University of West Bohemia
Faculty of Applied Sciences
Department of Computer Science and Engineering

Bachelor's thesis

Maven plugin for identification of incompatible changes in product source code

Pilsen, 2022

Martin Brožek

ZÁPADOČESKÁ UNIVERZITA V PLZNI
Fakulta aplikovaných věd
Akademický rok: 2021/2022

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

(projektu, uměleckého díla, uměleckého výkonu)

Jméno a příjmení:	**Martin BROŽEK**
Osobní číslo:	**A18B0180P**
Studijní program:	**B3902 Inženýrská informatika**
Studijní obor:	**Informatika**
Téma práce:	**Maven plugin pro identifikaci nekompatibilních změn v produktovém zdrojovém kódu**
Zadávající katedra:	**Katedra informatiky a výpočetní techniky**

Zásady pro vypracování

1. Seznamte se s přístupy, jak se v zákaznických projektech rozšiřují produkty firmy Eurosoftware s.r.o.
2. Seznamte se s programovacími jazyky a technologiemi použitými v produktech firmy Eurosoftware s.r.o. (Java, XML, domain specific languages).
3. Prostudujte obecné přístupy, jazyky a nástroje vhodné pro analýzu kompatibility zdrojového kódu.
4. Dle předchozích bodů navrhněte a implementujte nástroj (Maven plugin) pro analýzu zdrojového kódu v produktu a jeho kompatibility s customizacemi pro jednotlivé zákazníky. Nástroj bude identifikovat změněný zdrojový kód mezi dvěma verzemi produktu, který není kompatibilní s projektovým kódem.
5. Výsledné řešení otestujte a zhodnoťte.

Rozsah bakalářské práce:	**doporuč. 30 s. původního textu**
Rozsah grafických prací:	**dle potřeby**
Forma zpracování bakalářské práce:	**tištěná/elektronická**
Jazyk zpracování:	**Angličtina**

Seznam doporučené literatury:

Dodá vedoucí bakalářské práce.

Vedoucí bakalářské práce:	**Ing. Radek Hoštička** Eurosoftware s.r.o.
Konzultant bakalářské práce:	**Doc. Ing. Roman Mouček, Ph.D.** Katedra informatiky a výpočetní techniky
Datum zadání bakalářské práce:	**4. října 2021**
Termín odevzdání bakalářské práce:	**5. května 2022**

L.S.

Doc. Ing. Miloš Železný, Ph.D. děkan	**Doc. Ing. Přemysl Brada, MSc., Ph.D.** vedoucí katedry

V Plzni dne 14. října 2021

Declaration

I hereby declare that this bachelor's thesis is completely my own work and that I used only the cited sources.

Pilsen, 23rd June 2022

Martin Brožek

Abstract

The bachelor thesis is focused on designing and developing a new tool for the company Eurosoftware s.r.o.. The tool described in the thesis will be able to recognize code changes in the company's main product source code, which could affect project migration from one product version to another. The tool is intended for ES's project developers to save time during the project version migration. The current migration process is done entirely manually by the developers. The tool execution will be able to point out files on which the developer should focus during the migration process.

Abstrakt

Bakalářská práce je zaměřena na návrh a vývoj nového nástroje pro společnost Eurosoftware s.r.o.. Nástroj popsaný v práci bude schopen rozpoznat změny ve zdrojovém kódu hlavního produktu společnosti, které by mohly ovlivnit migraci projektu z jedné verze produktu do druhé. Nástroj je určen pro projektové vývojáře, aby ušetřili čas během migrace verze projektu. Současný proces migrace je manuálně vykonáván vývojáři. Spuštění vytvořeného nástroje ukáže na soubory, na které by se měl vývojář během procesu migrace zaměřit.

Contents

1 Introduction

Eurosoftware s.r.o. is a subsidiary of a German company, GK Software SE, offering a sophisticated retail solution called Omni-Chanell worldwide to customers such as Coop, Lidl, Adidas, and many others. The solution, among other things, provides functionality for selling items and supports several payment types, handling discounts, and customer-oriented features.

After a customer purchases the product, it transfers to a project, which personalizes the product for a customer based on their needs. After the new product version is released, the project should migrate to the newer version.

The current version migration process is done manually by developers. During the process, developers have to download all artifacts, including product source code for both versions, then compare them together and with the project to find all files, which would cause problems after migration. The current comparison procedure does not consider inheritance, leaving it up to a developer to go through all the files, potentially affecting the initial one.

The goal is to create a solution for identifying files that could cause problems during migration. The solution must be capable of downloading two versions of the product from the company's repository and comparing them with the selected project, producing reasonable output in the process.

The thesis provides the migration tool's development process, including an analysis of the company's product, technologies used in the development, versioning, and the current migration process leading to solution options and reasoning behind the selected choice.

2 GK Software SE

The chapter provides basic information about the company and primarily focuses on POS product with a succinct description of the product components and their functionality. It also explains the difference between product and project at the end, which is crucial for the thesis.

2.1 About company

GK Software SE is a worldwide German company founded in 1990, focusing mainly on providing software and services to operate sizeable retail company branches such as Lidl, Addidas, PGA TOUR Superstore, and many others. GK Software SE's primary mission is to provide the most innovative and adaptable retail solution possible.

Its leading product is Omni-Channel, the primary product GK Software SE offers to its customers. Omni-Channel is a sophisticated retail solution providing Point of Sale Software to the end customer retail establishment [1].

The company is also committed to developing an innovative solution based on cloud technologies and artificial intelligence to become a leading cloud solution provider.

GK Software SE is deeply involved in the research and development of new technologies by trying to identify future trends in software development in advance, currently developing applications using virtual reality and Kinect.

This thesis was developed under the leadership of GK's second-largest subsidiary company Eurosoftware s.r.o. located in Pilsen, focusing on the Point of Sale, the end of Omni-Channel, and its version migration [2].

2.2 Point of Sale

Point of Sale (POS), standardly referred to OmniPos, is software providing functionality for selling items and supports several payment types. OmniPos robust set of services gives a customer access to all functions, not just in the store but across the entire enterprise. It also enables other features related to this primary goal, such as handling discounts and customer-oriented features.

POS functionality requires several types of hardware (such as receipt printers and terminals), depending on customer store policies. It handles two databases master data (read-only) and transaction (read/write) [3].

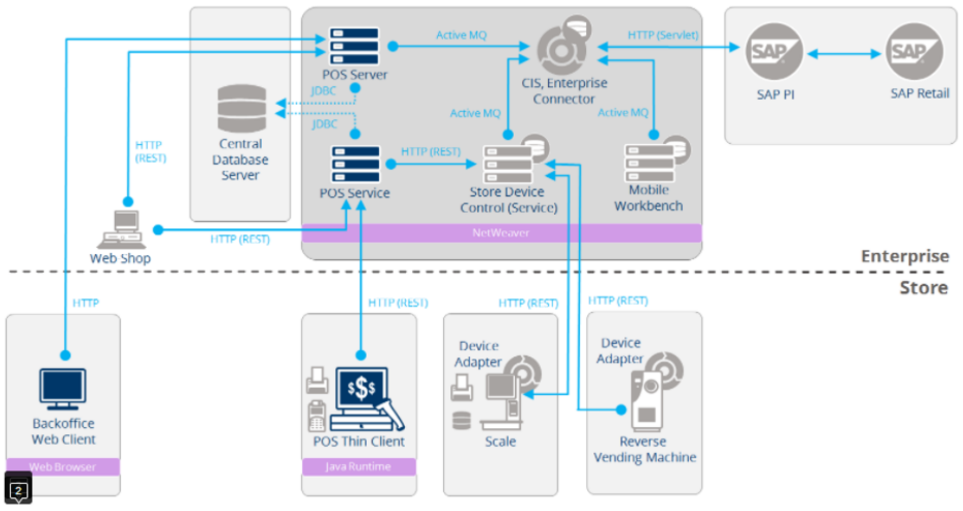


Figure 2.1: Big picture diagram displaying Thin-POS variant [4].

2.2.1 Central

The central concept, occasionally mentioned in the thesis, is that it serves as a wrapper for all the server-side services. Central consists of services such as POS server, SDC, ECON, and others. Through the POS server and POS service, it also handles the database.

POS server

The central application POS server's primary purpose is transaction processing through its portal functions for controlling financial movements, such as loading, saving, and searching transactions. It also handles creating reports and exports [5].

SDC

Purpose of Store Device Controller (SDC) is processing of the master data and distributing it to other applications in the GK/Retail OmniPOS. The distribution can be done with data replication or by services like SDC API [5].

ECON

The Enterprise Connector, also known as Universal Connector, represents a POS connector to an external world. ECON supplies communication with other systems through imports and exports. Incoming data are identified and mapped here to the internal format (XML), and outgoing data are mapped to the required external format [5].

CIS

The Cluster Infrastructure Server is a central data distributor. CIS handles all Java Message Service processing and distribution. It is also the first central application that needs to run [5].

POS service

POS service is a connector between POS-Client and the rest of the Central, persisting data to the services. In the Thin-POS variant, the client's data change must go through the service.

2.2.2 POS - Client

POS can be divided into three parts based on the database position. Thin-POS, databases are accessed remotely [6]. Fat-POS, databases are embedded within [7], and Smart-POS, a combination of both [8].

The POS-Client's main functionality is interaction with user and database data representation. It consists of a user interface, hardware, and client libraries. The Thin-POS variant fetches all the data through the POS service. This makes Thin-POS client highly dependent on network stability [6].

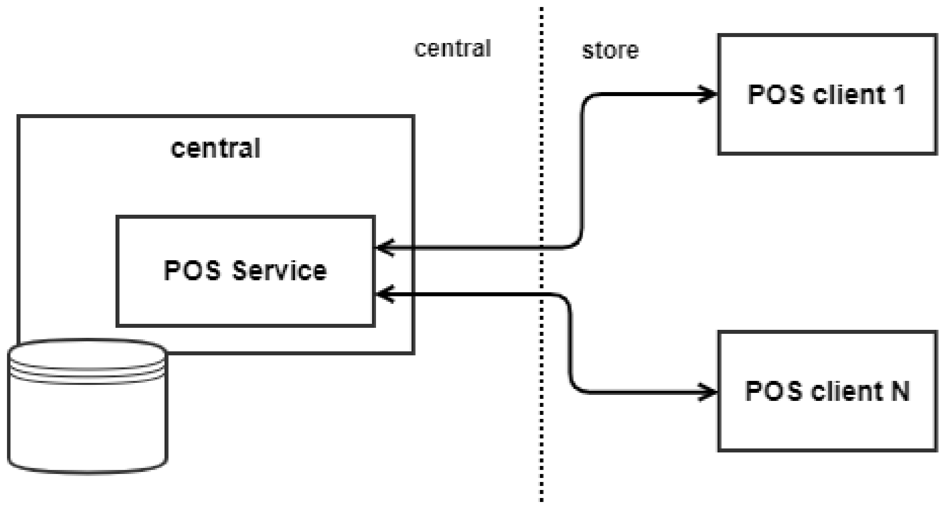


Figure 2.2: Thin-POS variant Central connection [6].

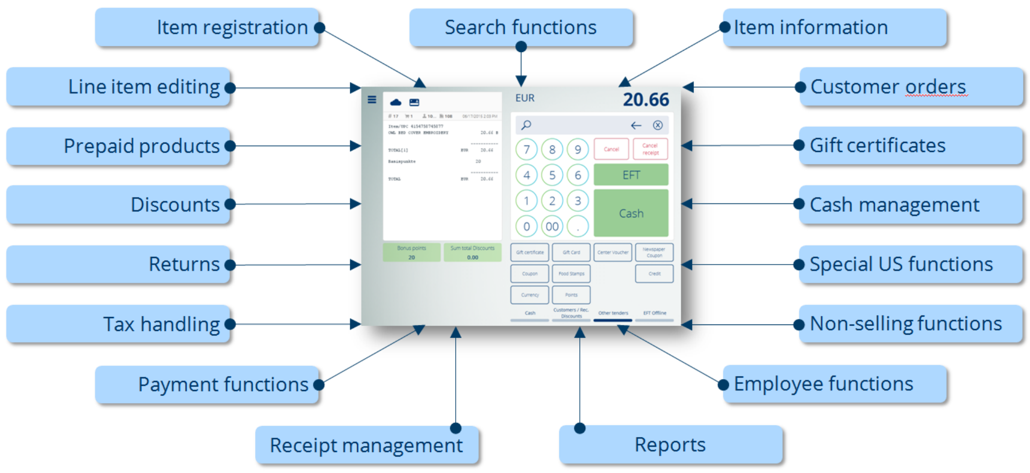


Figure 2.3: POS-Client functionality overview [9].

2.3 Company product vs. project

The thesis focuses on GK Software SE's product OmniPos and its migration between product versions. After an unspecified company buys the product, it transfers to a project. Project personalize product for a customer based on their needs. After the new product version is released, the project should migrate to the newer version, which is long and expensive. Project developers must manually compare custom project files and every file potentially influencing them during the migration.

3 Technologies used in ES product development

At the begging of the chapter is a short description of tools and frameworks used in ES's product development, followed by languages on which the thesis primarily concentrates. The development languages play a significant role later in the file parsing section of the analysis (§6.3).

3.1 Tools and frameworks

3.1.1 Maven

Apache Maven is a project management and comprehension tool for software projects. Maven can manage a project's build, reporting, and documentation from a single piece of information, thanks to the concept of a Project Object Model [10].

POS is built as a multi-module Maven project, an aggregator POM, usually located in the root directory that controls a group of sub-modules. Sub-modules, which are regular Maven projects, are allowed to be built separately or through the aggregator POM [11].

3.1.2 Nexus

Nexus a binary repository server, developed by Sonatype, used mainly for centralizing, building, and storing software packages such as Java/Maven, npm, NuGet, PyPI, RubyGems, CocoaPods, and more, improving a company's security in the process [12].

3.1.3 Jira

Jira is a software tool for keeping bugs and issue records by the company Atlassian. It helps to manage and simplify project workflow through a flexible user interface for monitoring employees during work hours [13].

Developer logs their work hours on specific tasks in Jira. In ES, the issue's id is used in the company's git workflow (Table 4.1).

3.1.4 Java Swing

Java Swing is a widget toolkit for creating a Graphical user interface (GUI) in Java that includes many widgets. It comes with several packages for creating Java desktop applications. Swing has built-in controls for displaying HTTP or rich text format, such as trees, image buttons, tables, and many others. Swing components are platform-independent because they are developed entirely in Java [14].

3.1.5 Jenkins

Jenkins is a free, open-source server written in Java used for easing both Continuous integration (CI) and Continuous deployment (CD) as well as for process automatization of software tasks such as testing, building, and deploying an application [15].

3.2 Languages

The Point of Sale (POS) product source code is mainly compiled from Domain Specific Languages (DSL) and Java. The layout definition of the project is defined in XML files.

Due to the signed non-disclosure agreement with the company, the syntax of the DSL languages is described only to the extent relevant to the thesis and in accordance with the specification provided by the company (§5.3).

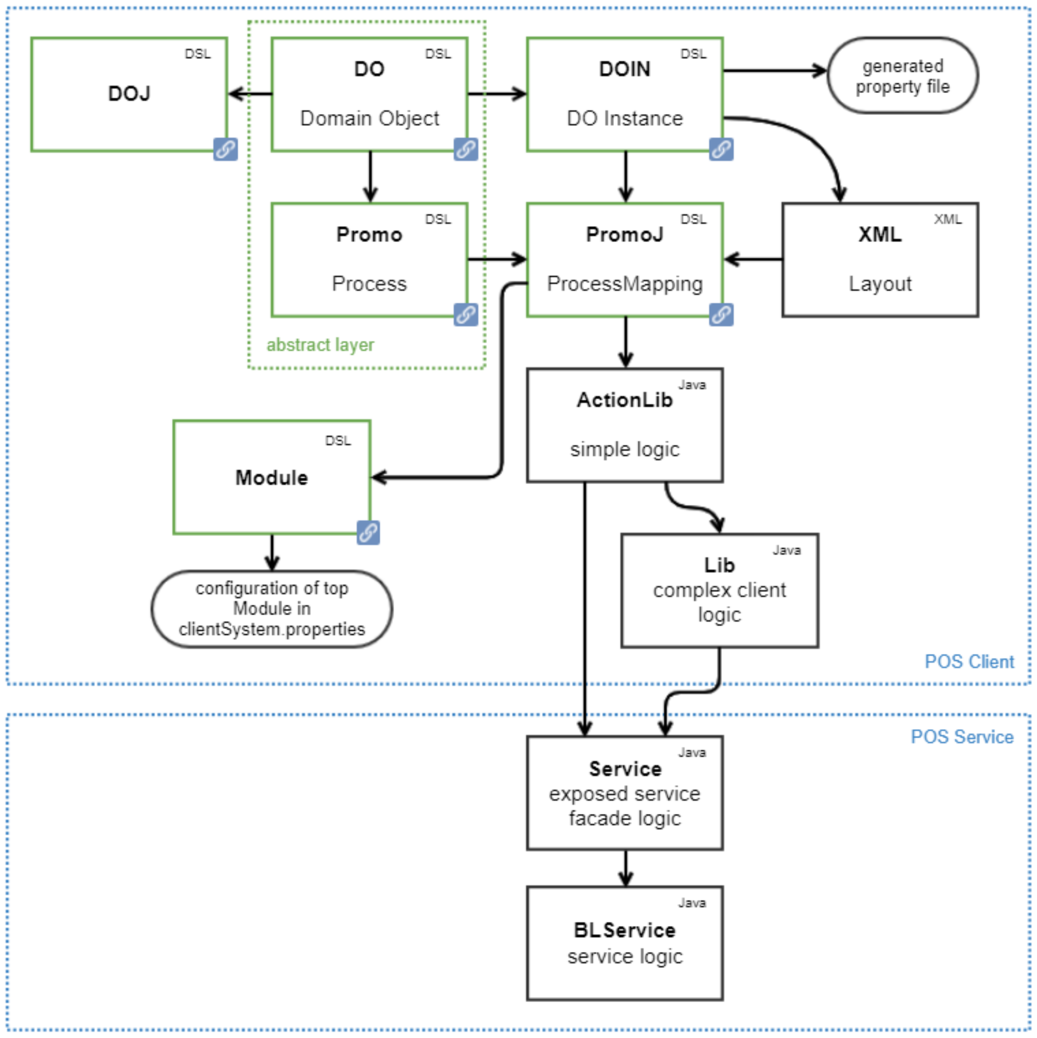


Figure 3.1: Diagram of languages used in POS [16].

3.2.1 Java

Java is an Object-oriented programming (OOP) language invented by Sun Microsystems. Java is designed to be simple, robust, secure, and portable. It is built to accommodate programs that will run in various network environments. Applications must be able to run on a range of hardware architectures in such environments [17], making Java a preferable choice for a primary programming language at ES.

```
package com.john.utilities;

import java.util.ArrayList;

class Subclass extends Superclass {

}
```
Listing 3.1: Exmaple of Java syntax including inheritance, import and a package [18].

3.2.2 XML

Extensible Markup Language (XML) is a markup language similar to HTML but does not have any predefined tags. Instead, users can create tags tailored to their needs. This is an effective method of storing data in a format that can be searched, stored, and shared [19].

3.2.3 CSS

Cascading Style Sheets (CSS) is a stylesheet language for describing the presentation of documents in HTML or XML and its dialects, such as SVG. CSS specifies how elements should be displayed on a computer screen [20]. In ES's development, it is used to help stylize UI and layouts.

3.2.4 Domain Specific Languages (DSL)

A Domain Specific Languages (DSL) is a programming language with a higher level of abstraction that is customized to a particular set of issues. The concepts and rules from the field or domain are used in a DSL [21].

It creates a way to describe and create program elements within an application domain (mechanism allowing application/code isolation [22]) without knowing anything about programming. A DSL gives significant productivity improvements and even allows end-user programming by offering notations appropriate to the application domain [23].

DSL used currently in ES's development can be divided into four categories: DO-DSL, DO-DSL-Generators, ProMo-DSL, and Module-DSL.

DO-DSL

The Domain Object DSL plays a significant role across development phases and software components. Its functionality is as specifiers that abstractly define the structure of entities used in merchandise management functions. Server developers provide entities as a database-independent representation of persistent data, and function implementors use these entities as their data layer [24].

The DO-DSL lets us use a technology-neutral specification of Domain Object and entities as the common reference point for all roles and components. It also allows us to perform Quality assurance validation at an abstract level [24].

DO-DSL can be divided into four categories by their extension [25]:

.do - Domain Object

.doj - Domain Object Java mapping

.doin - Domain Object Instances

.dodoc - Domain Object Documentation

DO-DSL-Generators

Generators are crucial in compiling code. They generate Java code from Domain Object and create properties configuration files from DOIN models or layout XML/properties from DOIN models for Store Manager and many others.

.do/.doj - Java classes (Pojos, Enums, Factories)

.doin - Property files (OPOS configuration)

```
Package com.gk_software.config.do_types.component.client

Import com.gk_software.pos.api.model.app.superclass

 domain object object-name {

     master entity Subclass extends Parent{

     }
}
```
Listing 3.2: Exmaple of DO and DOIN syntax including inheritance, import and a package.

ProMo-DSL

ProMo-DSL is a Domain Specific Languages for process modeling. It sets **common language** for various **stakeholders**/roles. ProMo-DSL speaks about the behavior of the system [25].

.promo - Process Models

.promoj - Process Models Java binding

ProMo-DSL is a language that eliminates ambiguity and misconceptions in communication. All relevant stakeholders, such as a consultant, interaction designer, product owner, tester, and developer, can communicate seamlessly with ProMo-DSL. Although the perspectives of these groups range substantially, they are all attempting to construct and improve the same system. Avoiding these misunderstandings can prevent problems while they are still easy to fix [26].

The primary purpose of a common language is to capture essential elements precisely. In this language, each sentence must have a clear meaning. The language encourages a usage in which each sentence makes a relevant judgment by focusing on critical components. It is not enough to have a language and the will to focus on the essentials and be precise. It is necessary to develop a process that will guide each stakeholder in effectively using the language and its tools [27].

```
Package processModels.tpos.simple.registration

Import processModels.tpos.simple.registration

Process Entity specializes Parent
```

Listing 3.3: Exmaple of Promo and Promoj syntax including inheritance [28], import and a package. Unlike Java, DO or DOIN, promo and promoj files do not use curly braces [28].

Module-DSL

The Module-DSL is intended to link the elements of other DSLs into Modules, with the application being the graph at the top of the module. As a result, the Module-DSL connects the semantic levels of the DSLs, such as DO-DSL and ProMo-DSL, with concerns of configuring and packaging modules and, eventually, the application. The module serves as a materials list. It also allows for decomposition, nesting, and the capture of declarations that influence many processes [29].

Module-DSL creates a structure of (empty) boxes. The module consists of declarations, such as package/namespace, imports, name, lists of included elements, lists of referenced elements, and list of contained nodes.

4 Version control

Version control is a class of systems responsible for managing changes to computer programs, documents, large websites, or other information collections. It falls into Software configuration management, which is task of tracking and controlling software changes [30].

Versioning is storing histories of all changes made on source code. Examples of the current best versioning systems are Git, AWS CodeCommit and Microsoft Team Foundation Server.

4.1 Version control in GK Software SE

GK Software SE uses git follows a Gitflow as a branching model primarily for its tooling support, good documentation and to increase stability and quality of the develop branch through continuous code reviews and merging standards into develop.

Gitflow provides a robust framework for a larger project by defining a rigid branching model devised around the release branch [31].

4.2 Branches

Branch	Naming convention
Stable latest release	`master` (`main` on new projects)
Development trunk	`develop`
Release	`release/<version_number>`
Feature or Bug-fix	`feature/<JiraID_ShortDescription>`
Hotfix (HF)	`hotfix/<JiraID_ShortDescription>`
Maintenance	`maintenance/<version_number>`

Table 4.1: Branch naming conventions in GK Software SE. JiraID is an identification of an issue located in Jira Software [32].

4.2.1 Master and develop branch

The Gitflow approach uses two branches to record the project's history instead of a single master branch. The master branch has the official release history, while the develop branch acts as a feature integration branch.

The master branch includes shortened project history, and the develop branch contains a complete version [33] [31].

4.2.2 Feature branch

The feature branches are used not only for developing new features for an upcoming release but also for any changes connected to the develop branch, such as bug fixes, technical or pom changes. The feature branch naming convention remains the same on all the mentioned occasions (Table 4.1).

After completion, the branch is merged back into the develop branch. The primary idea is that the feature branch exists as long as the feature is in development or abandoned [34].

4.2.3 Hotfix branch

A development team may need to address bugs without disrupting the rest of the development process or waiting for the next release cycle; that is where the Hotfix branch comes in.

The Hotfix branch's only purpose is for patching production releases. It is the only allowed branch that has to fork off directly from the master. After a fix completion, the branch should be merged into the master and the develop [35].

4.2.4 Release branch

The release is done on two occasions when all required features are added or the release date is looming. The release branch in the Gitflow approach allows developers to perform last-minute changes by permitting minor bug fixes and adding documentation directly to the release branch. This allows the develop branch to receive new features for the next release in the meantime. New features should never be added to the release.

The release branch is created from the develop, and after all altercations are made, it is merged into the master branch [36].

Release version name is set by pattern:

`[mayor].[minor][HF_number]_b[build_number]`

, where mayor stands for mayor number of the release version, minor stands for minor number of the release version and HF for Hotfix, if needed (e.g. `1.2.HF01-b01`).

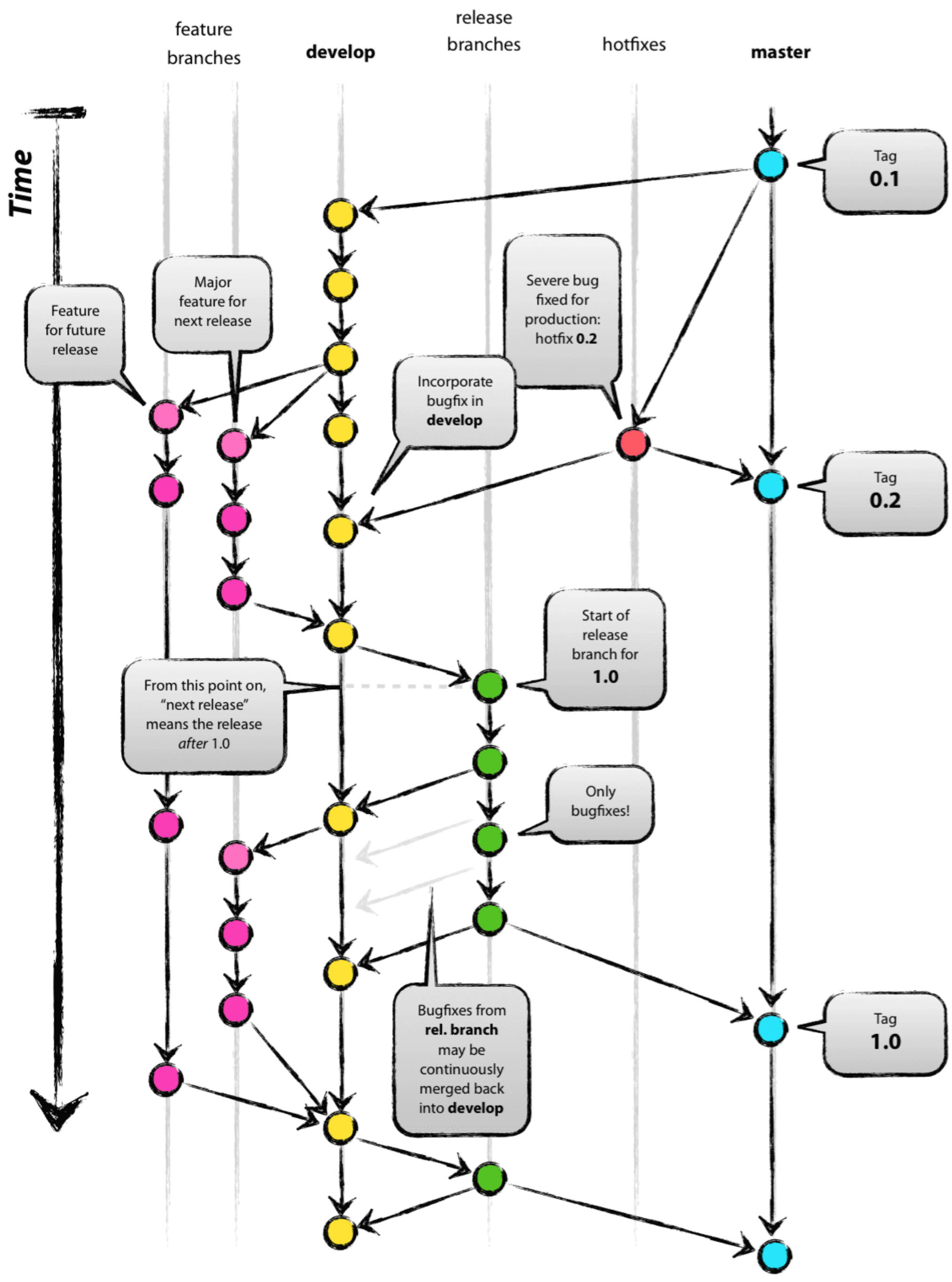
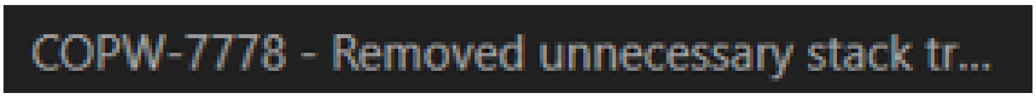

feature branches
develop
release branches
hotfixes
master
Time
Tag 0.1
Severe bug fixed for production: hotfix 0.2
Major feature for next release
Feature for future release
Incorporate bugfix in develop
Tag 0.2
Start of release branch for 1.0
From this point on, "next release" means the release after 1.0
Only bugfixes!
Bugfixes from rel. branch may be continuously merged back into develop
Tag 1.0

Figure 4.1: ES's full git model based on Gitflow workflow [37].

4.3 Commits

In ES, the commit structure does not follow well-known commit structures, such as semantic commits from semantic versioning. Commits on a project usually vary from developer to developer, but they should follow the pattern: `<JiraID/NameOfTheIssue> - <message>`.



COPW-7778 - Removed unnecessary stack tr...

Figure 4.2: Screenshot of commit from COP project, provided by ES project developer [38].

5 Migration

In migration, a developer's primary goal is to merge all the changes done between the old product version and the project into a new product version if there are differences (changes) between an old and a new product version. If there are no changes between product versions, file migration is unnecessary [39].

The developer's goal is only to identify all the potential problems and migrate them as quickly as possible. The final migrated project is not necessarily backward compatible; a hardware or software system that can successfully use interfaces and data from previous system versions [40].

At the beginning of the migration, a project developer should check the new product guideline provided by product developers. The guideline provides all the most significant changes made in the last version, which quickly gives into perspective where potential problems could lie. Next, a project developer must acquire relevant data from old and new product versions, such as POS-Server, UCON, SDC from Central, and OmniPOS. These parts can be found on the company's SVN, Git, or maven repository.

5.1 Files comparison

A developer makes all the files comparison through Total Commander's function dir-sync. The function produces changed files, which are checked manually. Dir-sync does not take inheritance as a factor. The developer has to examine every parent file if there is some.

Suffix	Explanation / usage
_EXT	the suffix is used on files that are extending another file, such as java classes or domain files using extends, or promo, promoj, module files using specializes.
_CST	in a case where no extension or specialization is implemented (e.g., implementation of an interceptor)
_CPY	when a workaround is needed, and a class is copied from the current product version and overridden. If functionality is available in the newly migrated product version, files are deleted.
_BRY	files usually extending `_CPY` file or overriding bean in Spring

Table 5.1: Project's files naming convention [41].

5.2 Migration steps

At first, a developer must migrate Central (Database, master data). The following task is to increase a product version in the main POM.XML file to respond with the intended new product version. The main pom file includes all the version variables such as *<version.dep.pos>, <version.dep.pos_server>* and *<version.dep.project_central>* inherited by minor project modules. After that, each module's clean phase must be completed before the module's install phase can begin by running `mvn clean install` command. A developer then refreshed each module in Eclipse and ran the project. Provided steps should get Eclipse up and running with the recently migrated libraries [39].

5.2.1 Compare and migrate

This subsection provides information about everything a developer must focus on in a project migration. It is divided into three parts by the fact if it is specific only for POS, Central or if it is for both [39].

Common

- **Java classes -** Check all the classes and merge the differences with particular attention to files `_EXT`, `_BRK`, `_CPY` suffix. Files with `_BRK` suffix usually extend product classes overriding bean in Spring; therefore, these changes are highly relevant and must be merged.

- **Spring and other XML files -** Overlapped XML files located under resources such as *web.xml*, and *bootstrap.xml* need to be checked, as well as each extended XML file in the project.
- **Tests -** After migration, it is vital to run all the unit tests and fix those which fail.
- **Database -** DB structure needs to be updated by building/updating central and setting fitting connection settings in OPOS. It is also required to update a master data database for OPOS by creating a pump on OPOS.
- It also needed to merge changes from other parts of a program such as *version descriptor, templates and patches, installer* and *data containers.*

Point of Sale (POS)

- **Promo/promoj files -** Extended promo/promoj files, primarily focusing on the files with the `_EXT` suffix.
- **XML layouts -** Check for any issues caused by XML layout hierarchy, such as iFrames. Examine the differences in the product between versions.
- **CSS -** Examine wheater any CSS change is able to affect layouts.
- **Modules -** A developer must check if modules newly included in the new product version are not overridden in a project.
- **DOIN -** Files with `_EXT` suffix must be replaced entirely from the new product version since they may include attributes. Files with `_CST` suffix must be checked for changes.

Central

- **UI5 -** Changing the versions in appropriate places is necessary since there could error in a release build.
- **Import mapping -** Merge is required from new product import mapping.

5.2.2 The final steps after migration

After migration, a developer should check pending bugs in the project and whether they have been resolved by migration in a new product version. As a result of the migration, technical debt records can be resolved, resulting in a smaller bundle size for the whole project. For example, a project workaround is not needed anymore since its functionality was implemented in a newer product version. Lastly, a developer should inform QA about any essential changes done during migration that they need to consider, and a migrated project is ready for release [39].

5.3 Migration process summary

Overall, project version migration is a very complex process, which is also highly time-consuming for the developer performing it. The manual file comparison with the use of Total Commander's tools is inefficient. Since it does not consider inheritance in both DSL and Java used in the development, it leaves room for a developer to overlook the source of a problem slowing down the migration in the procedure. The manual downloading of the product artifact primary placed on the company's maven repository is another ineffective part of the migration process.

With the current technology merging changes and resolving all migration conflicts have to be done by a developer. However, apart from that, downloading relevant data and identifying possibly damaged files have room for improvement. Therefore, Eurosoftware s.r.o. requested a solution facilitating the current migration process. The solution needs to be cable of performing the following tasks:

- Identifying all overwritten files in the project that was changed in the product between the new and old version.
- Identifying all *java, promo,* and *promoj* files that can be influenced by some change in the product.
- Checking layout *XML* files on specific paths.
- Identifying all product files that changed between versions.

6 Analysis

Compatibility is the ability of two systems to function together without being modified. A product's compatibility indicates that it satisfies some or all of the requirements for a certain standard [42]. An operating system, product, or technology's ability to work with an earlier legacy system is known as backward compatibility [40].

In the scope of the thesis, the project is compatible with the new product version when it can run on the new version without any malfunction. When the project can run on the newer version, it does not have to be compatible with the previous version. Therefore it is not backward compatible.

The solution's goal is primarily to point out product changes that could cause the project to malfunction, such as a change in a product file that is overwritten in the project or a change in a product file that is inherited in the project.

The ES's migration process optimization can be primarily divided into two separate tasks: downloading product artifacts for comparison and file comparison between a project and two product versions. Besides that, the program must be capable of parsing requested data from different files (DSL, Java) and reasonably present all program's findings.

6.1 File comparison

The application's primary task is to compare files. It has to be capable of finding changes in product versions that are preferably relevant to the project. For example, a project file inherits from a product file, which is changed in-between versions.

Multiple tools can perform a simple file comparison, such as Total Commander through the Dir-sync function, Beyond Compare Version 3 [43], Kaleidoscope [44], and many others.

There are multiple technologies to choose from for comparing files. Programming languages usually have solutions for comparing files available in accessible libraries that are already optimized, such as `FileUtils` class in Java with the `contentEquals` method [45]. Some languages have better performance (e.g., C++), but the performance difference between these already optimized processes is slight compared to the time spent downloading artifacts and, therefore, will not be considered a crucial factor for choosing the technology.

6.2 Acquiring product source code

Almost all product artifacts are located on Nexus including not only artifact binaries but also their source code.

A REST API is an application programming interface (API or web API) that sticks to the REST architectural style's rules allowing users to interact with RESTful web services [46]. Most modern programming languages, such as Java, C++, JavaScript, Bash script, and more, have a simple way to interact with an API. Nexus has REST API available [47], allowing easy artifact extraction. For example, a user can obtain artifacts using Bash script and cURL, a tool for transferring data using popular protocols such as HTTP [48], and then unpack the artifact with `unzip`.

As mentioned above, Nexus is working with Maven. Therefore, the most straightforward way to obtain data would be to use the already created **unpack** execution from the **maven dependency plugin** by Apache. This execution not only downloads, if necessary, selected artifacts but also unpacks them [49].

6.3 File parsing

It is critical to check whether a file has a superclass and obtain the file's location from the file's context to achieve the requirement of identifying all the files that can influence Java, promo, and promoj files. Because ESs, besides Java, uses their custom DSL in development, there is no available solution for this problem. A custom parsing system needs to be created that suits each of the used languages and is capable of finding a file's superclass and its location.

Searching and parsing text can be achieved by using regular expressions; patterns used to match character combinations in strings [50]. Read a file from top to bottom, save all the imports, and create the superclass's file path from the saved imports in the presence of a superclass.

6.4 Findings presentation

Presenting the program's findings in a user-friendly way is one of the main factors in terms of usability. A more sophisticated approach for logging findings could be by generating a web-page, which would show all the findings with a piece of more specific information upon expanding. This could be achieved through generating HTML, coding that organizes a web page's structure and content [51], and CSS for styling.

A straightforward approach could be creating log files through selected language available libraries and then log file paths of the potentially harmful files.

6.5 Integration into the development process

The last issue that needs to be solved before building an application solution is integrating the solution into the development process, which is highly connected to solution usage. As mentioned in ES's technologies chapter (§3.1.1), POS is divided into sub-models. Therefore, solutions need to be capable of working on separated sub-modules and the whole project.

The only reasonable way this can be achieved is through Maven.

6.6 Chosen technology

Presently, there is no tool capable of comparing files and finding changes valid for the project migration. Due to the product's extensive DSL usage, it is impossible to use the already created application and customize it. The only possible way is through a completely custom application. There are many possible languages to use. The most suitable solution is the maven plugin, which is easily integrated into the current ES's development process.

6.6.1 Maven plugin

Maven is nothing but a plugin execution framework in which the usage of plugins accomplishes every task.

Maven's key feature is plugins, which allow common build logic to be reused across different projects. They accomplish this by performing an activity in the context of a project's description - the Project Object Model (POM). A set of unique parameters exposed by a description of each plugin goal can be used to tailor plugin behavior (or Mojo). Maven, in a nutshell, is nothing but a plugin execution framework in which the usage of plugins accomplishes every task [52].

Maven plugins are primarily written in Java, allowing easy future development of the plugin in ES, where Java is a primary programming language.

A plugin can be configured separately for each sub-module enabling its execution for separate sub-modules and everywhere through parent POM. The plugin can also be isolated from the development using profiles. Profiles are defined using a subset of the POM's elements (plus one extra section), and they can be triggered in various ways. They alter the POM during the build process and are intended to be used in pairs to provide equivalent-but-different settings for diverse target environments. As a result, different profiles can easily result in different construction results [53] letting ES's developers set the plugin in a profile and run only the plugin during the clean install phase solving the integration into the development process problem.

Maven plugin is the only reasonable solution to solve all the problems provided in the chapter above, and it is a recommended solution by ES development team.

7 Application

For the reasons provided throughout the thesis, the maven plugin was created. The plugin is created in java 1.8. to be responsive to the usage of the current version in ES.

The main goal is to speed up the version migration process by pointing out product changes that could lead to project malfunction or unwanted behavior. This is done by three-way scanning and comparing the two products and the project.

The plugin can be executed on single or multiple modules at once based on the project's POM configuration. The correctness of the result is highly dependent on the configuration set correctly. The plugin can work on already downloaded files for comparison or download them by itself if they are not provided.

The main functionality is broken into executions, each concentrating on a different issue (§7.3). As a result, the plugin creates one log file for each execution, consisting of the file paths of the project file, which is affected, and the product file that is the cause.

7.1 Architecture

The source code is divided into three main packages **mojo, utils** and **parser**.

Mojo is a plain old Java Object in Maven. In Maven, each Mojo is an executable goal [54]. In the migration plugin, every Mojo is located in the **mojo** package and extending `AbstractExecution` class, which deals with all the product and project downloading, validating, and loading, making the plugin easy for adding future extensions.

As mentioned in the technologies chapter, the migration plugin must parse all ES languages correctly. The **parser** package includes all the parses which are used in different mojos. The parses are Java objects with a file as only constructor arguments, which can acquire via regular expression if a file has a superclass and its potential location (§7.2.3).

The **utils** package incorporates all the helper functions and objects used throughout the application. The most significant is `ModificationHelper` class, which is responsible for all the string alterations such as excludes from the configuration into regular expression format, the output string modification, and others. The `ModificationHelper` is also accountable for filtering files based on includes and excludes from the configuration and filtering by suffixes.

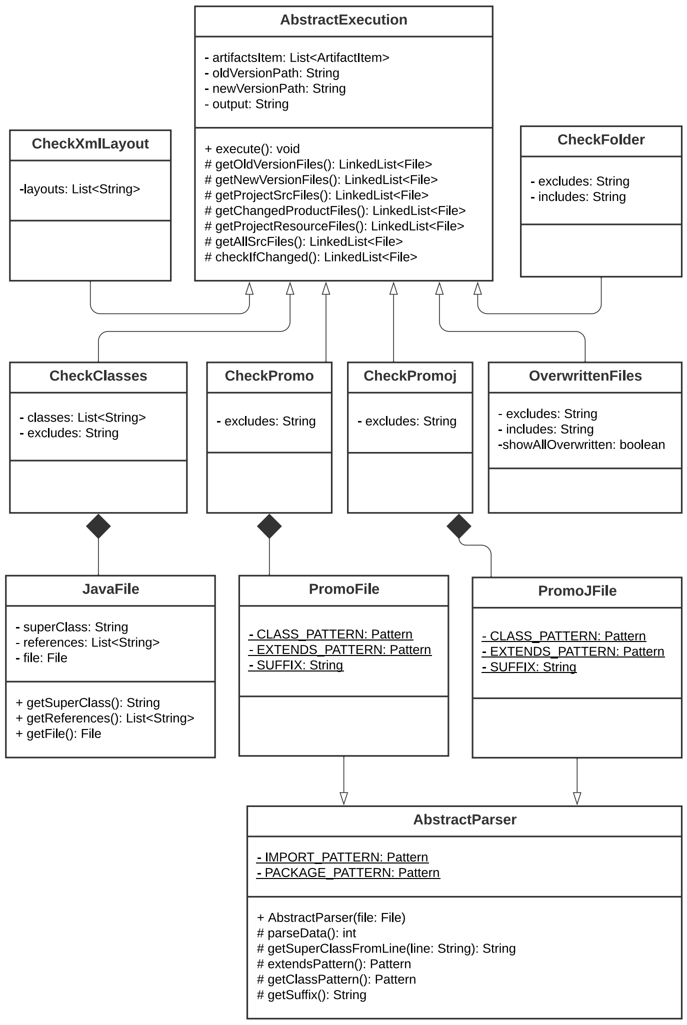

AbstractExecution
- artifactsItem: List<ArtifactItem>
- oldVersionPath: String
- newVersionPath: String
- output: String
+ execute(): void
# getOldVersionFiles(): LinkedList<File>
# getNewVersionFiles(): LinkedList<File>
# getProjectSrcFiles(): LinkedList<File>
# getChangedProductFiles(): LinkedList<File>
# getProjectResourceFiles(): LinkedList<File>
# getAllSrcFiles(): LinkedList<File>
# checkIfChanged(): LinkedList<File>
CheckXmlLayout
-layouts: List<String>
CheckFolder
- excludes: String
- includes: String
CheckClasses
- classes: List<String>
- excludes: String
CheckPromo
- excludes: String
CheckPromoj
- excludes: String
OverwrittenFiles
- excludes: String
- includes: String
-showAllOverwritten: boolean
JavaFile
- superClass: String
- references: List<String>
- file: File
+ getSuperClass(): String
+ getReferences(): List<String>
+ getFile(): File
PromoFile
- CLASS_PATTERN: Pattern
- EXTENDS_PATTERN: Pattern
- SUFFIX: String
PromoJFile
- CLASS_PATTERN: Pattern
- EXTENDS_PATTERN: Pattern
- SUFFIX: String
AbstractParser
- IMPORT_PATTERN: Pattern
- PACKAGE_PATTERN: Pattern
+ AbstractParser(file: File)
# parseData(): int
# getSuperClassFromLine(line: String): String
# extendsPattern(): Pattern
# getClassPattern(): Pattern
# getSuffix(): String

Figure 7.1: Migration plugin UML diagram.

7.2 Migration plugin's common workflow

7.2.1 Loading artifacts

The migration plugin can work with already downloaded files as well as download files during the process. At the beginning of the execution, the plugin checks if files are on *oldVersionPath* and *newVersionPath*; if not, the plugin examines configuration for artifacts items, which are then downloaded and unpacked using *maven-dependency-plugin* internally.

7.2.2 File comparison

Comparing files is a big part of the plugin. Since the project is using java as a programming language, the plugin file-to-file comparison is made through *FileUtils* class using the `contentEquels` method. Before comparing the contents byte by byte, the method checks whether the two files are of different lengths or if they point to the same file [55].

7.2.3 Working with files

The program loads single files using the Java stack class to avoid recursion, which could easily overload JVM on loading more extensive data.

Files are reloaded for each execution, insignificantly slowing the plugin's performance. This can be solved with a carrier, which must implement logic to remember current modules. In the case of running a plugin over multiple modules, the plugin keeps data in the carrier unchanged.

Some executions (e.g., Check classes (§7.3.1)) must also analyze parts of the code; in that case, a buffered reader is used for reading, and specific regular expressions are implemented for parsing and validating lines.

Parsers

All custom parsers used in the plugin are working on the same principle. During reading lines of a file, parsers save the package and all the imports from the files. The reading continues until the main header, which potentially contains information about the file's superclass, is found (e.g., public class extends superclass - in java). The path to a superclass is then created based on imports and packages.

A warning is printed out to notify a user when a superclass is not located in provided files.

Superclass is skipped during validation when it is still located in the project; therefore cannot damage the project during product version migration.

Specific regular expressions are displayed under executions.

7.3 Executions

Maven lifecycle phases are executed progressively from validation to deployment [56]. Each execution is executed during the install lifecycle phase. The install phase is second to last, ensuring all other plugins, which could potentially modify the project files, were executed before the migration plugin.

7.3.1 Check classes

The execution checks all product java files enhanced by annotation in the project or set in the configuration. The plugin also dynamically validates all superclasses (parents) that may affect the appointed file. The configuration also allows the user to set an excludes list of excluded java files separated by a comma.

Process

The Mojo collects all the java files from the project folder and all between versions changed product files. If there are no changed files, execution stops with the appropriate console log. Elsewhere execution continues by scanning already located product files for the reference annotation. Java files from annotations together with files located in the configuration provide the execution's focus group.

The application's custom parser parses selected files so that they would provide information about their potential superclass and its location. The superclasses are then validated in the same matter recursively.

The execution prints the number of possible errors found and the execution time into the console and creates an output file containing detailed information about the changed classes and file paths affected by the change. If any file in output was referenced using annotation, output also adds the path to the file where the annotation occurs.

Regular expression used for parsing

```
(?:public class)\s([a-zA-Z0-9_\-.]+)\s(?:extends)\s([a-zA-Z0-9_\-.]+)
```

Listing 7.1: Regex used for parsing and finding superclass in java files

```
(?:@ProductReference)([a-zA-Z0-9_\-.(), "]+)
```

Listing 7.2: Regex used for finding product reference annotation

```
(?:package )([a-zA-Z0-9_\-.,"]+)
```
Listing 7.3: Regex used for finding package

```
(?:import)\s([a-zA-Z0-9_\-.]+)
```
Listing 7.4: Regex used for finding imports

7.3.2 Check Promo and Promoj

Both promo and promoj currently have similar execution processes with only minor differences. The execution checks all product promo (promoj) files extended by any project promo (promoj) files and validates them among their superclasses. The user can also create an excludes list of files to be excluded, separated by a comma.

Process

The execution goes through all project files with the appropriate suffix (promo, promoj) and selects the ones with superclass. The plugin goes through the inheritance of the project promo (promoj) files, picking out all the product files and checking if there are any changes in those files between product versions, which could present possible errors in the initial file during migration. A file's superclasses are parsed from a file via the custom parser using regexp. Logged output file presents a potentially problematic product file with the name of the project file.

Regular expression used for parsing

```
(?:Process)\s([a-zA-Z0-9_\-.]+)\s(?:specializes)\s([a-zA-Z0-9_\-.]+)
```
Listing 7.5: Regex used for finding and parsing superclass

```
(?:Package )([a-zA-Z0-9_\-.,"]+)
```
Listing 7.6: Regex used for finding package

```
(?:Import)\\s([a-zA-Z0-9_\\-.]+)
```
Listing 7.7: Regex used for finding imports

7.3.3 Check XML layout

Check XML layout is the most straightforward execution in the plugin. Execution checks if XML files were changed in the products.

Process

The execution has a different configuration from the other files. By the client's requirements and since there are only a few XML files, it is not worth going through all the product files to find them. Therefore, there must be a full relative path to those files in configuration. The execution loads these files in both product versions and compares them, resulting in the output log file.

7.3.4 Check overwritten files

Sometimes, it is necessary to completely overwrite the original product file during the project development to ensure intended behavior. These files tend to be problematic during migration since the original file can change its conduct or even be deleted altogether. The check overwritten files execution locates these files and points out the questionable ones to the developer.

Process

In the beginning, the Mojo requires all project and old product files, filtering them by set includes (e.g., it *.java, *.promo) and excludes from the configuration. By comparing the relative file paths, the execution finds all the files that have been overwritten in the project. The plugin finds all the changed files on these relative paths between product versions. In the output, the plugin notifies a user which files have been altered during the migration or are missing altogether. If *showAllOverwritten* is set to true in the configuration plugin provides relative paths of all the overwritten files that have not been modified in any way in the migration.

7.3.5 Check folder

The check folder Mojo has an easy process with the primary goal of comparing product files/folders. The execution loads all the products files based on the includes (e.g., **.do, *.doin*) and excludes from the configuration, and compares them. The project files are not considered in this Mojo.

Process

The Mojo first obtains all the files from both product versions based on set includes and excludes. Then the files are loaded into hashtable with the relative path as a key and file object as a value. The execution iterates through the hashtable with old files and checks if there is a file on the same path in the new product version. The output of the execution consists of three parts. Changed that changed, files that were added, and files that were deleted in-between versions.

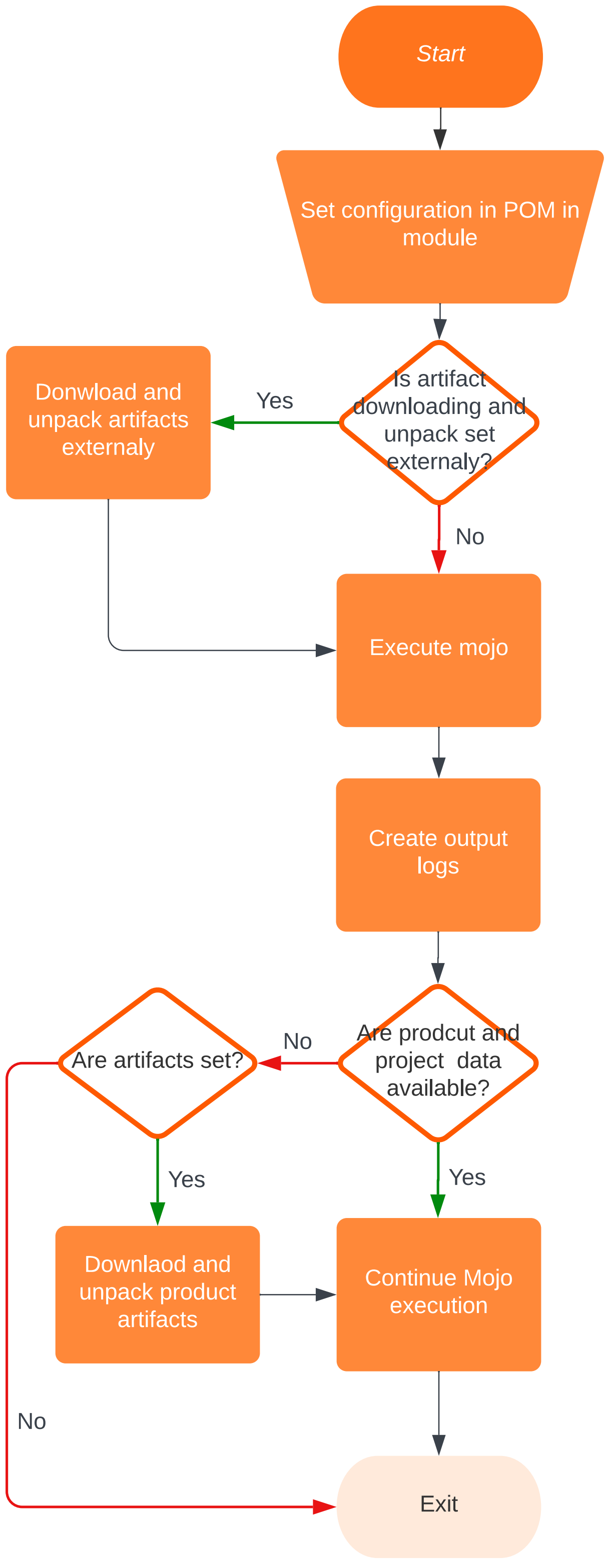
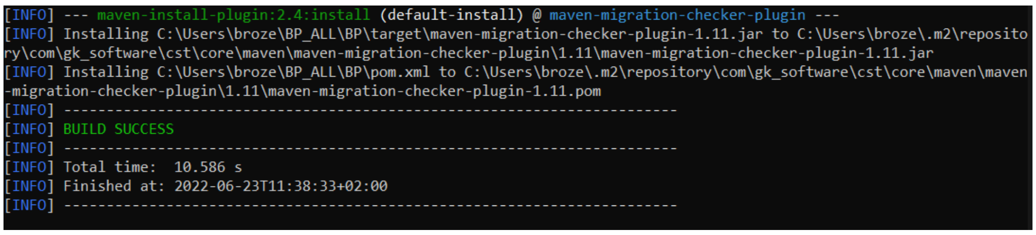

Start
Set configuration in POM in module
Is artifact downloading and unpack set externaly?
Yes
Donwload and unpack artifacts externaly
No
Execute mojo
Create output logs
Are prodcut and project data available?
No
Are artifacts set?
Yes
Yes
Downlaod and unpack product artifacts
Continue Mojo execution
No
Exit

Figure 7.2: The migration plugin's single execution flowchart.

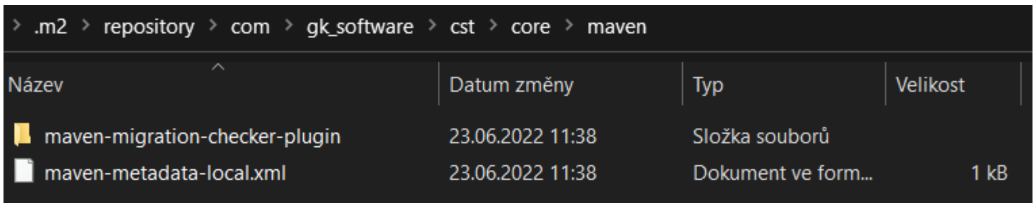
7.4 Launch

The maven plugin was built in Java 1.8 using Apache Maven 3.6.3. The application can be installed with the `mvn clean install` maven command over the root migration plugin directory.



```
[INFO] --- maven-install-plugin:2.4:install (default-install) @ maven-migration-checker-plugin ---
[INFO] Installing C:\Users\broze\BP_ALL\BP\target\maven-migration-checker-plugin-1.11.jar to C:\Users\broze\.m2\reposito
ry\com\gk_software\cst\core\maven\maven-migration-checker-plugin\1.11\maven-migration-checker-plugin-1.11.jar
[INFO] Installing C:\Users\broze\BP_ALL\BP\pom.xml to C:\Users\broze\.m2\repository\com\gk_software\cst\core\maven\maven
-migration-checker-plugin\1.11\maven-migration-checker-plugin-1.11.pom
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time:  10.586 s
[INFO] Finished at: 2022-06-23T11:38:33+02:00
[INFO] ------------------------------------------------------------------------
```

Figure 7.3: Screenshot of `mvn clean install` command execution over plugin's root directory.



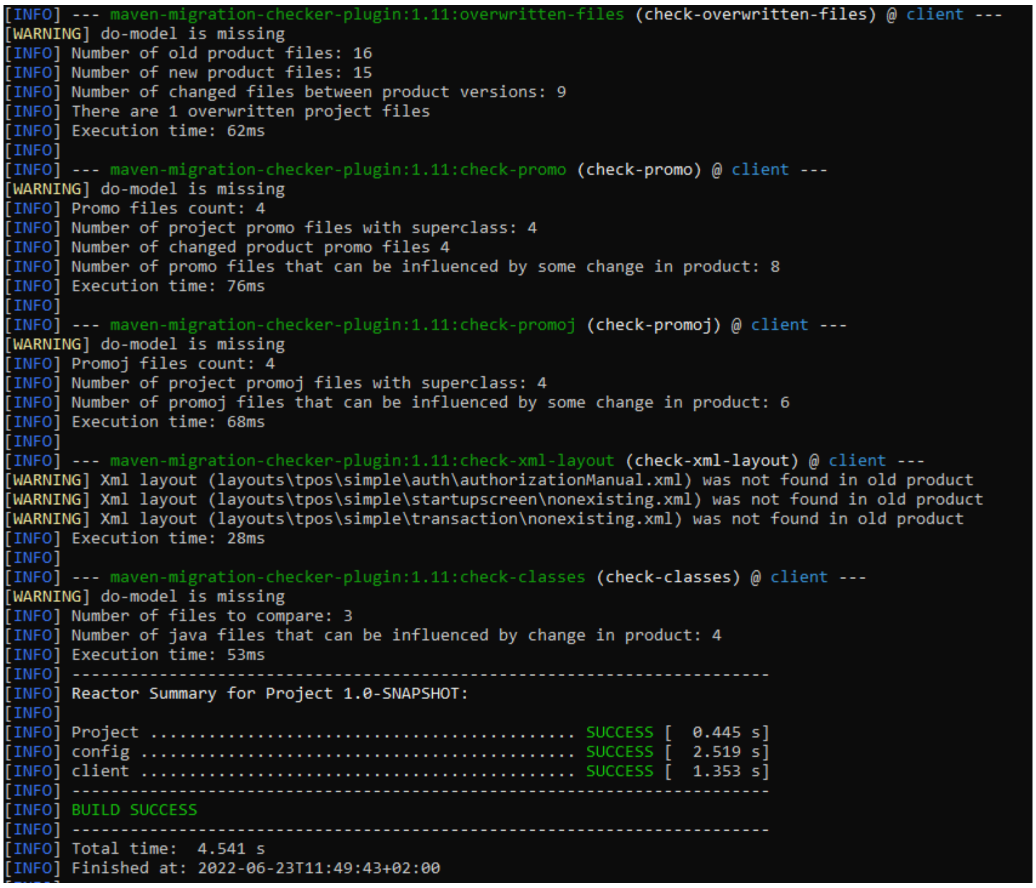
› .m2 › repository › com › gk_software › cst › core › maven

Název	Datum změny	Typ	Velikost
maven-migration-checker-plugin	23.06.2022 11:38	Složka souborů	
maven-metadata-local.xml	23.06.2022 11:38	Dokument ve form...	1 kB

Figure 7.4: Screenshot of added plugin in the maven repository after installing.

7.4.1 Sample plugin execution

Maven multi-module project with predefined profiles was added to the electronic appendencies to test plugin functionality. The plugin configuration is filled in the POM of the project's sub-modules. The execution is identical to the execution in the ES using `mvn clean install -Pmigration` over the parent module POM directory.



```
[INFO] --- maven-migration-checker-plugin:1.11:overwritten-files (check-overwritten-files) @ client ---
[WARNING] do-model is missing
[INFO] Number of old product files: 16
[INFO] Number of new product files: 15
[INFO] Number of changed files between product versions: 9
[INFO] There are 1 overwritten project files
[INFO] Execution time: 62ms
[INFO]
[INFO] --- maven-migration-checker-plugin:1.11:check-promo (check-promo) @ client ---
[WARNING] do-model is missing
[INFO] Promo files count: 4
[INFO] Number of project promo files with superclass: 4
[INFO] Number of changed product promo files 4
[INFO] Number of promo files that can be influenced by some change in product: 8
[INFO] Execution time: 76ms
[INFO]
[INFO] --- maven-migration-checker-plugin:1.11:check-promoj (check-promoj) @ client ---
[WARNING] do-model is missing
[INFO] Promoj files count: 4
[INFO] Number of project promoj files with superclass: 4
[INFO] Number of promoj files that can be influenced by some change in product: 6
[INFO] Execution time: 68ms
[INFO]
[INFO] --- maven-migration-checker-plugin:1.11:check-xml-layout (check-xml-layout) @ client ---
[WARNING] Xml layout (layouts\tpos\simple\auth\authorizationManual.xml) was not found in old product
[WARNING] Xml layout (layouts\tpos\simple\startupscreen\nonexisting.xml) was not found in old product
[WARNING] Xml layout (layouts\tpos\simple\transaction\nonexisting.xml) was not found in old product
[INFO] Execution time: 28ms
[INFO]
[INFO] --- maven-migration-checker-plugin:1.11:check-classes (check-classes) @ client ---
[WARNING] do-model is missing
[INFO] Number of files to compare: 3
[INFO] Number of java files that can be influenced by change in product: 4
[INFO] Execution time: 53ms
[INFO] ------------------------------------------------------------------------
[INFO] Reactor Summary for Project 1.0-SNAPSHOT:
[INFO]
[INFO] Project ............................................ SUCCESS [  0.445 s]
[INFO] config ............................................. SUCCESS [  2.519 s]
[INFO] client ............................................. SUCCESS [  1.353 s]
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time:  4.541 s
[INFO] Finished at: 2022-06-23T11:49:43+02:00
```

Figure 7.5: Screenshot of the plugin execution over the sample project.

8 Testing

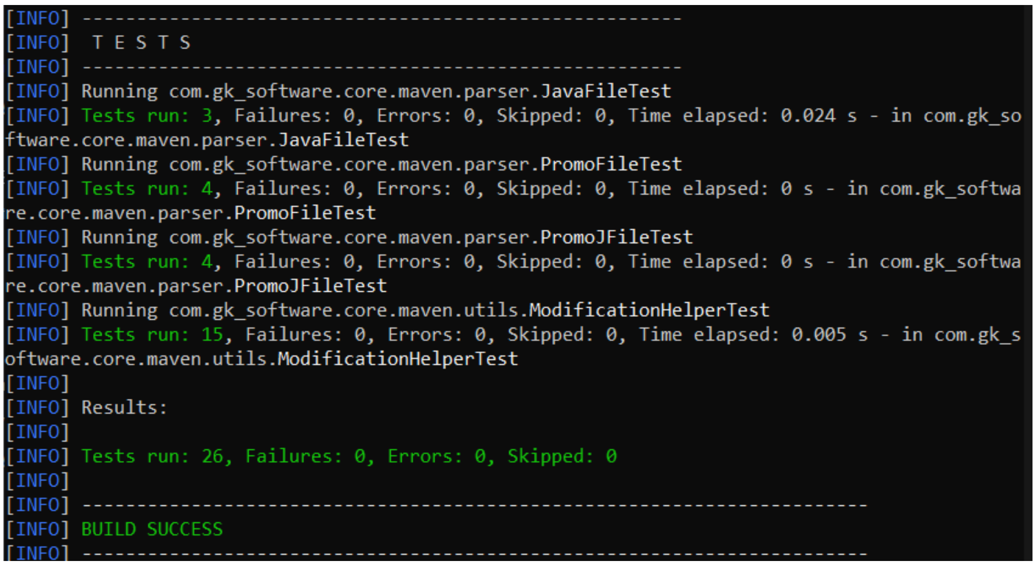
Each plugin's execution was tested to eliminate bugs during the development process. The plugin was tested through the unit and functional testing. In the begging mock data were used and gradually transferred to actual data from OmniPos COP migration.

After extensive functional testing, ES's developer also provided the plugin's functionality assessment.

8.1 Unit testing

Individual pieces of source code are tested as part of the software testing process known as unit testing to see if they are ready for usage [57, p. 75]. A unit in Object-oriented programming (OOP) is typically a whole interface, a class, or a single method.

The application uses **JUnit-5** [58] for unit testing. In the application, unit tests were issued for the *ModificationHelper* class, which method is used in every Mojo. Used parser classes, such as *JavaFile*, *PromoFile*, and *PromoJFile*, were also tested. All the mentioned classes are crucial for the application to run correctly.



```
[INFO] -------------------------------------------------------
[INFO]  T E S T S
[INFO] -------------------------------------------------------
[INFO] Running com.gk_software.core.maven.parser.JavaFileTest
[INFO] Tests run: 3, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.024 s - in com.gk_so
ftware.core.maven.parser.JavaFileTest
[INFO] Running com.gk_software.core.maven.parser.PromoFileTest
[INFO] Tests run: 4, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0 s - in com.gk_softwa
re.core.maven.parser.PromoFileTest
[INFO] Running com.gk_software.core.maven.parser.PromoJFileTest
[INFO] Tests run: 4, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0 s - in com.gk_softwa
re.core.maven.parser.PromoJFileTest
[INFO] Running com.gk_software.core.maven.utils.ModificationHelperTest
[INFO] Tests run: 15, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.005 s - in com.gk_s
oftware.core.maven.utils.ModificationHelperTest
[INFO]
[INFO] Results:
[INFO]
[INFO] Tests run: 26, Failures: 0, Errors: 0, Skipped: 0
[INFO]
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
```

Figure 8.1: Screenshot of the unit text execution through `mvn test` command on the plugin's version 1.11.

8.2 Functional testing

Functional testing is a type of black-box testing that is part of the Quality assurance (QA) process and bases its test cases on the requirements of the software component being tested. During functional testing, input is provided, and the results are examined and validated [59, p. 42].

This section presents each execution example usage on actual data from two COOP modules and a complete assessment of the migration plugin done by ES's project developer on OmniPos COP migration from product version *5.17.3-b01* to *5.19.6-b06*.

8.2.1 Mod-functions-client-cst module

Mod-functions-client-cst module is part of POS-Client's modules that test the plugin's internal download and unpacking ability, *overwritten* execution, *check-promo* execution, *check-promoj* execution, *check-xml-layout* execution, and *check-classes* execution.

The migration plugin's findings are briefly summarized in the tables below. Complete plugin execution, including defined profile and logs from executions, is provided in the electronic appendencies.

Description	Data
Located old product files	6195
Located new product files	6347
Number of changed files between product versions	561
Overwritten files influenced by product changes	2
Execution time	162069ms

Table 8.1: check-overwritten-files execution

Description	Data
Located project promo files	336
Project promo files with superclass	209
Changed product files	49
Promo files influenced by product changes	110
Execution time	3653ms

Table 8.2: check-promo execution

Description	Data
Located project promoj files	336
Project promo files with superclass	2059
Promoj files influenced by product changes	35
Execution time	3641ms

Table 8.3: check-promoj execution

Description	Data
Java files selected for checking	15
Java files influenced by product changes	19
Execution time	4551ms

Table 8.4: check-classes execution

Description	Data
Number of selected xml layouts for comparision	10
Number of xml layouts not found	1
Number of changed xml layouts	0
Execution time	7ms

Table 8.5: check-xml-layout execution

8.2.2 Mod-config-client-cst module

Mod-config-client-cst module is part of POS-Client's modules that test the plugin's external download and unpacking ability through *maven-dependency-plugin* unpack goal execution and *check-folder* execution.

The findings of the migrating plugin are summarized in the table below. The appendices provide complete plugin execution, including a defined profile and logs from executions.

Description	Data
Includes set	*.do,*.doj,*.doin
Located old product version files	694
Located new product version files	694
Number of deleted files between versions	0
Number of added files between versions	0
Number of changed files between versions	25
Execution time	18258ms

Table 8.6: check-folder execution

8.3 Overall usability evaluation

ES's developer issued the following points after the plugin assessment [60].

1. Good usability
   (a) check promo a promoj - very good (all changes marked and clear what you should check)
   (b) check-classes - very good - all annotated classes, which were changed were marked (but necessary to maintain this annotations - eg. during the review of new feature)
   (c) overridden classes - OK - after new changed it should show only changed overridden files
   (d) data containers - quite ok after some configuration (include only instances, which are used in project, exclude files, where changed only version - eg. descriptors) - there will be new update of plugin, which will determine also removed / added file
   (e) layouts - not changed during our migration
2. Usable with minor issues
   (a) installers - quite ok via check folder (mostly xml, properties), but necessary to check for example also pom, which is not possible via plugin
   (b) db (sql scripts) - quite ok, filtered only used db, which are used on given project, quite clear results
   (c) do/doj/doins - via check folder - changed file mentioned in output, but not so clear like - java/promo changes

3. Usable with bigger issues

    (a) UI5 - check folder changes available, but is very complicated to identified, what exactly relates to project changes - a lot of functions could be not related (overridden in project etc) - not so usable now

    (b) templates - for now not so usable - it will be good to implement new execution for example for check-patches, which will check project patches against changed file in product

    (c) beans - not sure if we want to use it here - currently not used

    (d) import mapping - can be used to identified change in some mfg file -> then has to be mapped in altova

4. Not usable now

    (a) pom - not possible now

    (b) app-container changes - theoretically it is possible to use check folder here, but don't know if it makes sense to solve it via migration tool. Mostly very specific changes. If we want to use it we need to add plugin into app-container artifacts

5. Does not make sense

    (a) tests - probably doesn't make sense - it is identified by failed build

    (b) technical debt - some workarounds because of GKRI, which should be fixed on new version - has to be mentioned else - e.g., confluence page

9 Conclusion

The primary goal of the thesis was to design and develop a solution for Eurosoftware s.r.o. that would simplify product version migration by pointing out product files, which could cause malfunction after the project is migrated from the version on which the project was built to the newer product version.

In the beginning, the thesis informs briefly about the company's product and present versioning system. However, the thesis primarily focuses on technologies used in ES's development and the current migration process.

In order to pick the most suitable solution, an analysis was made, examining available tools and popular programming languages such as Java, C++, Bash, and others. The analysis was divided into five sub-problems (acquiring product source code, file comparison, file parsing, findings presentation, and integration into the development process). Each sub-problem was explained and solved in multiple ways, resulting in the custom maven plugin as a chosen technology.

The final version of the migration plugin is easily expandable. The plugin can download and unpack files or work with already downloaded directories. It is capable of identifying overwritten files, checking XML layout, complete folder comparison, and complex identification of all Java, promo, and promoj files, which detects if files could be influenced by any change in product, including inheritance. Findings are presented simply in log files created by the plugin for each execution.

Unit testing and functional testing were issued to test application correctness. Functional testing was performed on mocked data as well as on actual project migration by the ES's developer.

The created application meets the ES's specification (§5.3) and has already been introduced in the company's development practice. After submitting the plugin in January, ES issued future development, which altered the presentation of the findings from log files into a dynamic creation of HTML files.

List of Acronyms

QA Quality assurance. 18, 28, 44

SCM Software configuration management. 21

SDC Store Device Controller. 10, 25

UCON Universal Connector. 11, 25

UI User Interface. 17

XML Extensible Markup Language. 6, 7, 11, 15, 17, 27, 28, 37, 38, 48

Bibliography

[1] G. S. SE, “GK Software SE.” [Online]. Available: https://www.gk-software.com/us/company/about-gk

[2] “Úvod.” [Online]. Available: https://www.eurosoftware.cz/#co-delame

[3] G. S. SE, “GK Software SE.” [Online]. Available: https://www.gk-software.com/us/solutions/omnipos

[4] M. Štrobl, “Migration plugin assessment on cop – migration plugin,” source placed in electronic appendencies. [Online]. Available: confluence/Big-Picture.png

[5] M. Strobl and S. Schwarz, “Gk software se’s services description,” source placed in electronic appendencies. [Online]. Available: confluence/services.png

[6] M. Strobl, “Pos - thin-pos variant,” source placed in electronic appendencies. [Online]. Available: confluence/thin-pos.png

[7] ——, “Pos - fat-pos variant,” source placed in electronic appendencies. [Online]. Available: confluence/Fat-pos.png

[8] ——, “Pos - smart-pos variant,” source placed in electronic appendencies. [Online]. Available: confluence/smart-pos.png

[9] ——, “Pos client functionality overview,” source placed in electronic appendencies. [Online]. Available: confluence/pos-client-funcionality.png

[10] “Maven – Welcome to Apache Maven.” [Online]. Available: https://maven.apache.org/

[11] D. Szczukocki, “Multi-Module Project with Maven | Baeldung,” Oct. 2018. [Online]. Available: https://www.baeldung.com/maven-multi-module

[12] S. Inc, “Nexus Repository Manager | Sonatype.” [Online]. Available: https://www.sonatype.com/products/nexus-repository

[13] Atlassian, “What is Jira Software used for?” [Online]. Available: https://www.atlassian.com/software/jira/guides/use-cases/what-is-jira-used-for

[14] “What is Java Swing? - Definition from Techopedia.” [Online]. Available: http://www.techopedia.com/definition/26102/java-swing

[15] “Jenkins.” [Online]. Available: https://www.jenkins.io/

[16] M. Strobl, “Pos language diagram,” source placed in electronic appendencies. [Online]. Available: confluence/language-diagram.png

[17] “The Java Language Environment.” [Online]. Available: https://www.oracle.com/java/technologies/introduction-to-java.html

[18] “Inheritance in Java,” Mar. 2017, section: Java. [Online]. Available: https://www.geeksforgeeks.org/inheritance-in-java/

[19] “XML introduction - XML: Extensible Markup Language | MDN.” [Online]. Available: https://developer.mozilla.org/en-US/docs/Web/XML/XML_introduction

[20] “CSS: Cascading Style Sheets | MDN.” [Online]. Available: https://developer.mozilla.org/en-US/docs/Web/CSS

[21] “What are Domain-Specific Languages (DSL) | MPS by JetBrains.” [Online]. Available: https://www.jetbrains.com/mps/concepts/domain-specific-languages/

[22] cartermp, “.NET Framework technologies unavailable on .NET Core and .NET 5+.” [Online]. Available: https://docs.microsoft.com/en-us/dotnet/core/porting/net-framework-tech-unavailable

[23] T. Kosar, P. E. Martınez López, P. A. Barrientos, and M. Mernik, “A preliminary study on various implementation approaches of domain-specific language,” *Information and Software Technology*, vol. 50, no. 5, pp. 390–405, Apr. 2008. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0950584907000419

[24] S. Herrmann and M. Mosconi, “Do dsl overview,” source placed in electronic appendencies. [Online]. Available: confluence/DO-DSL-overview.png

[25] M. Strobl, “Gk software se dsl overview,” source placed in electronic appendencies. [Online]. Available: confluence/DSL-overview.png

[26] S. Herrmann, "Gk software se - stakeholders," source placed in electronic appendencies. [Online]. Available: confluence/stakeholders.png

[27] ——, "Gk software se - common language," source placed in electronic appendencies. [Online]. Available: confluence/common-language.png

[28] ——, "Gk software se - promo and promoj inheritance syntax," source placed in electronic appendencies. [Online]. Available: confluence/promo-promoj-inheritance.png

[29] ——, "Gk software se - module dsl introduction," source placed in electronic appendencies. [Online]. Available: confluence/module-dsl.png

[30] R. S. Pressman, *Software engineering: a practitioner's approach*, 7th ed. New York: McGraw-Hill Higher Education, 2010, oCLC: ocn271105592.

[31] Atlassian, "Gitflow Workflow | Atlassian Git Tutorial." [Online]. Available: https://www.atlassian.com/git/tutorials/comparing-workflows/gitflow-workflow

[32] A. Grzesik, "GK Software SE's git naming convention.s," source placed in electronic appendencies. [Online]. Available: confluence/branch-naming-convention.png

[33] ——, "GK Software SE's git workflow," source placed in electronic appendencies. [Online]. Available: confluence/workflow.png

[34] ——, "GK Software SE's git feature branch," source placed in electronic appendencies. [Online]. Available: confluence/feature.png

[35] ——, "GK Software SE's git hotfix branch," source placed in electronic appendencies. [Online]. Available: confluence/hotfix.png

[36] ——, "GK Software SE's git release branch," source placed in electronic appendencies. [Online]. Available: confluence/release.png

[37] ——, "GK Software SE's git model," source placed in electronic appendencies. [Online]. Available: confluence/Git-model.png

[38] P. Kopal, "Eurosoftware's commit structure example from COP project in IDEA," source placed in electronic appendencies. [Online]. Available: confluence/commit-msg.png

[39] M. Štrobl, “Cop - coop (omnipos) – product migration,” source placed in electronic appendencies. [Online]. Available: confluence/Product-migration.pdf

[40] “What is Backward Compatible (Backward Compatibility)?” [Online]. Available: https://www.techtarget.com/whatis/definition/backward-compatible-backward-compatibility

[41] M. Štrobl, “Project naming conventions - GK project development rules,” source placed in electronic appendencies. [Online]. Available: confluence/extensions.png

[42] “What is Compatible?” [Online]. Available: https://www.computerhope.com/jargon/c/compatib.htm

[43] “Scooter Software: Home of Beyond Compare.” [Online]. Available: https://www.scootersoftware.com/features.php

[44] “Kaleidoscope.” [Online]. Available: https://kaleidoscope.app

[45] “FileUtils (Plexus Common Utilities 3.4.0 API).” [Online]. Available: https://codehaus-plexus.github.io/plexus-utils/apidocs/org/codehaus/plexus/util/FileUtils.html#contentEquals(java.io.File,java.io.File)

[46] “What is a REST API?” [Online]. Available: https://www.redhat.com/en/topics/api/what-is-a-rest-api

[47] “REST.” [Online]. Available: https://oss.sonatype.org/nexus-restlet1x-plugin/default/docs/rest.html

[48] C. Sabato, “Using cURL in a Bash Script: Get the Response Code from an API,” Mar. 2020. [Online]. Available: https://codefather.tech/blog/curl-bash-script/

[49] “Apache Maven Dependency Plugin – dependency:unpack.” [Online]. Available: https://maven.apache.org/plugins/maven-dependency-plugin/unpack-mojo.html

[50] “Regular expressions - JavaScript | MDN.” [Online]. Available: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Regular_Expressions

[51] "HTML basics - Learn web development | MDN." [Online]. Available: https://developer.mozilla.org/en-US/docs/Learn/Getting_started_with_the_web/HTML_basics

[52] "Maven – Introduction to Maven Plugin Development." [Online]. Available: https://maven.apache.org/guides/introduction/introduction-to-plugins.html

[53] "Maven – Introduction to build profiles." [Online]. Available: https://maven.apache.org/guides/introduction/introduction-to-profiles.html

[54] "Maven – Plugin Developers Centre." [Online]. Available: https://maven.apache.org/plugin-developers/

[55] "FileUtils (Apache Commons IO 2.11.0 API)." [Online]. Available: https://commons.apache.org/proper/commons-io/apidocs/org/apache/commons/io/FileUtils.html#contentEquals-java.io.File-java.io.File-

[56] "Maven – Introduction to the Build Lifecycle." [Online]. Available: https://maven.apache.org/guides/introduction/introduction-to-the-lifecycle.html

[57] A. Kolawa and D. Huizinga, *Automated Defect Prevention: Best Practices in Software Management.* Wiley-IEEE Computer Society Press, 2007.

[58] S. Bechtold, S. Brannen, J. Link, M. Merdes, M. Philipp, J. Rancourt, and C. Stein, "JUnit 5 User Guide." [Online]. Available: https://junit.org/junit5/docs/current/user-guide/

[59] C. Kaner, J. Falk, and H. Q. Nguyen, *Testing Computer Software*, 2nd ed. Wiley Computer Publishing, 1999.

[60] P. Kopal, "Migration plugin assessment on cop – migration plugin," source placed in electronic appendencies. [Online]. Available: confluence/COP-testing.png