Software Engineering, University of Oulu

**Lab 0: Introduction**

*Disclosure: The material in this paper has not been reviewed, endorsed, or approved of by the Rust Foundation. For more information on the Rust Foundation Trademark Policy, click* [*here*](https://docs.google.com/document/d/1ErZlwz9bbSI43dNo-rgQdkovm2h5ycuW220mWSOAuok/edit)*.*

1. **Overview**

In these laboratory exercises, you will learn about the practices of continuous integration (CI) and continuous delivery (CD) in software development. These are often referred to as CI/CD.

<https://en.wikipedia.org/wiki/CI/CD>

* 1. **Structure**

The exercises are divided into following chapters:

* **Lab 0: Introduction**
  + Overview and setup for exercises
* **Lab 1: Version Control**
  + Using Git and GitHub for version control of a Rust project
* **Lab 2: CI/CD**
  + Using GitHub Actions to create tasks for a CI/CD pipeline
* **Lab 3: Linting**
  + Improving code quality by linting a Rust project using Clippy
* **Lab 4: Unit Testing**
  + Creating unit tests for a Rust project

1. **GitHub**

The laboratory exercises utilize GitHub Classroom. You need a GitHub account in order to complete the exercises.

If you do not have an account yet, you can create one here: <https://github.com/>

**2.1 GitHub Classroom**

In the following exercises, we will utilize GitHub Classroom. You will be given a link to a Classroom repository which you will use to create a pre-made remote repository for each exercise.

1. **Rust**

The programming language used in these laboratory exercises is [Rust](https://en.wikipedia.org/wiki/Rust_(programming_language)). Rust is a systems programming language initially developed by Mozilla Foundation. Rust's primary goal is to provide a reliable alternative to low-level languages while avoiding common pitfalls of low-level programming, such as memory errors. Its performance is comparable to C and C++ with a very minimal runtime, which makes it suitable for many applications. In the future, Rust may play a big part in embedded systems, [including Linux kernel programming](https://thenewstack.io/rust-in-the-linux-kernel/).

**3.1. Installing Rust**

Download Rust from the official website: <https://www.rust-lang.org/>.

(Note that you might also need to install [Microsoft C++ Build Tools](https://visualstudio.microsoft.com/visual-cpp-build-tools/), in case you do not have them on your machine already.)

After the installation is completed, open a command prompt and type:

rustc --version

If the command prints out a version number, the installation was successful.

**3.2. Creating a new project**

After successfully installing Rust, we can create our first project. In the exercises, you will be given pre-made templates, but it is important to understand the initialation process and project structure beforehand.

Open a command prompt and navigate to a folder where you want the project folder to be saved in.

Create a new Rust project (called example) by typing:

cargo new example

Open the project in your preferred code editor. If you do not have one, [Visual Studio Code](https://code.visualstudio.com/) is a good choice.

The default setup should include a simple Hello World program.

**3.3. Compiling a Rust program**

Rust is a [compiled language](https://en.wikipedia.org/wiki/Compiled_language) so the source code cannot be directly run.

While in the project folder, compile the source code in command prompt by typing:

rustc src/main.rs

An executable file called main.exe should appear in the project folder.

You can run the program simply by typing:

./main.exe

If the program prints out “Hello, world!”, you have created and compiled your first Rust program.

Some editors or their extensions offer convenient ways to run your program without using the command line:

A computer screen shot of a computer code

Description automatically generated

1. **Notes**

* If you want to dive more into Rust syntax, you can find many [tutorials on YouTube](https://www.youtube.com/watch?v=ygL_xcavzQ4).