KARL STORZ develops, manufactures, and distributes medical devices in the scope of minimally invasive surgery. To comply with the high-quality standards of medical industry, KARL STORZ follows a Global Product Development Process. As part of this process, software developers must follow the Free and Open-Source Software (FOSS) compliance process to protect intellectual property rights. This requires any software product to be audited, reviewed, and approved, so that the company fulfills the license obligations resulting from using the various software components integrated in the product.

As maintaining FOSS components is costly and time-consuming, the process of gathering FOSS information should be automated as far as possible.

In a previous student research project, a feasibility study for automating the FOSS process was conducted and a prototype application was implemented (FOSSer tool). The prototype was able to create the Bill of Material for any Golang Application and gather meta-information of third-party dependencies from the official source repositories.

Within this research project, the existing prototype should be extended to work reliable with different programming languages that are used by KAL STORZ to create a FOSS manifest that complies with the KARL STORZ Global Product Development Process.

Within this practical semester, the basic concepts of the FOSSer tool and its environment should be analyzed and documented. Furthermore, a tooling evaluation should be conducted to create a concept for the future architecture of the FOSSer tool and its extensions for multiple languages.

The results of the evaluation should create a basis for the future implementation of the FOSSer extension as part of the T1000 thesis (24.07. – 01.10.2023).

**Step 1: 13.02.2023 – 21.02.2023: Golang Hands-On**

* Goals: The apprentice gets introduced to the programming language Golang and understands its core features, syntax, and general project structure. At the end of the week, a small Golang application shall be implemented.
* Resources:
  + [Tour of Go](https://go.dev/tour/welcome/1)
  + [Go by Example](https://gobyexample.com/)
  + [Go Project Structure](https://github.com/golang-standards/project-layout)
  + [Package Management](https://go.dev/doc/modules/managing-dependencies)
  + Further Online Research
* Task:
  + Environment Setup
    - Setup latest Golang runtime on your local computer
    - Install development environment for Golang (vs code & Golang extensions)
    - Implement, build, and run the “Hello World” application using Golang
  + Introduction to Golang
    - Conduct the “Tour of Go”
    - Research the main concepts of Golang
    - Understand the differences between Golang and the classic OOP languages like Java, C#, C++
  + Hands-On: Implement a small weather forecast app
    - Create a new private repository “WeatherForecastGolang” on GitLab
    - Setup Git integration for the project
    - Setup the project structure (stick to common Golang project structure)
    - Implement the following requirements:
      * The console application should fetch and print the weather forecast to the console line. To get weather data, the app should use the data from the [Open Meteo Weather API](https://open-meteo.com/)
      * It should be possible to pass a city name as a parameter to the application startup. The application should then provide a weather forecast for the city.
      * If no city name was passed as a parameter to the app, then the app should allow the user to type in a city name for which to create a forecast.
      * A weather forecast should consist of the following items
        + temperature
        + rain
        + showers
        + cloud cover in percentage
        + humidity
      * The forecast should show a summary of the forecast parameters for every two hours of that day, starting from the point of time when the forecast was queried/the app was started.
      * The Open Meteo Api is based on the geo location of a city. So the app has to translate a city name into its geo location to query the weather API. A good starting point is the free [World Cities Database CSV](https://simplemaps.com/data/world-cities) file
      * Consider the SOLID & Clean Code principles!
    - Implement some unit test to get in touch with the unit testing frameworks in go

**Step 2: 22.02.2023 – 24.02.2023: CI Hands-On**

* Goals: The apprentice gets a basic understand of Continuous Integration at KarlStorz and can define a basic CI pipeline for an existing Golang project
* Resources:
  + CI introduction by Thomas Fritz
  + Online Research
* Contact:
  + Thomas Fritz (infrastructure)
* Task:
  + Define a basic CI pipeline for the Weather Forecast Application that was created previously. The CI pipeline should contain at least three different stages:
    - Linting: The source code gets scanned by a linter. The stage should fail if there are any linter warnings or errors
    - Test Stage: Executes the Unit Tests
    - Build Stage: The application shall be built as a standalone binary with Release configuration

**Step 3: Requirements Engineering**

* Goals: The current status quo and the existing architecture / structure of the FOSSer tool should be documented. The apprentice should get a basic understand of the problem. Requirements for the extension of the FOSSer tool should be analyzed and documented
* Resources:
  + FOSSer introduction by Matthias Konkel
  + GPDP / FOSS introduction by Tasso Stroehle
* Contact:
  + Matthias Konkel (RDM)
* Task:
  + Document the current status quo and architecture of the FOSSer tool
  + Gather requirements from the different teams (NBR, MDC & Gamma)
  + Document the requirements

**Step 4: Tool Evaluation**

* Goals: Based on the requirements for the FOSSer tool, an evaluation matrix should be defined to evaluate different tools for generating the Bill of Materials for any application. The evaluation should be documented and comprehensible. Based on the matrix, a tool for the extension of the FOSSer should be selected. Depending on the tooling choice, a possible architecture, and a concept for the integration of the FOSSer into the CI environment should be defined (in agreement with the RDM and infrastructure team)
* Resources:
  + [Decision Matrix](https://cloudfresh.com/en/blog/decision-matrix-7-steps-to-make-a-decision/)
  + Online Research
* Contact:
  + Matthias Konkel (RDM)
  + Thomas Fritz (infrastructure)
* Task:
  + Define different criteria for the evaluation matrix based on the previously defined requirements
  + Define weights and a score system for the different criteria (are there any deal breakers?)
  + Compare different BOM tools
  + Based on the total weighted score, a BOM tool should be selected
  + Discuss results with RDM team and adjust weights if necessary
  + Document results and findings
  + Define an architecture for the extension of the FOSSer tool (UML, textual description, …)
  + Define a concept of how to integrate the FOSSer into different CI pipelines
  + Document results