Maven Enterprise Dependency Management

Standardising the pom of micro services so that we have separate version and configuration control for them... by using bom.

https://www.baeldung.com/spring-maven-bom#:~:text=BOM%20stands%20for%20Bill%20Of,that%20we%20should%20depend%20on

Learn about bom from above link.

We can override some properties of parent pom by declaring in our own pom. We can see out effective pom to see our over-riden changes.

Dependency management allows to set up the dependency that can be inherited from this pom downstream. Basically we can setup what version of dependencies we want.
They are not transitive dependency of that project, necessarily we are not making it dependency of downstream projects. We are saying if we gonna use this dependency we will inherit the version.

Refer to this for more about bom and pom:
https://stackoverflow.com/questions/38882221/how-to-inherit-dependency-from-a-parent-pom-to-a-child-pom

If we setup direct dependency in parent pom then it becomes dependency for all child pom. It becomes optional for child pom to include it. They can also handle scope.

Awaitility is an artifact used for testing. When we have something synchronous and give time to complete then we use it....

For maven bom common dependencies to be used by all child, we can declare that in dependencies section directly....

If we are inheriting pom of spring boot we do not need to mention the versions they will inherit those versions as they have same bom as we are making for our own....

We can override spring boot starter test -> spring boot 2.1 is still using junit 4, if we want junit 5

```
<dependencies>
 <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-test</artifactId>
        <scope>test</scope>
```

```
            <exclusions>
               <exclusion>
                  <groupId>junit</groupId>
                  <artifactId>junit</artifactId>
               </exclusion>
            </exclusions>
 </dependency>
<dependency>
            <groupId>org.junit.jupiter</groupId>
            <artifactId>junit-jupiter-api</artifactId>
            <scope>test</scope>
        </dependency>
        <dependency>
            <groupId>org.junit.jupiter</groupId>
            <artifactId>junit-jupiter-engine</artifactId>
            <scope>test</scope>
 </dependency>
</dependencies>
```

We can also declare common build plugins under build section for our downstream projects... we can also add configuration (like auto clean running every time we build our project).

We can setup maven enforcer plugin to setup required versions of maven (like 3.6 brings additional features for junit 5) and java, etc...
There is also <requireReleaseDeps> which ensures at time of release we have only release dependencies.

Learn more about relative path in pom
https://stackoverflow.com/questions/37062491/maven-complaining-about-parent-relative-path

To work within a single folder and have many projects
Create empty project and then go to file and then new and then click on add existing module.