automated tests
- Introduction to Test-Driven Development
- Use PGP sign git commits

Communication

- **Journaling:** Reflecting on various learnings and their application and summarising them, in written format, in a journal. Journaling hones the ability to describe technical concepts and summarise them in a way that others can understand. The more reflective a developer is, the better they are at communicating.
- **Presentations:** Delivering short-form presentations known as Pecha Kucha to peers. Giving presentations develops public speaking skills and the ability to prepare slides and use them to deliver information effectively.

Team Work

- **Collaboration:** Working in small teams to build a program that, ordinarily, is too much for one person to complete on their own in the prescribed time. Students must collaborate in groups of 4 to build a program that works end-to-end.

- **An Iterative Way of Working:** The teams work on a group project in 3 iterations of 2 weeks each. Time-boxing the scope of work introduces concepts of planning and the necessity of shared understanding.
- **Demos:** At the end of the semester, each group present their projects to staff and guests from the industry.
- **The Team Project:** The brief is to build a console-based system that manages the Code Clinics. The project requires students to use their fundamental knowledge to integrate with the Google Calendar API to implement the code clinic schedules. Each team collaborates on the code using a shared git repository on GitHub.

102 Object-Oriented Programming

This module is an introduction to software design through the lens of Object-Oriented Programming (OOP).

We assume that students are comfortable with fundamental programming constructs introduced in the 101 Programming Fundamentals module. In this module, we introduce Java as the programming language. The switch from Python to Java further entrenches the fundamentals of programming.

By the end of this module, students:

- will know the basics of object-oriented programming such as encapsulation, inheritance, polymorphism and composition;
- would have refactored procedural code to object-oriented code;
- have been introduced to DevOps by using GitHub actions to build, test and package their programs;
- have programmed a client-server architecture system from scratch;
- will know the fundamentals of network programming by programming with sockets;
- have written a multi-user system using thread-based concurrency; and
- will learn how to implement a custom application protocol with JSON.

Programming

- **Java Fundamentals:** How to use the basic programming constructs in Java to mould the design of a program.
- **Encapsulation:** Create classes to implement behaviour, hide and govern access to data.
- **Inheritance:** Use abstraction as an instrument of design.
- **Polymorphism:** Implement specialisation of objects at run-time, based on the abstraction designed.
- **Composition:** Combine objects to create more complex objects.

Engineering

- **Build tools:** Build using Apache Maven, the most popular build tool for Java programs, to manage program dependencies and compile and test programs.
- **Unit Testing:** Use JUnit to reinforce existing testing techniques.
- **Refactoring:** Refactor a procedural program to an object-oriented program.

Communication

- **Team Leads & Daily practices:** Team leads are nominated to be the interface to each team. The teams continue honing their agile techniques such as stand-ups, managing a task board, iteration planning, showcases and retrospectives.

Team Work

- Students must collaborate in teams of 4 to build a program that works end-to-end. The project consists of 3 two week iterations.
- Students follow agile practices for building software, including pair programming, daily standups, showcases, and retrospectives at the end of each iteration. A final demo takes place at the close of the semester.
- **The Team Project:** The brief is to build a multi-user network-based game called RobotWorlds wherein players launch robots into a grid world and combat each other. The system requires students to apply their knowledge of object orientation to implement client socket-based communication over a network and a multi-threaded server. The choice of the user interface is left to the team to decide. Groups share their code on GitHub and are required to use GitHub actions to automate their programs' build, test, and packaging.

Block 2 (8 months)

201 Brownfield Software Development

Most software development jobs entail programming in an existing codebase. This kind of programming is known as brownfield development. Furthermore, software developers work in teams. This module, therefore, focuses on how to work on an existing codebase as a team.

Students work exclusively in the team for the entire module with a codebase that is given to the team. We also introduce other technologies, practices and techniques expected of software developers in the workplace.

By the end of this module, students:

- have worked in a team using agile principles and practices;
- have analysed a codebase to understand its technical debt and quality;
- have continually practised refactoring an existing code base to improve its quality and repay the technical debt;
- can write unit tests, acceptance tests and integration tests;
- will have implemented a build pipeline for continuous integration and delivery;
- will have packaged an application using Docker;
- programmed persistence with SQL based relational databases and an object-relational mapper;
- will have learned how to recognise commonly encountered problems and solve these using design patterns;
- understand coupling and cohesion in software design and architecture.

Programming

- **Automated Testing:** Write unit tests, acceptance tests and integration tests.
- **Persistence:** Use a relational database and SQL to store and retrieve data.
- **Code Analysis:** Analyse a codebase to establish its technical debt and code quality.
- **Refactoring:** Refactor existing code to improve its quality and design by applying TDD.
- **Design Patterns:** Apply patterns to enhance the structure at the code level.
- **Fundamentals of Software Architecture:** Understand coupling and cohesion.

Engineering

- **DevOps**: Create build pipelines for continuous integration and continuous delivery, including automated testing and deployment with Docker.
- **Effective Version Control:** Use either trunk-based development or pull requests to maintain a stable main branch of the codebase

Communication

- **Showcases:** Every iteration ends with a showcase. Students prepare and present their work for that iteration to mentors and staff.
- **Technical Documentation:** Illustrate the architecture and design of the system under development, from component level down to modules and classes.

Team Work

- **Planning:** Students are given goals for each iteration and plan, estimate and prioritise their work to achieve the iteration goals.
- **Taskboard:** Use a task board to manage the backlog of work, track work progress, and keep the board up to date via daily stand-ups.
- **Retrospectives:** Reflect on the iteration and identify and implement improvements in the way the team works.
- **Pairing:** Practice pair programming to increase shared understanding and quality of the code.

202 Client-side Web Applications

This module is half a semester.

By the end of this module, students:

- will have written a web-based application using HTML, CSS and JavaScript;
- know how to consume an HTTPAPI in the client-side web application;
- will be familiar with asynchronous programming;
- will know the difference between server-side generated web pages and client-side dynamic web pages.

Electives

Students are required to select one of the following electives as the last module in their studies.

All electives are half a semester.

203 Quality Assurance Engineering

In this module, students will learn:

- Test strategies
- Automated tests for web APIs
- Automated tests for client-side web applications (GUI testing)
- Continuous integration and continuous delivery
- Automated provisioning of run-time environments
- Regression testing

204 Mobile Application Development

In this module, students will learn:

- Cross-platform mobile application development with the Flutter framework
- Automated Tests for Mobile Applications
- Design patterns
- Consuming web APIs
- Packaging and deployment of mobile applications

205 Service-Oriented Architecture

In this module, students will learn:

- Service decomposition and design
- Inter-service communication using message queues
- Asynchronous communication and eventual consistency.

206 Data Engineering

In this module, students will learn:

- Understand what data engineering is and where it fits into the modern data lifecycle/landscape.
- Understand the difference between batch and streaming workloads.
- Understand data formats, how to structure data, and how to select the relevant/best technology for storing and processing.
- Build, monitor, and orchestrate data pipelines.
- Understanding data quality/integrity.

207 Cloud Development

In this module, students will learn:

- Understanding Cloud Computing Concepts
- Proficiency in Cloud Services and Platforms
- Cloud Application Development and Deployment
- Cloud Security and Compliance
- Cost Management and Optimization