

Aalto University
School of Science
Master's Programme in Computer, Communication and Information Sciences

Nikolai Denissov

Creating an educational plugin to support online programming learning

A case of IntelliJ IDEA plugin for A+ Learning Management System

Master's Thesis
Espoo, January 11, 2021

Supervisor: Professor Lauri Malmi
Advisor: Juha Sorva PhD (D.Sc. in Tech.)

Aalto University
 School of Science
 Master's Programme in Computer, Communication
 and Information Sciences

ABSTRACT OF
 MASTER'S THESIS

Author:	Nikolai Denissov		
Title:	Creating an educational plugin to support online programming learning A case of IntelliJ IDEA plugin for A+ Learning Management System		
Date:	January 11, 2021	Pages:	69
Major:	Software and Service Engineering	Code:	SCI3043
Supervisor:	Professor Lauri Malmi		
Advisor:	Juha Sorva PhD (D.Sc. in Tech.)		
<p>Educational plugins are extensively utilized in both, the industry and academia. Academically, the plugins are used to optimize certain labor-intensive and error-prone manual operations for students, such as loading, installing, completing and submitting the assignments. The thesis sets out to explore the features and functionalities of different educational plugins by developing the plugin to support a programming course at the Aalto University. As follows, the thesis goes through a thorough process of pitching, developing, implementing, reporting feedback and resulting feedforward. The process is facilitated by a three-member development team with the author of this thesis acting as a hands-on Scrum Master. The stakeholders' requirements of stability and fault-tolerance were met in an IntelliJ IDEA plugin, written in Java 8 and Scala. The Eclipse-based IDE was replaced with IntelliJ as the former was discontinued in 2018. Java 8 was chosen over Kotlin as none of the team-members are Kotlin-fluent and as Java 11 was not yet released at the start of the development. Also, as the plugin interacts with Scala components, a minor part of the application is written in Scala. The plugin is designed to interoperate with the existing backend learning management system A+ as a third-party data provider. Dependencies pruned during the process include IntelliJ Platform's compatibility with future IDE versions (solved through regular adjustments by the developers), Gradle IntelliJ plugin's compatibility with the Platform, standard Java libraries (solved through an automated vulnerability scanning), etc. The project is ongoing, with testing routinely performed with the stakeholder and student feedback fed back into the development process. However, future improvements include reworking the architecture, extending the testing scope, and multi-course support.</p>			
Keywords:	CS1, educational plugin, teaching tools, IntelliJ Platform		
Language:	English		

Contents

Special Terms & Abbreviations	5
1 Introduction	8
2 Background	10
2.1 Development environments in CS1	10
2.2 Modern toolset	12
2.2.1 Automated assessment	12
2.2.2 MOOC platforms	14
2.2.3 The role of the IDEs	15
2.3 Plugins, plugins are everywhere	16
2.3.1 Intro to plugins	17
2.3.2 Plugins' internals	17
2.3.3 IntelliJ Platform and plugins	19
2.3.4 Plugins in teaching and research	21
3 Current state analysis	22
3.1 Teaching programming at the Aalto University	23
3.2 Scala	24
3.3 Prerequisites for creating software	26
3.4 Choosing the IntelliJ IDEA	27
3.5 Overview of analogical solutions	27
4 Methods and environments	30
4.1 Software development methods	30
4.2 Development environment	32
4.3 Requirements	35
5 Solution	37
5.1 Architecture	38
5.2 Integration with the Platform	40

5.3	Integration with the external systems	41
5.4	Dependencies	42
5.5	Approach to UI/UX	44
5.6	Testing & quality	45
5.7	Delivery and operations jump-start	46
6	Discussion	48
6.1	Meeting requirements	48
6.2	Technology choice and the learning curve	49
6.3	Documentation and versioning	50
6.4	Architecture	51
6.5	Integration	52
6.6	Approach to UI/UX	53
6.7	Testing & quality	53
6.8	Operations	54
7	Conclusion	55
7.1	Closure	55
7.2	Future developments	56
8	Appendices	63
8.1	Appendix 1. Semi-structured interview questions	63
8.2	Appendix 2. Technologies and tools	64
8.3	Appendix 3. Scala Plugin repo branches	65
8.4	Appendix 4. A+ Courses plugin UI/UX	66
8.5	Appendix 5. Student feedback	68
8.6	Appendix 6. Final feedback	69

Special Terms & Abbreviations

.jar Java ARchive

accessibility practice of making websites usable by as many people as possible, regardless of disability type or severity of impairment

Amazon Web Services a part of Amazon providing on-demand cloud computing platforms and Application Programming Interfaces (APIs)

API Application Programming Interface

CI/CD Continuous Integration/ Continuous Deployment

COVID-19 a global respiratory virus-caused pandemic that started in Asia in 2019

CS0 a common name for orientation programming courses

CS1 a common name for introductory programming courses

CSE Computer Science Education

DI Dependency Injection

dockerized from "Docker"—a set of Platform as a Service (PaaS) products that use OS-level virtualization to deliver software in packages called containers¹

Emacs an ancient, highly customizable technology used for editing text and code on UNIX systems

EPFL *École Polytechnique Fédérale de Lausanne*

¹docker.com

FJ Fork—Join model

freemium a marketing strategy that offers basic service features for free and additional ones if paid

hook from "hooking"—a set of techniques to augment the behavior of a software system

HTTP Hypertext Transfer Protocol

IDE Intelligent Development Environment

IM Instant Messenger

JVM Java Virtual Machine

linter a static code analysis tool used to flag programming errors, bugs, stylistic mistakes, and suspicious constructs

LMS Learning Management System

LOC Line of Code

LTl Learning Tools Interoperability Specification

MIT Massachusetts Institute of Technology

MOOC Massive Open Online Course

MVP Model—View—Presenter

OOP Object-Oriented Programming

OpenAPI a part of the IntelliJ Platform API that helps to handle its internal state

OSS Open-Source Software

PaaS Platform as a Service

PoC Proof of Concept

PR pull request, in distributed Version Control System (VCS), a technique for a developer to notify the team members of a completed feature

REPL Read–Eval–Print Loop

REST Representational State Transfer

SaaS Software as a Service

SDK Software Development Kit

TA Teaching Assistant

TDD Test-Driven Development

TMC TestMyCode Platform

UI User Interface

UI/UX User Interface/ User Experience

UNIX a family of multitasking, multiuser computer operating systems

VCS Version Control System

VFS Virtual File System

Vim (from a family of vi-derived tools) a free and often default text editor on UNIX systems

WCMS (Web) Enterprise Content Management Systems—a larger group of software targeting various business functions: document management, internal and external corporate portals, websites

Web World Wide Web, commonly known as the Web

XML Extensible Markup Language

Chapter 1

Introduction

Multiple introductory programming courses are taught at the Aalto University Computer Science department. The courses vary in terms of the programming languages used and utilize slightly different teaching approaches. Many of the novice-level courses in the Computer Science at the Aalto are arranged as Massive Open Online Courses (MOOCs). Since recently, the staff of one of them, namely Programming 1 (*Ohjelmointi 1* or simply O1), have decided to update supporting tools used for the course. The Eclipse¹-based Scala Intelligent Development Environment (IDE) was swapped for IntelliJ IDEA², which also has rich feature-support for the Scala programming language and affiliated technologies. Course teachers have already considered creating an IDE plugin. The change of IDE, however, has triggered them to set up a small development team to create the educational plugin. The goal of the software was to interoperate with the existing backend A+ Learning Management System (LMS)³ and optimize or even eliminate certain labor-intensive or otherwise error-prone manual operations for the students. Using an IDE plugin to support teaching and student troubleshooting is quite a novel approach for the Aalto University School of Science.

The thesis aims to demonstrate a way to implement a plugin to meet the University course context requirements best. At this point, the plugin intends to support one course only with the potential of extending to others. The work mostly covers technical implementation details and challenges met, providing thus insights into the development processes and environments used. Additionally, the thesis highlights the impact on the learning process for the students participating in O1. The plugin has been developed during 2020 under the Aalto-LeTech group in a team of three student-developers. The author of this thesis acted as a hands-on Scrum Master.

¹eclipse.org

²jetbrains.com/idea

³apluslms.github.io

The thesis has a spillover structure, starting with a conceptual/contextual introduction, following with implementing and applying the concepts and the context at the Aalto University. It begins with an introductory Chapter 1, giving key rationales for creating a plugin and briefly explaining the context, followed by Chapter 2 (Background), describing the approaches to teaching used during CS1 courses and highlighting recent trends in the field with a further focus on tools (for automated assessment, MOOC platforms, and IDEs) affecting CS1 teaching/learning, giving a historical perspective on the development of these tools within the academic environment and the industry. Lastly, the chapter defines plugins and explains common approaches to building them for IDEs.

Chapter 3 explains Current State Analysis of the present situation in the CS education at Aalto. It includes in-depth stakeholder-interview-based explanations about prerequisites to create the software (plugin) and approaches to teaching programming at the Aalto University, justifies the selection of Scala as a programming language and explores resulting technical choices in terms of supporting tools (IDEs). Finally, the chapter presents a quick overview of other relevant educational plugins used in the field (both in academia and the industry).

Chapters 4-6 describe the practical aspects of the solution. Chapter 4 (Methods) describes the processes, environments, and requirements used throughout the plugin development phase. Chapter 5 (Solution) introduces the created solution, namely IntelliJ IDEA plugin and its details: architecture, implementation technologies and quality metrics. Chapter 6 (Discussion) reports on the team challenges during the software development stage, including practical aspects like overall learning curve, component selection, inter-dependencies, documentation and other real-file constraints. The chapter explains why the resulting software is the way it is and evaluates the project's success level. The final chapter of the thesis concludes by summarizing fundamental discoveries, suggesting further research directions.

Chapter 2

Background

The chapter aims to introduce some historical and recent developments in the introductory Computer Science Education (CSE). The narrative takes a programming environment perspective and highlights the most relevant trends. Additionally, it briefly presents a modern toolset to support the CS1 education. Finally, the section describes a theoretical viewpoint on plugins as candidates for a place in the Computer Science educators' toolbox. The chapter's primary methodological approach is a literature review seasoned with the author's prior experiences.

2.1 Development environments in CS1

The research in the CSE has been evolving during the last fifty years, swiftly changing focus and introducing new approaches. While the debate about programming languages and paradigms dominated the research in the early years, the trend, having gone through multiple iterations (Pascal, C-likes¹, etc.) [from personal communication with prof. L. Malmi in 2020] - seems to have settled by now. During the last two decades, Java Virtual Machine (JVM)-based languages and Python have had the most attention among the CSE teachers. As follows, the discussion about the teaching approaches and ways to structure and design the CS1 courses has also diminished. [12]

Other trends from the 1990-s (likely related to the boost in the development of the Web) reflected the changing CSE landscape and a market in need for more programmers. This demand opened a niche for tools and services that would accelerate students' maturity and facilitate their entry into the job market. Trends included a deeper focus on the tools for the course material delivery (MOOC platforms) and more sophisticated development

¹oreilly.com/C-style-languages

environments (IDEs). These changes have also given a fresh start to the concept of automated assessment directly and/or through the development of educational IDE plugins. [12, 31, p. 20]

Another CS1 research area focused on better explaining the programs operation principles [12]. The area concentrated on debugging and visualization tools. Both subgroups of utensils targeted solving similar challenges faced by the CS1, e.g. problems in understanding the underlying processes of the computers, related common misconceptions, difficulties with tracing the state of the program and spotting the buggy parts of the code [32, 45, pp. 85-86].

The thesis mainly focuses on the programming environments. Moreover, the author intentionally narrows the scope further down, focusing on the non-visual part of these, the IDEs (in contrast to visual development environments, which try to explain the deeper process internals). Furthermore, the recent steps in the evolution of the programming environments are also covered.

Nowadays, a set of long-standing used technologies dominate the software development scene. The positions of Java and Python still hold strong with JavaScript climbing the tips and tops of the rankings [49]. While getting to the peaks, Java and Python have correspondingly affected the teaching-related studies and research. Historically, as Java played an important role in the industry and academia, numerous teams tried to create tools to support its learning. Those involved multiple open-source platforms for teaching basic programming, for example, BlueJ² and Eclipse with educational plugins on-board. [17]

Professional IDEs such as default flavour of Eclipse (as opposed to heavily customized versions for platforms-specific software development like Liferay Studio³) or used in the educational process and mentioned in this work IntelliJ IDEA and NetBeans⁴, are quite complicated tools. Initiatives like BlackBox⁵, WebCat⁶, TestMyCode⁷, and few others were launched to address the complexity for novice developers, facilitating automated assessment and collecting insights about the learning process [17]. This need to tackle the perceived complexity of the professional IDEs has also been emphasized by Reis & Cartwright [40].

The author of the thesis suggests that the effect of directing the entry-level students into using professional IDEs is versatile. Conceding to the complexity of the tools, he suggests that this can be dealt with by learning

²bluej.org

³help.liferay.com/liferay-studio

⁴netbeans.org

⁵bluej.org/blackbox

⁶web-cat.github.io/Web-CAT

⁷testmycode.github.io

the tools' configuration and customization methods. Another aspect that advocates for the usage of such IDEs is them combining industry best practices and approaches out-of-the-box. Additionally, the development teams rarely have resources to maintain, document and improve the tools, especially if compared to the industrially-supported open-source or commercial readily-available instruments.

The challenge of programming development environments as too complex for the beginners has also been addressed by the major IDEs developer JetBrains⁸. Their latest product portfolio contains IntelliJ IDEA Edu⁹ (for Java-based tech) and PyCharm Edu¹⁰ targeted at the novice developers and even containing some extended, built-in documentation with code-snippet examples.

The latest trends in CS1, however, have put more focus on students in trying to understand how to help them improve the performance [12]. The research shows that additional intervention of the course staff into the process (in the form of the CS0 courses with an emphasis on media computation or deeper collaboration between the students) provides a significant positive effect [53]. No doubt that the teaching does and will matter in the future. The modern development environments, however, provide APIs and other special techniques with ways to analyze and collect educational data [16, 54], opening thus a path to intelligent (more automated, and also potentially algorithm-based) tutoring of the students.

2.2 Modern toolset

This section provides a more profound immersion into the tools that support the primary programming education. It includes a description of trends in automated assessment and their relation to the extensive growth of the MOOCs. Moreover, the section offers an overview of several massive online learning platforms. The subchapter's final part describes the most popular IDEs and their importance for the modern software development practices.

2.2.1 Automated assessment

With time, the need for assessment in the CSE grew from simple quizzes to sophisticated tools for evaluating practical programming tasks. There is no doubt that a genuinely massive course delivery is hardly possible without the automated assessment (assuming the feedback is provided). The List 2.1 is

⁸jetbrains.com

⁹jetbrains.com/idea-edu

¹⁰jetbrains.com/pycharm-edu

presented below to better understand the success factors and challenges of the approach.

Success factors:

- high-quality assignments
- unambiguous task definition
- well chosen test data suite
- comprehensive feedback
- plentiful submissions
- students' experience with Test-Driven Development (TDD)
- additional support

Challenges:

- challenging effort
- increased effort
- suppressed creativity
- plagiarism
- higher skills expectations
- execution environment security
- limited assessment capability

List 2.1: Success factors and challenges in automated assessment [37].

In order to succeed, an automated assessment must offer students high-quality well-formulated tasks. Additionally, the approach is characterized by a need for a comprehensive feedback. The latter requirement arises from the fact that there are multiple solutions to each problem in software engineering. Meeting the formal styling, performance or test compliance requirements, does not explicitly suggest the best quality solution. So, despite the objectivity and efficiency of such evaluation methods, the lack of feedback and additional support might negatively affect the students' motivation and creativity [10, 37]. In the real-life, issues like that are commonly resolved with the Continuous Integration/ Continuous Deployment (CI/CD) pipelines. As follows, the human-feedback is provided as additional comments to pull requests following the successful completion of the static analysis tools (e.g. linters, complexity checkers, and others).

Recent trends in the field go beyond the *post factum* assessment of the LMS-provided static feedback, focusing on the ways to create more dynamic, *ad hoc*, learning environments. This is achieved through the usage of Web-based IDEs such as Amazon Web Services' Cloud9¹¹ or Software as a Service (SaaS)/PaaS solutions, e.g. jsFiddle¹² and CodePen¹³ [9]. Alternatively, the learning process-related data for upgrading the systems to more dynamic can

¹¹aws.amazon.com/cloud9

¹²jsfiddle.net

¹³codepen.io

be extracted directly from the locally installed (professional) IDEs [54].

Browser-based development environments reduce entry-barriers and let the students focus on the learning programming. However, from the practical perspective, shielding them from the real-world constraints of installing, using, and learning to configure the development tools might shape an incorrect perception of a practicing programmer's daily work. One clear benefit of the remotely hosted environments is the outreach it gives students to participate regardless of their computing capacity, which in turn, increases the access equality.

2.2.2 MOOC platforms

This subsection provides a lightweight comparison-like overview of several MOOC platforms. The selection is based on the author's professional opinion and learning experience. The comparison contains two platforms from each group: developed in Finland (mooc.fi¹⁴, A+) and outside (Coursera¹⁵, edX¹⁶). By definition, the MOOC platforms are online services, often of an asynchronous nature. Offered courses are not usually limited to the CS education. Table 2.1 highlights the key features of common MOOC platforms. The author of the thesis has cherry-picked observation parameters based on his professional experience, highlighting the systems' key characteristics. The first four rows represent general qualities of the MOOC platform as a product. The rest of the table demonstrates its links to the educational context.

As seen from the Table 2.1, by default, the platforms are available globally, and are thus limited to the localization languages only. The Finnish-developed platforms contain the local language and English courses alike, while global players like Coursera and edX offer more language options. All the analyzed services originate from academic environments. They were either created in a single university or are affiliated with a university consortium (Harvard, Massachusetts Institute of Technology (MIT), Stanford are among the most well known). Platforms offer various complexity courses either for free or using the freemium subscription model. The researched platforms depend heavily on automated assessment techniques, with some, like Coursera and A+ LMS (via third-party LTI integration) providing web-IDE-like tools for specific scenarios. In contrast to the browser-based IDEs, some service providers rely on plugins and extensions for existing, third-party development tools (Coursera, A+ LMS, mooc.fi).

¹⁴mooc.fi

¹⁵coursera.org

¹⁶mooc.org

		Coursera	edX	mooc.fi	A+
1.	globally available	✓	✓	✓	✓
2.	provide multilingual support	✓	✓	✓	✓
3.	commercial product	✓	✓	✗	✗
4.	contains free(mium) study courses	✓	✓	✓	✓
5.	university course offer-based	✓	✓	✓	✓
6.	contains entry-level courses	✓	✓	✓	✓
7.	purely computer science-oriented	✗	✓	✓	✓
8.	uses automatic assessment	✓	✓	✓	✓
9.	provides web-IDE-like environments to use	✓	✗	✓	✗
10.	offers plugins for locally installed IDEs	✓	✗	✓	✓

Table 2.1: Comparison of the MOOCs platforms features. Compiled based on [8, 11, 15, multiple events of personal communication with M. Riekkinen in 2020].

2.2.3 The role of the IDEs

Text editors first evolved into the code editors and later into the full-featured IDEs. For example, tools like Vim¹⁷ and Emacs¹⁸, - with certain effort and skill can be customized into the IDE-like environments. However, few users have attempted it. As follows, the author focuses on more mainstream products on the market. Modern industry demand for an IDE is versatile. Some of the key desired features are: code editor with syntax highlighting, refactoring capability and autocompletion, fast built-in search and indexing, VCS, debugger (with visual elements), easy customization, integrated language frameworks and build tools, and extensibility through plugins [14, 39]. Such a rich set of features available out-of-the-box reserves the IDEs a reliable place in the professional developers' toolboxes. The latest features gaining popularity in the developers' community are focusing on the deeper collaboration through the live-, pair- or mob-programming [18, 35]. As the global situation with COVID-19 remains challenging, there are reasons to believe the trend will continue.

The list of the most popular IDEs is topped by the Visual Studio¹⁹ (Code) family of tools, followed by the products of JetBrains. Lightweight editors

¹⁷vim.org

¹⁸gnu.org/emacs

¹⁹visualstudio.microsoft.com

like Sublime Text²⁰ and Atom²¹ also claim a significant part of the usage, acting also as a secondary/auxiliary tool. Finally, Eclipse, despite the controversial image in the industry, still retains a strong position in the professional developers' community.

This list correlates with the overall popularity of programming languages in the CS1 (see subsection 2.1) and the industry. For example, Visual Studio Code²² known to suit well for JavaScript and Python developers heads the list. It is followed by amazing products created at the JetBrains, namely, IntelliJ IDEA and PyCharm²³ [48]. As seen, the popularity of certain tools depends heavily on the languages targeted and technologies applied. For example, IntelliJ IDEA usage among the Java developers amounts to 72% and even higher for the Scala ones, totalling at—94% [21, 25]. It is probably due to the level of support provided by the Scala plugin for IntelliJ IDEA²⁴ (or faults in the other available tools). Such a high market-share of the JetBrains' products among the JVM-technologies is likely to be conditioned by favouring Java over other technologies.

There have been multiple efforts of various success to develop educational development environments in academia. None of them made it to the mass-market. Creating and popularizing such a tool requires a significant amount of resources, whether human or economic. Also, the IDE vendors are trying to meet the demand by introducing simplified or even education-targeted versions of their products.

In practice, most of the recently developed CS1 courses in Finland's major universities (e.g., Helsinki University, Aalto University, and several in FITEch University network) utilize existing professional IDEs, rather than develop *ad hoc* solutions. Admittedly, these provide richer functionality and remove the otherwise duplicate effort of learning the tooling for the students.

2.3 Plugins, plugins are everywhere

*Rakkaalla lapsella on monta
nimeä.*

an old Nordic proverb

Plugin (plug-in), addon (add-on, addin, add-in), extension, app, hook—are all different, while conceptually same names for a component of a software

²⁰sublimetext.com

²¹atom.io

²²code.visualstudio.com

²³jetbrains.com/pycharm

²⁴plugins.jetbrains.com/scala

system or service that enhances the existing or introduces new functionality; hence is unable to operate independently. The author will use these terms throughout the thesis interchangeably. In the context of this work, only the software components that extend other software systems are considered. This reduction aims to focus the thesis on the interaction between the hosting software and the plugin instead of examining different software-OS communication mechanisms (that use similar terminology). For example, native Linux libraries and Android apps (sometimes addressed as libraries, packages or gadgets, and widgets) are excluded from the analysis.

2.3.1 Intro to plugins

Plugin development is considered to be one of the successful product development strategies. It allows large multinationals like Google, Microsoft, Adobe, mid-sized companies such as Mozilla, Slack, and many more to involve third-party software development agents in product development, reducing thereby any related risks. Further, it simplifies the addition of new features. It is achieved by keeping the API of the product cleaner, reducing the size and maintenance burden of the application core.

Most notable (numerous) examples of the plugin approach implementation include Web Browsers, WCMS systems, IDEs and text editors. Those usually involve vendor marketplaces as a means of delivering and allowing to install plugins into a locally run or a cloud-deployed instance (e.g., Jira²⁵).

2.3.2 Plugins' internals

One of the architectures that support vital non-functional requirements of the modern software systems, namely its flexibility and extensibility, originated from a Pluggable Factory pattern [55]. It was developed in the late 1990-s when design patterns rocked the software development stage. [29]

Microkernel architecture is an architectural pattern where one or multiple components extend the functionality of a larger (often monolithic) core. Another interchangeably used name for the pattern is Plug-in architecture. However, despite the conceptual and naming similarity, it has little to do with *kernel* as it is addressed in the field of the OS-development. Commonly, the template is employed by larger, locally installed systems. Sound feature operation and isolation are among the essential benefits of this approach [41].

As shown in Figure 2.1, the general idea of the pattern is to seamlessly add the new functionality to an existing core of the application or service. It

²⁵marketplace.atlassian.com

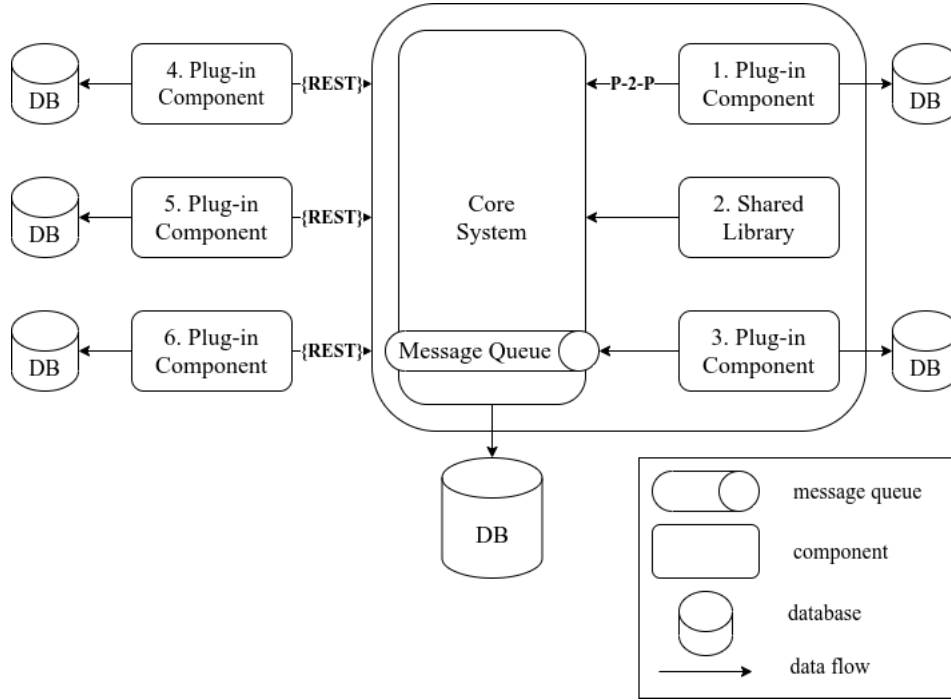


Figure 2.1: Diagram representing Plug-in architecture implementation [42].

can be achieved by creating either a plug-in component or a shared library (labeled as 1 (one) and 2 (two) on the Figure 2.1 correspondingly). These can communicate with the core system utilizing standard point-to-point extensions (for example, components One and Two) or existing messaging queue (component Three) for locally installed plugins. In case of the distributed systems Representational State Transfer (REST) API endpoints can be used for data exchange as shown for Plugin Four. It is common for the components to have dedicated persistent storage (marked as "DB" on the Figure 2.1). The core system can then be then internally layered, either technically or domain-based. [42]

There are three main steps of the plugin life-cycle. The first one is "plugging" and defined as:

An essential principle of a plug-in architecture is that system extensions are carried out in a controlled, restricted, and determined manner [56].

In practice, this means that external components interact with the core system via predefined interfaces, commonly known as extension points. These extension points, however, can be bypassed via access control mechanisms of

the implementation language if the platform developers have ignored encapsulation and security considerations. The next step—“deployment” assumes an existence of safe and reliable means of running the application on the platform; ideally without a need to stop or restart it. The final stage of the process is (automatic) component discovery. It is usually based on some configuration setting or is completely convention-based. [56]

The benefits of using a plug-in architecture are agility (flexibility), relatively easy deployment, smooth testing, and simplicity resulting in overall low development costs. In contrast, the downsides are low scalability (due to the core system’s nature), lacking elasticity and modest fault-tolerance. [41, 42] Additionally, handling the platform API versioning and accessing the underlying dependencies is problematic as the consensus between the change frequency and system stability must be preserved.

One curious case of complexity of maintaining an adequate level of interoperability between the plugins is a story of portlet specification²⁶. Portlets are smaller Java-applications (plugins) that branched from the servlets in the early 2000-s. In contrast to the web servers, where each application runs in its own namespace, all portlets run in the same environment, application-inside-the-application in its simplest. The noble idea behind implementing a common standard was to have portlets compatible between the WCMS portals (Oracle’s WebCenter Portal²⁷, IBM’s WebSphere Portal²⁸, Liferay²⁹ and others). Unfortunately, due to the differences in the specification’s implementation, in practice, plugins developed for one platform were unusable on another.

In 2007, D. Birsan predicted that pure plug-in architecture might become *a new big thing* in the industry [13]. However, nowadays, it’s clear that the battle was lost to the microservice architectures.

2.3.3 IntelliJ Platform and plugins

IntelliJ Platform is an Apache 2.0-licensed Open-Source Software (OSS) system created by JetBrains for building IDEs and language-aware developer tools. The flagship product and the most popular IDE for JVM-likes—IntelliJ IDEA derives from it. However, the platform is not limited to tools targeting Java Virtual Machine-based technologies. Another unique product from the same family is Android Studio, developed for Google in 2014. [20, 26]

The Platform’s main components are a Virtual File System, a Text Ed-

²⁶oracle.com/portlet-specification

²⁷oracle.com/webcenter-portal

²⁸hcltechsw.com/digital-experience

²⁹liferay.com

itor, a UI framework, a Debugger and a Test Runner [20]. Certain parts of the platform are made available through an OpenAPI—a set of grouped API endpoints that form a foundation for building features described in Subchapter 2.2.3. When writing this thesis, the IntelliJ Platform plugin repository contained close to a total of 6 000 plugins.

Source code of the platform and the many of plugins’ is hosted on the GitHub³⁰ or is otherwise available publicly. Third-party development efforts are supported through the Developer Documentation pages, specialized forums, and Instant Messenger (IM) channels (Slack³¹, Gitter³²).

IntelliJ Platform is a highly-modular system containing multiple extensible plugins. It is also responsible for handling the Dependency Injection (DI) when needed. IntelliJ Platform plugins can introduce various intrusion levels ranging from renaming and coloring the UI elements to providing a whole syntax-support for custom programming languages. As the platform itself is (mostly) Java-based, the plugins must be made with JVM-compatible languages like Java, Kotlin³³, Scala, or else. [26]

- Actions—a cornerstone Platform concept and a standard way to invoke the functionality of the plugin, requires custom implementation;
- Extensions—an advanced approach for interacting with the deeper-than-actions parts (for example Platform internal services);
- Services—singleton plugin services, can be scoped to the whole application, project or module;
- Listeners—a way to connect and listen on the Platform Message Bus;
- Extension points—a mechanism to propose plugin’s own functionality to extend to other plugins;
- Configuration file—an Extensible Markup Language (XML) configuration file that lists all the previously mentioned components and also some generic metadata of the plugin;

List 2.2: Key components of the IntelliJ Platform [23].

On the higher level, the IntelliJ plugin is a Java application that follows the general structure of a typical Java ARchive (.jar) file. Each plugin uses a different classloader to allow for plugins using different versions of the

³⁰github.com/intellij-community

³¹slack.com/intellij-platform

³²gitter.im/intellij-scala

³³kotlinlang.org

same libraries they depend on [22]. The main components of the plugin architecture are described in the List 2.2.

2.3.4 Plugins in teaching and research

Creating and maintaining an IDE for educational purposes is challenging. As follows, multiple educators have opted for plugins rather than alternatives [17]. The range of plugin use-cases in teaching and research is wide: from code and tests auto-generation and tracking students' keystrokes to learning patterns analysis and even teaching new languages like Scala [33, 47, 52, 54]. The latter example evolved into an Eclipse-based Scala IDE bundle. It's also curious to note that some of the educational tools themselves were transformed into plugins for the professional IDEs e.g. happened to BlueJ [36].

Chapter 3

Current state analysis

The given chapter describes current state of affairs at the Aalto University CS1 education. It explains what, how, and why programming is being taught based on semi-structured interviews with some key people responsible for the courses. The selection of the interviewees is made according to their organizational roles and involvement in the previous or current educational plugin development (shown in the List 3.1):

- prof. Lauri Malmi is heading the LeTech Group and has a long-term involvement in improving programming and online education at the Aalto University;
- Senior University Lecturer Juha Sorva, the main person behind the "crown-jewel" course Programming 1;
- University Lecturer Otto Seppälä, who is in charge of MOOC part of the Programming 1 course and also various next-level Scala-based courses at Aalto;
- Senior University Lecturer Arto Hellas has recently moved from the University of Helsinki where he led team responsible for the development of TestMyCode and its plugins;

List 3.1: Semi-structured interviews respondents.

The list of questions used to ignite the discussion with respondents is presented in the Appendix 8.1. The author's idea was to seek a consensus rather than demonstrate variation in opinions. The interview results are synthesized into a generic text without explicitly highlighting the authorship. This step is taken due to a small sample and in an attempt to anonymize the

content. The author is thus trying to avoid quoting directly, as individual opinions are not essential.

The chapter also contains brief explanations and rationales for creating the plugin. A separate part of the chapter is dedicated to present the Scala programming language. Finally, some preselected educational plugins from academia and the industry are presented to give a broader overview.

3.1 Teaching programming at the Aalto University

The Aalto University CS1 courses divide into two primary tracks: Scala and Python-based. The set of courses teaching Scala is considered more demanding and aiming at the CS students. The average number of unique participants for the Scala courses stream is approximately 1 000 students yearly. Python-based teaching is provided "as a service" for other departments like Engineering or Business. The average total headcount is close to 1 500 each year. [from personal communication with prof. L. Malmi in 2020] Most of the courses rely on automated assessment. Overall, the Aalto University has a long track of using such tools in teaching [28]. This thesis focuses on the courses being taught with the Scala: Programming 1¹ & 2², Programming Studio 1³ and 2⁴.

Discussing strong and weak sides of the CS1 at the Aalto University, respondents identified qualified personnel, robust pedagogical approach to the course design, early adoption of the automated assessment and bright-minded students as strengths of teaching novice-level courses. While discussing more specific strengths, fine quality learning materials were mentioned in the first place. In terms of the completeness, the contents of the courses like Programming 1 are characterized as being comprehensive, challenging, and intense. When discussing the complexity of the Scala-based branch of the novice-level studies at Aalto, multiple respondents emphasized them as being challenging or even more demanding than in most of the Finnish Universities. The ability to keep up and develop course materials is considered an additional benefit of the CS1 at Aalto. Interviewees agree that having multiple courses utilizing the same programming language is convenient and helps facilitating a continuous educational process. However, some notice that the interplay between these introductory programming courses could be tighter, shaping one bigger picture.

¹oodi.aalto.fi/programming-1

²oodi.aalto.fi/programming-2

³oodi.aalto.fi/programming-studio-1

⁴oodi.aalto.fi/programming-studio-2

Among the challenges of teaching the introductory programming courses, the respondents brought issues common for organizing any large-scale courses (e.g. limited direct guiding and high, unevenly distributed workload). Another factor mentioned, was a low level of cooperation between the classes/teachers behind the courses that shape a study track. Some of the problems, like the scale and load (along with overall staffing related matters), are possibly solvable by adding more qualified personnel, e.g. Teaching Assistants (TAs) with longer-term commitment and/or spare teachers for the courses. Some other challenges are harder to tackle. For example, better cooperation between the teachers might be hard to achieve. Multiple factors are in play here, including teachers' high level of freedom, their ambitions, different teaching methods, program design approaches and the curriculum structure.

While discussing the CS education's future at Aalto, the opinions divided into two main camps: organizational-educational and technological. Among the prospective organizational matters, further development of the courses and overall learning experience improvement were mentioned. Besides, respondents indicated deeper personification of the materials in the form of adaptive/intelligent learning and more socially-oriented tasks as a potential development area. Technical changes that the CS1 might face are the possible drive towards the cloud-environments (like cloud-based IDEs), broader integration between learning materials and the tools and introduction of additional engines to provide for visualization of program internals.

Overall, despite the issues and challenges that CS1 education at the Aalto University faces, the respondents agreed that it ranks highly in many aspects and has a bright future.

3.2 Scala

The official website of Scala provides the following definition:

Scala combines object-oriented and functional programming in one concise, high-level language. Scala's static types help avoid bugs in complex applications, and its JVM and JavaScript runtimes allow building high-performance systems with easy access to huge ecosystems of libraries. [57]

Initially Scala was developed by the team led by M. Odersky in *École Polytechnique Fédérale de Lausanne* (EPFL) in 2004. Nowadays it is being maintained by the Swiss-based Scala Center⁵, LightBend⁶ (a company behind

⁵scala.epfl.ch

⁶lightbend.com

the most popular Scala frameworks; namely Akka⁷ and built on top of it Play⁸) and a strong community. Another famous framework designed for working with the cluster-computing (big data) made with Scala is Apache Spark⁹. Further, Scala technologies are used by multiple tech giants and financial sector companies like LinkedIn¹⁰ Verizon¹¹, Zalando¹², PayPal¹³, UniCredit¹⁴, Barclays¹⁵, etc. [30].

The promise is that by the end of 2020, Scala will undergo a long-awaited release of the next major version, 3.0. This joint effort of academia, industry, and individual community developers will introduce new language features and tools and an improved performance [58].

There are specific reasons why Scala is chosen as a first language for teaching the CS1 branch courses at the Aalto University. First, it is a multiparadigm language that combines features of functional (declarative), Object-Oriented Programming (OOP) (imperative) and sequential programming [interview with J. Sorva 2020]. Other factors affecting the choice are: "strong" type system, high level of abstraction and clean concepts. Also, Scala was considered attractive from the pedagogical standpoint due to having a Read–Eval–Print Loop (REPL), language's ability to implement many helpful programming patterns and relevant documentation. Moreover, teachers' predisposition towards Scala played an important role. Usage of Scala, however, is not very common for teaching novice-level courses with only few universities choosing it over alternatives, including EPFL, Lund University, Trinity University and few others [interview with O. Seppälä, 2020]. [38, 46]

The programming development environments currently utilize two main approaches to handling Scala in the IDEs. The idea of the first approach is to "simply" apply all the language features and syntax-support into a plugin to work with the IDE platform. This approach has been used earlier by Eclipse, and for now, IntelliJ IDEA remains the only production-grade solution utilizing it. The second approach is based on the concept of the language server. The core idea stands in having a server that accepts the code to be checked/compiled or otherwise processed, returning the results via some standardized protocol. The approach seems to be more manageable for the developers of the IDEs and text editors. In this scenario, they do not have to dive into the language features deeply. The developers mostly need to provide implementation to handle the standardized output of the language server (Metals¹⁶ for Scala).

⁷akka.io

⁸playframework.com

⁹spark.apache.org

¹⁰linkedin.com

¹¹verizon.com

¹²zalando.com

¹³paypal.com

¹⁴unicreditgroup.eu

¹⁵barclays.co.uk

¹⁶scalameta.org/metals

The overall perspective language server approach is still emerging and somehow unstable. The method has a multidirectional effect. On one side, it devalues earlier efforts of the IDE developers in creating support for the language/ technology by providing easier access to the market for new tools' developers (companies) that otherwise would not invest. On the other hand, this approach introduces a potential dependency on a third party; which is not always acceptable for commercial companies. Additionally, Scala's development has not been very quick lately. Consequently, if this pattern applies to the language server, finding the users becomes a challenge in itself.

3.3 Prerequisites for creating software

Since the beginning of the Programming 1, the course has utilized the (Eclipse) Scala IDE which has not been updated regularly. As it was becoming obsolete, the decision to switch to another development environment was made. The step implied much painstaking work updating the course materials to become compatible with the new chosen IDE. It also provided an opportunity to introduce a plugin supporting many programming course related actions. An additional factor suggesting the need for a plugin was a more complex process of the module/project creation within the newly chosen IDE (IntelliJ IDEA) compared to the Scala IDE. The developed plugin's goal is to provide students with a smoother experience, shifting the focus on the learning and spending time and effort in a more efficient manner.

In practice, it means a simplified process of jump-starting the course, easier assignment submissions, and results viewing. From the students' perspective, the repetitive non-value-adding operations could be grouped as follows: assignment modules installation and maintenance, and course project configuration and management; but also, REPL customization and assignments submission (these groups of requirements are described in more detail in the List 4.2 of the Chapter 4.3).

Before beginning the development of the A+ Courses plugin, the stakeholders had almost none practical experience using plugins for teaching. On the higher level, having a working plugin with basic functionality opens exciting opportunities for further improvements, be it educational data collection or further simplification of the student learning experience.

3.4 Choosing the IntelliJ IDEA

The last release of the Scala IDE (or Eclipse IDE bundled with Scala plugin) occurred in 2018, which refers to a discontinued development of the product [44]. This fact by itself makes the need for an IDE change evident. The list of the Scala language-supporting IDEs recommended by the creators included multiple known tools: IntelliJ IDEA, Visual Studio Code, traditional Vim, and Emacs along with Atom and Sublime Text [57]. It is important to note that the list excluded Eclipse as irrelevant. As such, the main mass-market IDE candidates for the course were, therefore: IntelliJ IDEA with active support for Scala (support for the not-yet-released Scala v.3 "Dotty" [27], 18 million plugin downloads and the latest stable version published about a month ago) and Visual Studio Code + Scala (Metals) (support via language server, 88 thousand downloads, the latest release made approximately a month ago) [24, 34]. Already at this point, the choice gravitates towards the IntelliJ IDEA. It has a wider spread and correlates with the course staff's plan to teach the students real industry-grade tools. Additionally, when choosing the IDE to use, Scala Metals implementation of the language server technology was still emerging. It could not be considered fully production-ready yet due to instability and relatively narrow feature set [43].

Further analysis of additional factors - e.g. the usage of specific tools in the Scala developers' community and the level of language support makes the choice of the IntelliJ IDEA as a new IDE for the Aalto University Scala-based programming courses even more logical. The maturity and the extensibility of the Platform also played an important role. On the other hand, IntelliJ IDEA remains the only tool that implements language features without using a Metals language server.

3.5 Overview of analogical solutions

This section describes and compares vital features for several notable educational plugins. The first of them—EduTools¹⁷ is developed by JetBrains. The rest of the plugins have been developed at Helsinki University to interoperate with the TestMyCode Platform (TMC) MOOC platform. The older of them TMC NetBeans¹⁸, as the name implies, has been developed against the NetBeans Platform. The newer one TMC IntelliJ¹⁹ was created as a student project to work with the IntelliJ IDEA.

¹⁷github.com/edu-tools

¹⁸github.com/tmc-netbeans

¹⁹github.com/tmc-intellij

The analyzed plugins differ in the effort invested and product perspective taken. Thus, in the author’s opinion, together, they portray a good set of functionality. In addition, some of their features acted as a source of inspiration for the A+ Courses plugin. The evaluated features are shown in the Table 3.1. Comparison criteria are chosen in a manner that represents the broadest spectrum of the known educational plugin features. The criteria were selected based on the author’s professional expertise as a Software Engineer. The idea behind the functionality comparison is to show the commonalities between the educational plugins developed in different contexts.

		JB EduTools	TMC IntelliJ	TMC NetBeans
1.	support for multiple educational platforms	✓	✗	✗
2.	support for multiple IDEs ²⁰	✓	✗	✗
3.	support for multiple courses	✓	✓	✓
4.	programming language agnostic	✓	✗	✗
5.	developed in academia	✗	✓	✓
6.	provides a framework to create lessons ²¹	✓	✗	✗
7.	handles assignment submissions	✓	✓	✓
8.	installs and unpacks assignments	✓	✓	✓
9.	performs data collection	✗	✓	✓
10.	provides tools for running tests locally	✓	✓	✓
11.	OSS	✓	✓	✓
12.	actively supported ²²	✓	✗	✗

Table 3.1: Comparison of educational plugin features [19, 50, 51].

As seen in the Table 3.1, plugins have different support levels for educational platforms. Naturally, commercial products (such as EduTools) get more usage when they do not focus on a single platform. In contrast, the University-developed software should mostly adhere to the requirements of a single backend LMS. As a software developed by JetBrains, EduTools naturally supports a wide range of multiple IntelliJ Platform-based IDEs. All the compared plugins avail teaching multiple courses. Plugins vary in terms of supported programming languages. The TMC plugins focus on JVM-likes,

²⁰in the same products’ family

²¹for example ebook integration

²²last commit before September 2020

whereas EduTools can be considered language-agnostic. An outstanding feature of the JetBrains' tool is a custom course development framework it provides. The feature includes a special syntax for managing courses, assignment types, code, and tests. Academic research needs were one of the goals why TMC plugins were created in the first place. As such, plugins include functionality to collect student progress data. The data is being gathered in order to streamline the learning process. All the tools provide mechanisms for running assignment-related tests locally. All three plugins are released under the permissive open-source licenses (Apache 2.0²³, MIT²⁴, GNU GPL 2.0²⁵). Finally, the only project being actively developed currently is EduTools. Thus other mentioned plugins might still be in use. [19, 50, 51, multiple events of personal communication with A. Hellas in 2020]

²³apache.org/licenses/LICENSE-2.0

²⁴opensource.org/licenses/MIT

²⁵gnu.org/licenses/gpl-2.0

Chapter 4

Methods and environments

The given chapter describes the main methods and approaches used while developing the solution. Firstly, it focuses on the software development methodologies used throughout the process. Secondly, the development environment is presented in detail. Finally, hard and soft requirements for creating the software are defined and explained.

4.1 Software development methods

The author of the thesis is a certified Scrum Master with prior practical experience. This fact has directly affected the choice of the development methodology for the project. Also, an iterative approach with a higher level of transparency was favored by the stakeholders. It is natural to wish for deeper insights into the team's work at the beginning of the project, especially so in the student teams. So, although most of the business requirements were provided for the team in advance, the iterative approach was chosen *ad hoc*.

For a team of three, as was the case in the plugin development team, pure Scrum is oversized. The number of meetings and other joint activities will cut away the significant amount of time from the actual hands-on work. Additionally, as the team was co-located and had a high level of morale and motivation, it was always easy to spawn a meeting or a discussion when needed. To fit the team's reality, Scrum was slightly adapted. Apart from the short standard daily sessions, the weekly meetings (lasting about half an hour) with the stakeholders were held. This move allowed a) to ensure a high level of transparency for the stakeholders and b) provide a dedicated place to ask feedback, clarifications, or questions for the team. Often, Proof of Concept (PoC) solutions or implementation prototypes were demoed during these weekly meetings. Overall, the development team has been privileged

by having a source of fast and comprehensive feedback at hand (or, just over the corridor, before the COVID-19 limitations appeared). Otherwise, the team has been following the stages of a typical Scrum process. After every three-weeks-long Sprint, there has been a joint "Scrum event" meeting. In most cases, it included an extensive demo and feedback part, a retrospective with the stakeholders, the team's internal retrospective followed by a combined refinement and planning part. Sometimes the team has held additional planning or testing sessions.

On a higher level, the development process stages are described in the List 4.1.

- Design stage—when the team has analyzed the requirements, including both functional and general non-functional ones, agreed on the implementation details and overall fit into the architecture;
- Implementation phase—the actual hands-on programming part, mostly developing individual larger features. Initially, the team has learned to divide those larger features into smaller chunks to be worked on in parallel and only then merged. This approach allowed faster delivery of the complete features;
- Verification and testing was done on several levels: a) static analysis employing various tools (presented in the Subchapter 4.2), including validation of security and legal aspects, also styling and quality, b) testing of the software using the unit and integration tests, c) manual testing of the developed components and features;

List 4.1: Higher-level steps of the implemented development process.

Additionally, the team documented the process extensively, mostly as PRs comments and GitHub issues' (tasks) descriptions. Apart from that, the team actively used IM and video-conferencing tools for communication.

Finally, the working process description would not be complete without mentioning several tricks, introduced following the COVID-19 pandemic. As the isolation got stricter during Spring 2020, the team has started to face challenges related to some individual developers' dropping performance. This difficulty has been addressed by introducing more communication into the team's work. Joint debugging sessions and also weekly face-to-face meetings were started. This step has helped the crew maintain a good spirit and hold out until the restrictions got looser.

4.2 Development environment

The best-practices standard conditioned the development environment’s tools choice for OSS projects. Platform limitations mostly predefined the available technological decisions. It is common for plugin development that the Platform defines/often softly dictates such matters. Figure 4.1 shows a word cloud of the most notable technologies and tools used.



Figure 4.1: Word cloud representing the technologies and tools used in the project (for complete list see Appendix 2.).

Java 8 was preferred over Kotlin as a language of choice for the project. Selecting Kotlin could have harmed productivity because it would be a completely new language for all the team members to get acquainted with (while Java is not). Otherwise, developers would still have to learn Java to a certain extent, as the largest part of the core Platform remains in Java 8 and prior. Even if Kotlin provides a more fluent programming API, evolving faster with JetBrains creating new components of the IntelliJ Platform with it. Java 11 was deselected as at the beginning of the plugin project because the Platform’s official support for it was not yet released. As the plugin interacts with some Scala plugin components, to ensure seamless integration, a minor part of the application is written in Scala (v. 2.12).

JUnit was a team’s pick for a unit-testing framework. It is a well-known library and is also being actively utilized by the IntelliJ Platform. To add some additional capabilities, Mockito¹ and PowerMock² were introduced into the testing process later. For integration testing REST-assured³ library was used.

¹mockito.org

²github.com/powermock

³rest-assured.io

The usage of GitHub was very natural for the project. The team had employed GitHub's VCS, issue tracking feature⁴ and wiki⁵, and tried package repository (later discontinued due to the lack of need). Travis CI⁶ was used as the web-based CI/CD tool that is easy to integrate with GitHub's ecosystem. The described approach to tooling allowed the team to build and jump-start a cost-efficient (free, actually) development environment quickly. The additional validation functionality was provided by: Fossa⁷ (as OSS dependency license manager), Snyk⁸ (as vulnerability scanner), CheckStyle⁹ (to evaluate Java and Scala code styling) and SonarCloud¹⁰ (to provide comprehensive data on code quality static analysis, test coverage and other crucial metrics).

Figure 4.2 depicts the project's deployment viewpoint to provide a better overview of the development setup. The process starts with the developer creating a code with the IntelliJ IDEA IDE and pushing it to the GitHub VCS. Next, the quality assessment toolset triggers and evaluates the quality of the codebase. Additionally, the CI/CD tool Travis checks the checkstyle formatting and runs the tests. A notification of the created PR is sent to Slack¹¹. Depending on the case and the changeset's content, the code is later pushed to the JetBrains plugin repository (Marketplace). On reception of the codebase, JetBrains performs additional technical checks and reviews it manually. Once successful and safe, the plugin becomes available for users to download and install into their IDEs.

⁴github.com/a+courses/issues

⁵github.com/a+courses/wiki

⁶travis-ci.com/a+courses

⁷fossa.com

⁸snyk.io

⁹github.com/checkstyle

¹⁰sonarcloud.io

¹¹slack.com

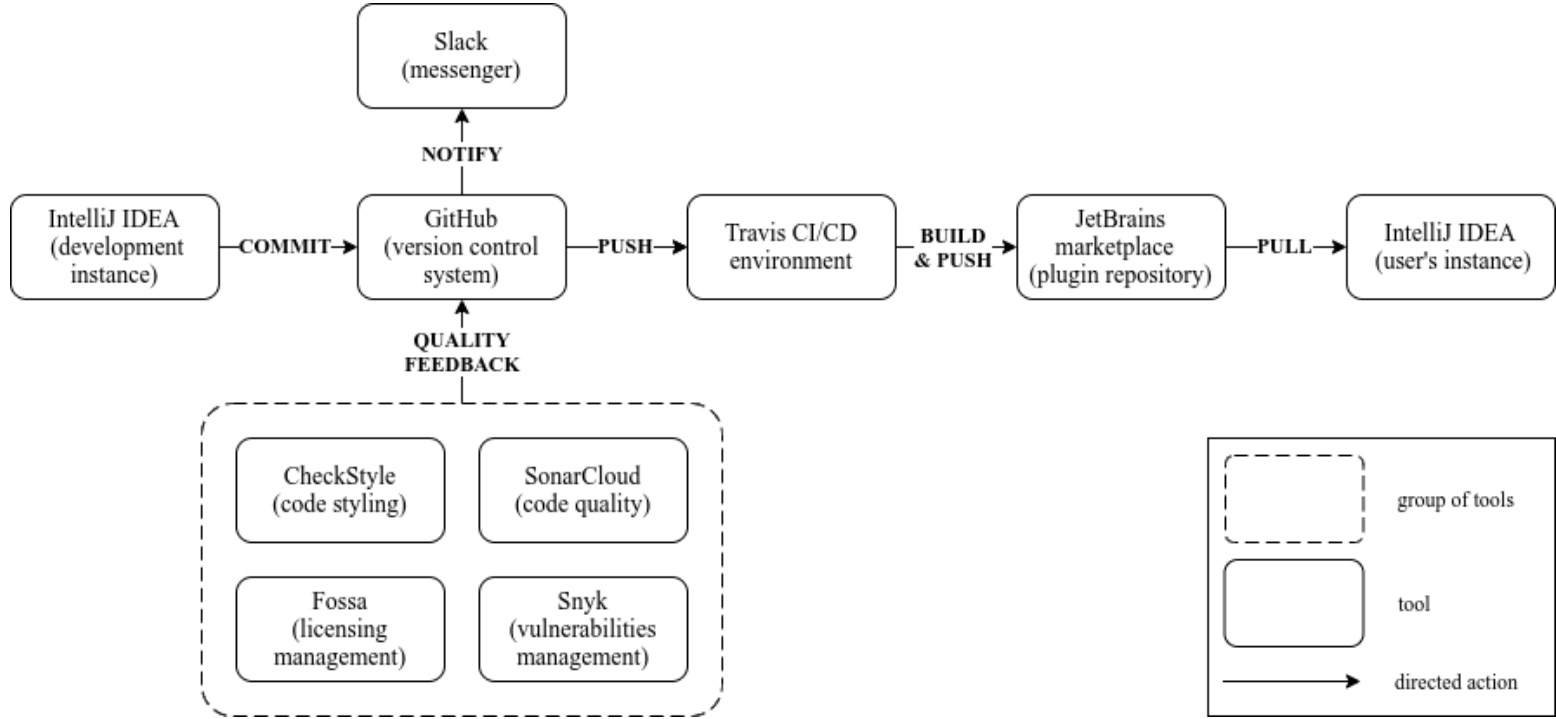


Figure 4.2: Deployment viewpoint of the A+ Courses Plugin.

4.3 Requirements

This project is outstanding when it comes to the requirements. The stakeholders have done great preparations before starting the project by creating and sorting features in the priority order. The given move allowed the team to better plan the development processes and the solution's technical architecture. Additionally, as the plugin development was conceptually tied to the beginning of the Programming 1 course, managing the deliverable scope was relatively easy.

The team's functional requirements list was presented in advance and divided into chapters like must-have, very important, desirable, and nice-to-have features. The list continued with a broad idea bank-like group of less crucial ideas for further development. The software's critical functional directions are enumerated in the List 4.2.

The plugin should have been operational with the IntelliJ IDEA Community Edition on the three main OS-platforms previously used by the Programming 1 course students. In 2019, the students' OS preferences distribution was the following: Windows (64%), macOS (24%) and Linux (13%) [2]. When it comes to the non-functional requirements, then one of the main messages by the stakeholders was:

A small but trustworthy feature set is far preferred to a large but untrustworthy set [5].

It resulted in a stronger emphasis on the stability and quality of the software. Resilience and fault-tolerance are vital for plugin success. Novice developers are overwhelmed with the cognitive load by default, so troubleshooting the plugin-related matters might reduce motivation (despite being a representative case of any software engineer's typical daily work in the industry or academia). Easy maintenance was also considered by the team when designing the plugin architecture. Funding and support for already developed services are a common problem in the education field. Another essential non-functional requirement for the project is extensibility. The architecture should ideally support an easy extension to handle other university courses that use A+ LMS as a backend environment.

- Plugin installation—as the name implies, the direction includes operations related to plugin installation into the IDE, configuring the environment to follow a unified set of keymaps, code styling, highlighting, and other minor tweaks. Completing the features from this direction provided end-users with a smooth startup experience and removed the need to shift the focus from learning the core material to learning the tool. The straightforward advantage of the feature set is installation speed and freedom from errors;
- Project, modules and configuration management—a larger group of requirements meant downloading and installing the project Software Development Kit (SDK), pulling and unpacking assignment modules and displaying them in a user-friendly way;
- Learning modules life-cycle management—a direction responsible for installing the assignment modules into the project, their updating and removals. Together with the previous group (project, modules, and configuration management), the given subset of features facilitates the modules' installation for the students making it clean and fast, compared to the manual process;
- REPL customization—a set of changes to the REPL, like classpath adjustment, distinctive naming, initial greetings, and auto-import of specifically required packages on the console startup. Starting modules from the correct working directories with the required dependencies on the classpath, importing required packages, descriptive naming, and customized initial message help students to understand the current state of the REPL and save them from multiple errors and frustration that earlier students faced while using it;
- Submissions from the IDE—this collection includes basic authentication against the backend LMS, sending the predefined assignment module files for grading, displaying grading results in the IDE, ability to quickly switch to the A+ LMS browser view. Integration with the backend LMS allows students to mostly work within the IDE, saving thus time usually spent on switching between the IDE and the web interface of the A+;

List 4.2: List of Plugin development directions, grouped by function [4].

Chapter 5

Solution

The given chapter describes the implemented solution in the form of a report of lessons learned. It starts with the higher-level architecture and presents the structure of the plugin from the component and layer perspectives. Additionally, the architecture subchapter describes the programming patterns used. The next parts of the "Solution" chapter (Integration with the Platform and Integration with the external systems) explain how the plugin interacts with a hosting platform and also an external data provider A+ LMS. The chapter also highlights challenging aspects of the development process, like managing the dependencies and approach taken for the User Interface/ User Experience (UI/UX) building. The pre-final part of the solution description reveals details of the testing path and overall quality assurance. In the last Subchapter 5.7, "Delivery and operations jump-start" the author explains the challenges and issues faced when bringing the plugin into the real-life use.

The plugin's official full name as presented in the JetBrains' plugin repository is "A+ Courses" plugin. Active development lasted 9 months from the beginning of January 2020 until the most critical issues were fixed after the Programming 1 course has started.

At the moment of writing this paragraph, the plugin had about 4 000 downloads. It took almost 1100 commits and 200 PRs to create the plugin. Overall, 412 issues were posted during the project, 89% of them are resolved by now and the remaining 9% are labeled as either developer or user-discovered bugs. The software size is approximately 16 000 Lines of Code (LOCs), where Java code makes up roughly 96% and Scala 4% (due to the extension of the Scala Plugin functionality implemented in Scala). The current unit test code coverage is 61.3%.

5.1 Architecture

The key factors having a significant influence on the plugin architecture are a) the target (IntelliJ) Platform's composition and b) functional requirements for developing the software (see Subchapter 4.3 for details). The general structure of the application is shown in Figure 5.1.

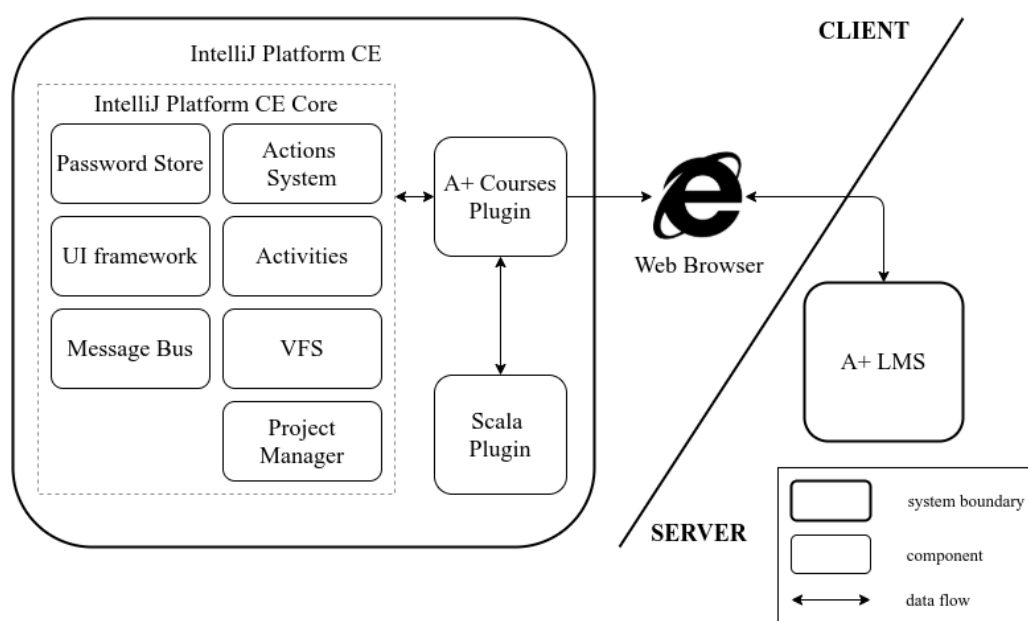


Figure 5.1: Component viewpoint of the A+ Courses Plugin architecture.

As seen from the Figure 5.1, the plugin interacts with multiple core components of the platform. Password Store is used to locally store users' passwords in a secure way. It uses OS-level installed software (like KeePass¹, Apple's Keychain Services², etc.) to encrypt and store sensitive data on the users' machine. A mixture of the JetBrains' own wrapper on the Swing³ User Interface (UI) framework is being used. However, in some special cases, default Swing components are applied. Platform Message Bus and Virtual File System (VFS) are used to handle the notifications and file changes correspondingly.

Additionally, the Message Bus is being actively utilized to communicate the plugins' internal state to the users. For example, notifying about the status of the tasks, showing notifications and error messages. Further, VFS

¹keepass.info

²developer.apple.com/keychain_services

³docs.oracle.com/swing

plays an essential role in the process of the assignment module lifecycle management. Project Manager and Actions Systems are applied in the Platform-standard way to control the project state and trigger Actions correspondingly. Finally, the Activities framework is being extended to launch specific background processes on the startup of the application.

As highlighted in the Figure 5.1, plugin interacts with A+ LMS to fetch the data related to the course and assignments over the Hypertext Transfer Protocol (HTTP). Also, it connects with the user's local web browser (for simple link opening). The architectural design of the plugin attempts to maximize the separation of generic functions from those of the Platform.

Following general convention for Java applications, the A+ Courses Plugin's source code is divided into two key packages: 'main' and 'test'. Each of those packages contains a code made with both Java and Scala. Gradle⁴ Scala Plugin⁵ is used to compile the mixed-language environment.

Inside the packages, the code is structured according to the layer it belongs to. The plugin has the following high-level layers:

- Data Access Layer holds the code responsible for the plugin's external iterations with A+ LMS. The most notable parts are interfaces `Client.java` and `Parser.java` and the implementing them `APlusExerciseDataSource.java` class;
- Model Layer, as the name implies contains classes and interfaces that represent different objects the plugin operates with. It includes two primary groups of entities: one to accommodate data received from the A+ backend (for example `Exercise.java`) and another reflecting the internal structure of the plugin (e.g., `Module.java`);
- Presentation Package is a bundle of base and actual `*ViewModel.java` classes that contains presentation logic and stores code responsible for View manipulation such as filtering;
- UI Layer contains classes in charge of rendering the actual UI (dialogues, forms, renderers and other common visual components);
- Utils Package is the multi-tool package that holds specialized code for various tasks: a common Fork—Join model (FJ) implementation, data types-manipulation code, generified data structures used throughout the plugin (like `Tree.java`), language resources and properties bundles-management code, and classes that handle events processing and data parsing functionality;

⁴gradle.com

⁵github.com/gradle-intellij-plugin

- "IntelliJ" Package is listed last and contains most of the platform-related source code, grouped by the component of the Platform it interacts most with;

The plugin builds on the industry's best practices and utilizes multiple design patterns. The overall structure of the plugin follows the Model—View—Presenter (MVP) programming pattern. The source code contains three distinctive packages that serve corresponding goals: accommodating Model classes that provide the plugin with the key entities (Model layer); storing View logics (UI layer); and enclosing presentation logic that shows and processes the data (Presentation package). Additionally, the Observer pattern is used inside the MVP to ensure communication between the layers with `ObservableReadWriteProperty.java` and `ObservableReadOnlyProperty.java` being the two main implementations. The Factory pattern is actively utilized to spawn various classes such as models, configurations, and UI components. A custom FJ framework is used to handle tasks during the modules installation process. At places, the Builder pattern is being utilized to provide a better abstraction of the code (`TreeModelBuilder.java`) as well. Finally, the Singleton pattern is used to ensure that the instance of `PluginSettings.java` is presented just once in the runtime.

5.2 Integration with the Platform

This chapter focuses on the plugin integration with the IntelliJ Platform exclusively. Further, it covers the interplay between the designed A+ Courses Plugin and the Scala Plugin.

IntelliJ Platform provides multiple ways to use its resources. These include point-to-point (extension points) integration and messaging or direct programmatic calls using available endpoints (through the Java access control for classes and their members). The developed A+ Courses Plugin applies all of those. Mostly, the Application/Project/Module Managers' features use OpenAPI, but not limited to it. In addition, other Platform components are reached through the API endpoints (VFS, Actions System, UI). Regardless of this approach playing a central role for the IntelliJ Platform, features like notifications and bulk changes in the VFS still connect to the Message Bus to communicate with the System. As applied with `LocalHistory`, `PropertiesComponent`, `PasswordSafe`, the least used approach for contacting the Platform is direct calling of the components. One more way to interact with the IntelliJ Platform is via so-called extension points (dedicated parts of the API designed to extend the functionality), exemplified by

the `InitializationActivity.java` that extends the existing (post) startup activities of the Platform (shown in the Listing 5.1).

```
<extensions defaultExtensionNs="com.intellij">
  <backgroundPostStartupActivity implementation=
    "...InitializationActivity"/>
</extensions>
```

Listing 5.1: IntelliJ Platform’s point-to-point extension declaration example (part of the full path to `InitializationActivity.java` class is omitted).

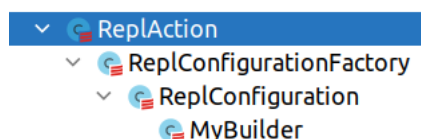


Figure 5.2: Embedded structure of the `ReplAction.scala`’s inner classes.

The last example of the plugin interaction within the Platform can be seen in the source code’s part responsible for extending the Scala Plugin (mostly REPL-related). In this case, the functionality to be enhanced is not exposed as an extension point. The feature implementation is done through the direct extension of the Scala Plugin classes, shaping a complex embedded structure (see Figure 5.2).

5.3 Integration with the external systems

In contrast to the previous Chapter 5.2, Integration with the Platform, given part describes plugin interactions with the A+ LMS backend via Django⁶ REST API and the static resources stored in the same ecosystem.

Integration with the A+ LMS that acts as a source of data for the Programming 1 course assignments was implemented using Apache’s HTTP Client⁷. At the given stage of the project, users are authenticated with a token stored in the A+ user interface. The authentication feature is of critical importance for the assignment submission. However, such token-based workaround was chosen over a proper HAKA⁸ authentication due to the complexity of the latter and a need for an additional development on the A+ API side. Overall, plugin extracts the following data from the A+ LMS

⁶djangoproject.com

⁷hc.apache.org

⁸csc.fi/haka

backend: results, attempts, history, etc. As such, the A+’s REST API has been slightly modified to provide the information required for the plugin.

Lastly, a significant amount of data and main configurations of the A+ Courses Plugin uses come from yet another source. As configuration and customization of the course in A+ is a complex process, the decision to create a separate configuration file for the plugin was made. It contains various information, including download locations of the IDE and Project settings, but also locations of the assignment modules with unique id’s. Additionally, it has data on the localized (FI/ENG) names of the modules and sets of initial commands to be run on the REPL startup. Finally, the file presents the course (Programming 1) related information for example id, mandatory-to-install libraries, etc. The configuration file is created by a semi-automated script and stored along the other course materials in the Aalto University’s internal VCS. It is exposed as a static resource in one of the components A+ LMS (Grader).

5.4 Dependencies

Despite the modest size, the A+ Courses Plugin has many various dependencies. The first and most obvious one, is its dependency on the platform; or, to be more precise, on the version of the Platform SDK, used to build it. The next layer of dependencies contains the Gradle build plugins that do the actual work when translating the plugin’s source code into the executables. One more layer has the regular application dependencies (or merely other Java libraries, whose code is being used in the plugin as a 3-rd party source). Lastly, the plugin is affected by the Scala SDK version used in the Scala Plugin. The A+ Courses Plugin relies on it when enhancing some of the Scala language features. As follows, this subsection will briefly present the approach taken in handling all of the dependency layers listed.

The version of the IntelliJ Platform SDK used to build the plugin has a critical impact on its compatibility: only plugins developed for a particular version (based on the version range) are allowed to run in the corresponding IDE. It results in a challenging situation, where several versions of the plugin artifacts must be maintained to support multiple major versions of the IDE. This challenge has not yet been faced. However, the Programming 1 course’s duration is about five months, while the the IntelliJ IDEA major versions are released quarterly. As such, the development team will have to address this challenge once the IntelliJ IDEA 2020.3.x series are released.

There are two main solutions for the problem. The technical and straightforward one is to follow the IntelliJ’s common approach and create a version

of the plugin able to handle the latest version. Alternatively, users could be asked not to update the IDE for the duration of the course (which is less preferred, as can not be controlled). In this case, an additional challenge is related to the evolution of the Platform itself. Being a commercial product, it is continually moving forward with a large team of developers facilitating the change. This results in regular adjustments in the APIs, ranging from minor to complete changes of the fundamental components (like the recent change of the Marketplace-related processes). Mechanisms for introducing and deprecating new endpoints exist, using annotations. However, the process still seems suboptimal, often leading to undocumented changes.

The next layer of dependencies is the Gradle plugins. Most notable of being 'org.jetbrains.intellij' and 'scala'. The Gradle Scala plugin (not to be confused with the "Scala Plugin" used elsewhere) has been mentioned earlier with regard to the A+ Courses Plugin deployment (see Subchapter 5.1). The Gradle IntelliJ plugin that provides various build tasks to support the development and deployment-related processes has proven tricky. It seems that in some cases, the compatibility between the Platform versions and the Gradle IntelliJ plugin versions is not thoroughly tested and can cause unexpected and hard-to-discover bugs [1].

Another slice of dependencies consists of the standard Java libraries, including Scala (SDK) library for JVM, Apache Commons IO⁹ and several libraries to support the testing process JUnit, REST-assured, Mockito, and Powermock. In general, this is the most straightforward part of the dependency management process. The automated vulnerability tool regularly checks the source code repository and notifies the development team of the discoveries for them to follow up [6].

However, few aspects were not as simple. For example, it was discovered that the chosen JUnit (version 4.12), which is considered one of the most commonly used and stable versions of the 4.x range in the industry, conflicts with the IntelliJ Platform's bundled version from the older 3.x range. This fact introduced inevitable confusion and additional complexity to the process. Also, an update of the Gradle IntelliJ Plugin to the newer version caused a conflict with the REST-assured library blocked the build of the A+ Courses Plugin, but was eventually resolved [3]. Finally, the versions of Scala SDK of the Scala Plugin and the one used in the plugins dependant on them (like the A+ Courses Plugin) must match with mismatch resulting in inability to compile and other minor issues.

So far, the approach to the versioning of the A+ Courses Plugin as a software has been relatively simple. Each new version nests all the things

⁹commons.apache.org/commons-io

from the previous versions with few fixes and improvements. No breaking changes have been introduced so far.

5.5 Approach to UI/UX

The general approach to the UI/UX is to mimic the IntelliJ Platform one. The plugin interaction elements are grouped and presented in a similar way. Standard Platform icons are used, partially or fully painted with an accent color ("acidic pink"¹⁰). The plugin provides differently-painted icons for both light and dark interface themes. The idea of the accent color is to highlight the elements that belong to the A+ Courses Plugin and distinct them from the IntelliJ IDEA ones. To facilitate easier mapping between the environments, icons that represent the state of the submittable assignments in the UI follow the corresponding coloring patterns used in the A+ LMS. To simplify the transition between the IDE and the A+ Web (browser) interface, plugin has embedded links pointing to the required A+ pages (like one in the assignment submission results notification).

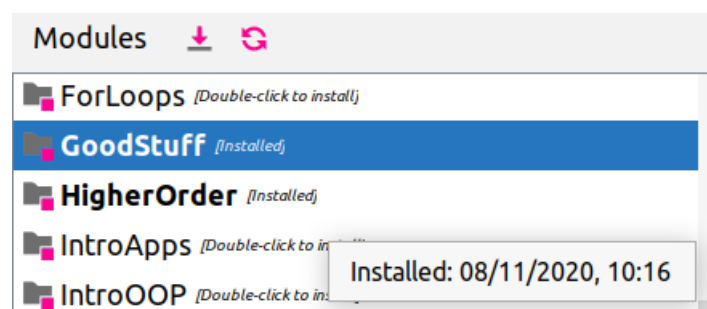



Figure 5.3: Modules view of the A+ Courses Plugin.

Figure 5.3 is a showcase of a Module view part of the A+ Courses tool window. It shows how the default IntelliJ IDEA icons (module's ones in this case) are adapted with an accent color. Additionally, it presents the main layout where the list of available modules is loaded and made available for various actions. The screen capture also contains buttons to download and check the remote server for the modules' newer versions. As seen from the figure, the statuses of the modules are also shown (*[Installed]*, *[Double-click to install]*; not shown ones are *[Removed]*, *[Error]* and some others, describing the intermediate steps).

¹⁰#FF0090 or (255, 0, 144) or 

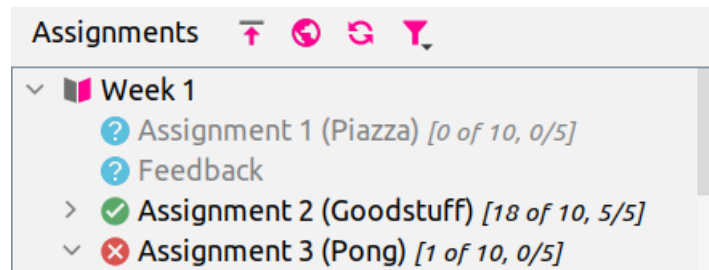


Figure 5.4: Assignments view of the A+ Courses Plugin.

Figure 5.4 shows the Assignment subview of the A+ Courses tool window. It allows users to see the assignments tree that contains weeks, corresponding assignments and submission links (if submission attempts have been made). Also, the figure comprises four action buttons: to submit the assignment, to view submission results in the browser, to refresh the assignments/submissions list and an option to filter assignments by their type. It is also possible to filter out the following types of assignments: non-submittable (from the IDE), completed and optional ones.

Furthermore, the UI shows the amount of attempted and remaining submissions. Multiple color codes are used. For example, greyed out assignments can only be submitted from the A+ Web UI. The green-tick-assignment-icon means that there has been at least one successful attempt. The cross-on-the-red-background pin represents a failed submission. Additionally, the maximum amount of points acquired is being shown. More examples of the A+ Courses Plugin UI/UX (configuration dialogs, REPL view) are presented in the Appendix 8.4.

The plugin architecture is designed to support many required tasks and Actions to be handled in the background threads. The move allows users to save time instead of waiting for the indexing and installation process to complete. Creation of the UI components is supported by a custom-built hierarchy of supporting elements like `BaseListView.java`, `OurDialogWrapper.java`, `DialogBaseHelper.java` and few others located at the 'ui' package. The A+ Courses Plugin, as well as the platform, supports English as the interface language.

5.6 Testing & quality

For the stakeholders, the quality and stability of the software are among the most valued non-functional requirements. The condition is addressed on several levels. The first level is static analysis tools that check and verify

the code's syntactic and stylistic validity against the plentiful rules. Next, the unit tests should be created and passed. Naturally, all the previous unit tests should hold as well.

At this point, there are at least three distinct approaches to the creation of unit tests. One of them tends to isolate plugins's code from the Platform and perform *pure* unit tests. Another method uses the IntelliJ Platform's dedicated features that help the testing (such as `BasePlatformTestCase.java`). However, this latter approach is not pure and can also be considered a Platform test, as multiple supporting Platform objects are created under the hood. The last technique to unit testing is not very common. It uses the PowerMock library that allows bypassing the Java access control features, breaking thus the encapsulation and allowing almost any part of the code to be tested.

If the task involves an interaction outside of the Platform, the integration tests against the target platform (A+ in this case) should also be created. These are conditionally executed in the CI/CD environment that spawns itself as a full-featured dockerized A+ LMS system filled with dummy data during the runtime.

The Aalto University remote desktop functionality¹¹ was used to test the plugin with Windows 10, Ubuntu 18.04 (offered to students as a fall-back guaranteed-to-work Linux environment) and manually on a standalone macOS Catalina-equipped MacBook [5]. More extensive features were described as a manual testing scenarios and walked-through at the end of each Sprint-/before a release. Additionally, the development team has involved a group of Summer interns for the final, pre-production exploratory testing of the plugin. It has given positive results and helped to reveal several technical problems.

5.7 Delivery and operations jump-start

The first engagement of the A+ Courses Plugin with the real users occurred in August 2020. A so-called 0-week is a preliminary week where the course materials are already open but the course itself has not yet officially begun. During that week, all the team's eight months-long development efforts were trialed. Specific preparations were made to support a smooth start for the plugin. Before the "final" delivery, a list of tasks to "productize" the plugin was developed, including updating the dependencies to the latest versions, creating all possible manuals and documentation, populating the JetBrains'

¹¹vdi.aalto.fi

Marketplace with marketing data, proof-reading the plugin texts, etc. Overall, the polishing took 3-4 weeks and the first production-grade version of the plugin was deployed in the last week of August as per schedule.

When the course has actually started, the development team has been actively involved by supporting and troubleshooting the most challenging installation cases. This step allowed the team to acquire insights into the actual student processes and proactively discover several bugs that were immediately patched with the next-version release.

Just few new developments have been made during the period. It was explicitly decided to keep a resource buffer for solving initial user issues if these arise. During the first weeks of the Programming 1 course, the key indicators were monitored daily. The last report and the plugin download distribution chart are shown in the Figure 5.5.

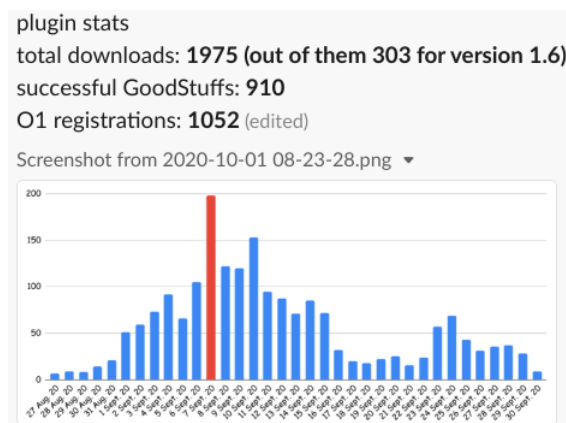


Figure 5.5: Regularly monitored plugin usage indicators.

The figure contains few parameters. Firstly, a total quantity of unique downloads on the 1st of October 2020 (roughly four weeks from the official courses start). The amount is close to 2 000 as each download of a new plugin version in JetBrains Marketplace counts as a unique. Next, the figure reveals the number of successful submissions 910 - for the first practical assignment "GoodStuff". The parameter indicates that the student has gone through the intended learning pipeline and submitted their first assignment successfully, which confirms the validity of the local setup. The last monitored indicator is the total amount of students registered for the Programming 1 course, slightly over a 1 000. The difference between the total number of registrations and the successful submission of the first assignments was continuously reducing since the start of the course. More and more students have made their way with the A+ Courses Plugin.

Chapter 6

Discussion

6.1 Meeting requirements

Overall, the plugin meets both sets of requirements: functional and non-functional. The simplified list of the developed features is shown in the List 6.1 (for a more detailed list of requirements see List 4.2). Completing those requirements provided the students of Programming 1 with a smooth and positive learning experience, shielding them from multiple potential mistakes and imperfections of the backend LMS. The plugin's introduction has significantly improved the usability of the IntelliJ IDEA for the purposes of the course. The validity of each functional requirement was ensured right after the feature delivery at the demo, but also through a formal quality assurance process.

Most of the discovered *post factum* issues were quickly fixed as per the agreed maintenance model¹. The feedback from the stakeholders has been positive. According to the feedback, students tend to like the plugin's features (see Appendix 8.6). Compared to the other relevant examples (see Table 3.1), the plugin - as an OSS, containing features for installing and submitting the assignments - evidently follows the common best practices and processes. In addition, there are certain features that the A+ Courses Plugin has already pre-planned, including multiple courses support, learning data collection implementation, tools to run assignments-related test-suites locally, etc.

Altogether, the choice of IntelliJ IDEA as the course IDE and the development of the A+ Courses Plugin has proved a success. The initial prerequisite of simplifying the students' on-boarding process and further submissions and results retrieval has been met. The plugin has had a positive impact on the

¹github.com/a+courses/maintenance

- Plugin installation—block of requirements responsible for a smooth startup experience and removing the need to nebulize the focus from learning the core material for students;
- Project, modules and configuration management;
- Learning modules life-cycle management—together with the previous group (Project, modules, and configuration management) facilitates the module installation, clean and fast if compared to the manual process.
- REPL customization—importing packages, descriptive naming and customized initial message helps the end-users to understand the current state of the REPL and saves them from multiple errors and frustration;
- Submissions from the IDE—integration with the backend LMS allows students to work within the IDE, saving time that would otherwise be spent on switching between the IDE and the web interface of the A+.

List 6.1: List of Plugin development directions, grouped by function [4].

streamlining the teaching of the Programming 1 course. The author suggests that educational plugins form a strong link between an excellent modern IDE and a MOOC platform, and help creating a smooth learning experience.

Compared to other educational plugins and, following the requirements² list, it becomes clear that there is a lot of potential development ahead. At this stage, however, the multiple course support is of highest priority for the team. Among the technical challenges, establishing a more comprehensive and a more profound automated testing framework are considered as the most important.

6.2 Technology choice and the learning curve

The choice of implementation technologies was simple. Most of the IntelliJ Platform is done with Java, with a small part utilizing Kotlin. When the project started, the Java version advised by the JetBrains was 8. The Java 11 became recommended closer to the plugin's release date, so it is promoted to future developments along with Kotlin. Lately, the JetBrains has taken numerous steps to simplify the start of the plugin development. One of those

²github.com/a+courses/requirements

attempts has introduced plugin templates hosted on the GitHub. Unfortunately, these were not available back in January 2020, so the team used the best approach available at the time, Gradle. After the key technologies were chosen, specific libraries and dependencies for the plugin were selected. It was done on a rolling-basis after discussion in the development team.

At the beginning of the project, none of the developers had experience developing the IntelliJ Platform plugins. Due to the prior programming experience and familiarity with the OOP-oriented technologies (e.g. Java, C#³ and confident usage of the JetBrains products), however, it was possible to start the features delivery quickly. An additional factor positively affecting the process, was an availability of comprehensive documentation, including quick-start guides and strong support from the vendor (Slack/Gitter channels, forums). Overall, the team managed to kick-off the development process quickly.

The learning of the Platform, however, did not go without bumps. No significant issues appeared; however, the inability to fully understand the Platform resulted in attempts to isolate the plugin components. It has been done through an additional layer of interfaces. The move slightly complicated the architecture and demanded extra tricks while testing. In the author's opinion, the team has not yet (ten months after the start of the development) achieved a mature understanding of the system. However, the present level of knowledge of the Platform structure suffices for a comfortable work on the plugin features. The IntelliJ Platform has been built for almost twenty years, evolving and changing all the time. It has resulted in several challenges related to documentation, interfaces and approach to versioning.

6.3 Documentation and versioning

The IntelliJ Platform is massive (approximately 9.9 million Lines of Code⁴), so creating and maintaining documentation even if for a part of the system is a resource-demanding job. Often, functionality is not described extensively or the documentation is even incomplete. This forces the plugin developers to dive into the source code of the Platform in their search for answers. Typical challenges are related to the age of the system. Different teams created parts of the Platform at different times (as mentioned, the period is decades here), so various programming paradigms and styles are inevitably nested within it. Despite this challenge, the code of the Platform still feels relatively uniform. A secondary challenge related to the codebase originates

³microsoft.com/csharp

⁴data by chrome.google.com/github-gloc

from a recent adoption of Kotlin for the new Platform components. As follows, the developers must be Kotlin-fluent as well.

One time, the author of the thesis tried to activate a recent Platform feature in the A+ Courses Plugin which failed for no apparent reason (forced auto-restart during the plugin installation). The solution came from one of the IntelliJ Platform software engineers, who hinted that the feature had not been released despite it being described in the docs.

The next faced challenge is related to the evolution of the codebase. Apart from the Platform being multi-paradigm and multi-language, some plugins, like the Scala Plugin, are written with the Scala programming language. This increases the environment's perceived complexity further.

The IntelliJ Platform development teams do invest into making the APIs evolution more transparent and predictable (for example, through annotating specific methods like `@ApiStatus.Experimental`, `@Deprecated` and `@ScheduledForRemoval`). However, situations where a large piece of the code is silently rewritten also occur. One such example is a renewal of the Marketplace-related functionality during the Spring-Summer period (this is likely related to introducing a new approach to the plugins licensing). It had a versatile impact on the A+ Courses Plugin: on the one hand, any change distracts the team from further feature development; on the other hand, this case proved a blessing in disguise. As follows, a better plugin installation mechanism provided by the Platform, allowed to deprecate and remove a part of the A+ Courses Plugin's functionality.

Finally, the versioning of the Platform is somehow challenging. For the plugin to be compatible with every new Platform (SDK) version, a separate branch must be spawned. Naturally, this increases the support lent to the deployment pipelines' variability (see Figure 4.2). For the older plugins or the ones that must support multiple major and minor version of the IntelliJ IDEA, the list of the deployment-related branches might be quite lengthy (and look like something shown in the Figure 8.1).

6.4 Architecture

The plugin has fit the architecture of the Platform rather well. Following main principles of the interoperation with the Platform allowed for a smooth functioning. The layering of the plugin's code worked well too. However, the latter resulted in quite a complex structure. The separation of the IntelliJ-related code into its own package did not add much value as the plugin's introduction to the non-IntelliJ Platform-based IDEs is rather unlikely. Also, the separation is not 100% clean as some of the IntelliJ Platform-related code

has leaked into the "UI" layer, if not further. Extracting the "business logic" code into a separate SDK-like library was discussed, but is currently deemed as unviable as could introduce excessive effort during the development process.

The pattern implementation has mostly proved successful. Simplifying the long-debated `TwoWayBindableObservable.java` class has resulted in a more lightweight Observer pattern. This is an important, smooth and elegant part of the Model—View—Presenter (MVP) architecture that acts as an application cornerstone despite it being hard to grasp conceptually. The secondary patterns (Builder and Factory) are used on-demand and their variety represents the developers' difference of opinions. In some cases, however, it was commonly agreed that the specific Factory methods must be simplified (e.g., the `createCourse(@NotNull String id, ...)` method of `IntelliJModelFactory.java`).

Some architectural decisions are, however, proven to be hard to maintain. As the application's performance was not considered as important, the implementation of several parts of the plugin in a concurrent manner is considered an overkill. As the time shows, these components have caused several hard-to-catch bugs. In the author's opinion, concurrency does not compensate for the maintenance challenges it introduces. The concurrent features are not fully tested either because of their complexity or because of the lack of time. A more pragmatic argumentation could be built if the benchmarking of the concurrently handled processes is done. However, as the plugin does not process large data pieces or performs complex computations, this is unlikely to significantly impact users' perception of the plugin performance.

The custom implementation of the FJ model feels unjustified. The additional complexity it introduces does not seem to balance out the benefits. A more straightforward approach to the FJ could be using Java's native FJ framework for parallelization of the computations. Overall, the plugin architecture seems uniform and fluid, but also oversized for such a small application.

6.5 Integration

Integration with the external data providing system (A+ LMS) was not always straightforward. The lack of documentation on the particular implementation of the Django REST API resulted in several cases of an "unexpected" behavior of the system. Fortunately, the horizontal communication with the A+ team was always smooth and allowed to resolve these misunderstandings quickly. The key challenge of working with A+ originated from the

resourcing of the corresponding team. As implementing smaller features was made possible, the larger requests were pushed for the future. This occurred, for example, when deciding on the user authentication. As it was impossible to authenticate against A+ with HAKA without introducing additional service changes, the more straightforward token-based process was chosen.

Further, the Scala plugin development was also not the friendliest. It was mainly used for the REPL feature. The Scala plugin does follow the Platform's standard extension points approach, but the REPL-related endpoints are missing. It took some collective effort and non-trivial thinking to extend it. Another challenge coming from the working with the Scala Plugin is self-evident: it is easier to do in Scala. As a result, A+ Courses Plugin is now a multi-language software.

6.6 Approach to UI/UX

The resulting UI/UX feels fluent and easy to use. The development team, however, is considering several changes. Current parts of the main tool window are slightly overloaded with (mostly textual) information (see Figure 5.3 and Figure 5.4). The coloring scheme used to communicate the assignment status is built on the one by A+ LMS. However, a recently conducted accessibility analysis exposed that the approach used in the A+ is suboptimal and should be thus changed [7]. The development team does not have any design-minded members, so an external expert is required to handle this. In general, the current user interface allows students to complete all the vital operations (see List 4.3) quickly and in a relatively intuitive way. Please check Appendices 8.5 and 8.6 for the student quantitative and qualitative feedback correspondingly. Overall, the feedback has been mostly positive with few wishes and feature requests. These have been translated into the project backlog items or even implemented during the early stages of the O1 course.

6.7 Testing & quality

The chosen testing approach has proved rather feasible. The unit-tests cover most of the critical functionality; integration tests verify the validity of the data received from A+, while the UI-part of the application is tested manually. For the pilot stage of the project, this is deemed enough. However, as the project continues into 2021 (and possibly further), more steps should be taken to validate and verify the functionality. The author suggests in-

roducing an automated UI (or end-to-end) testing as vital for the project. The currently used approach is deemed demanding in terms of the resources manual testing entails. Additionally, the level of the test coverage should be increased higher (close to 80% or exceeding), otherwise making the extensive refactoring too risky if not impossible. This is, however, a typical challenge for small-scale development projects.

There are few more specific project-related testing aspects to consider. Firstly, to achieve a better readability, the process of the unit (platform) testing should be unified. The mocking Platform objects have been proven feasible, so the heavier platform-tests could be remade to match the approach. Secondly, tools like PowerMock could be eliminated as they bring little additional value, while affecting the application size and potentially supporting bad code design decisions. Finally, the integration test coverage must be raised to a higher level with the local testing environment matching the latest state of the A+ LMS. Additionally, pre-release testing with the interns has provided the team with great insights and helped to locate and fix potential issues before the release.

6.8 Operations

Team members' involvement with the students of the Programming 1 course at the early stage helped to collect first-hand feedback on the plugin functionality and provide solutions *ad hoc*. As most of the course TAs were not capable of solving the plugin-related matters, team's participation in the troubleshooting was vital. Unfortunately, not all disclosed root causes were readily fixable. It happened due to a temporary shift in priorities for the team members. However, solutions that allowed the students to continue the learning process were found and described⁵ quickly. The feedback was collected through multiple channels: course forum (Piazza⁶), feedback section of the course in A+, direct communication with the students and TAs in IM. One considerable challenge in terms of the feedback originates in the nature of the course. O1 is the first programming course for the many of students, so they might not yet have any relevant experience to compare to. Especially, if the usage of the IDE is new for them. However, some students who have participated in the Programming 1 course earlier provided positive feedback on both, the IDE and how the plugin, suggesting simplicity of the process as the main positive.

⁵github.com/a+courses/troubleshooting

⁶piazza.com/class/O1

Chapter 7

Conclusion

7.1 Closure

The chosen methodologies and research tools allowed the author to shape a proper context to tell the A+ Courses Plugin creation story. The literature review has helped to describe the modern technical toolset and approaches that the CS1 educators have. Also, it facilitated introducing the concept of plugins, starting from a higher-level and going deeper into the more specific implementations in the industry and application in research. Conducting a semi-structured interview with the key stakeholders of the LeTech group at Aalto helped explaining the state of affairs in teaching the introductory programming courses. As a part of the current state analysis, the insights and rationale for creating the plugin were collected and presented. To build a bolder context, a brief research of the industry's analogical solutions was presented. Later on, the plugin (software) development methodologies and environments were discussed. The practical part of the thesis described the solution and the challenges the team faced on the way. The thesis painted a detailed picture of the educational plugin development. Arguably, these lessons can mark the way for the future IntelliJ Platform-based educational plugins developers.

Overall, despite the difficult times and technical challenges, the A+ Courses Plugin project is considered successful. The formal requirements have been met, and the team has done an excellent job by jump-starting the plugin-based teaching process. The software development tempo was high, and so was the satisfaction level of the stakeholders. Non-functional requirements like stability and maintainability were also met considerably, shaping a solid base for any further development. As a small team, the A+ Courses development threesome rarely had enough resources to cover in detail all the

aspects of the application development and several sacrifices have been made.

7.2 Future developments

Slightly oversized architecture and possibly lack of testing might still fire back at developers in the future. However, the challenges have been identified and are likely to be addressed in the next development cycle, scheduled for 2021. Based on recent information, the expectations are high, but more resources will be added to the team.

However, this fact imposes a challenge by itself as new people can slow down the whole team for a while and add additional managerial overhead. On the technical side, the planned support for any additional courses will present a new set of requirements. It is likely to trigger a rework of the existing and addition of new UI components, demanding a better user authentication mechanisms and stress-testing the existing plugin architecture. Learning-related data collection, if attempted, might raise ethical, legal and technical concerns. Finally, ideas like integrating the Code With Me¹ JetBrains Plugin into the CS1 courses workflow might have a significant effect on the plugin structure.

¹plugins.jetbrains.com/code-with-me

Bibliography

- [1] A+ COURSES DEVELOPMENT TEAM. Aalto-LeTech/intellij-plugin: A+.MENU_ACTIONS bugfix, Pull Request #188. <https://github.com/Aalto-LeTech/intellij-plugin/pull/188>, 2020. Retrieved 2020-10-29.
- [2] A+ COURSES DEVELOPMENT TEAM. Aalto-LeTech/intellij-plugin: decide and configure testing environments, Issue #18. <https://github.com/Aalto-LeTech/intellij-plugin/issues/18>, 2020. Retrieved 2020-11-7.
- [3] A+ COURSES DEVELOPMENT TEAM. Aalto-LeTech/intellij-plugin: hacking continues, Commit #cb7627d. <https://github.com/Aalto-LeTech/intellij-plugin/commit/cb7627d9355c0b8633349110ac177c404b986c55>, 2020. Retrieved 2020-10-22.
- [4] A+ COURSES DEVELOPMENT TEAM. Aalto-LeTech/intellij-plugin: Project Milestones (Issues). <https://github.com/Aalto-LeTech/intellij-plugin/milestones?state=closed>, 2020. Retrieved 2020-10-22.
- [5] A+ COURSES DEVELOPMENT TEAM. Aalto-LeTech/intellij-plugin: Project requirements (Wiki). <https://github.com/Aalto-LeTech/intellij-plugin/wiki/Requirements>, 2020. Retrieved 2020-10-22.
- [6] A+ COURSES DEVELOPMENT TEAM. Aalto-LeTech/intellij-plugin: Update version of zip4j library to 2.6.1, Commit #bf0e222. <https://github.com/Aalto-LeTech/intellij-plugin/commit/bf0e222c6a35289364daa0499c0952c7b80d5213>, 2020. Retrieved 2020-10-30.
- [7] A+ LMS TEAM. A+ LMS, Accessibility and Inclusive Design Audit. <https://apluslms.github.io/accessibility-audit>, 2020. Retrieved 2020-11-09.

- [8] A+ LMS TEAM. A+ LMS, The extendable learning management system. <https://apluslms.github.io/>, 2020. Retrieved 2020-10-15.
- [9] ADMIRAAL, W., HUISMAN, B., AND PILLI, O. Assessment in massive open online courses. *Electronic Journal of e-Learning* 13, 4 (2015), 207–216.
- [10] ALA-MUTKA, K. M. A survey of automated assessment approaches for programming assignments. *Computer Science Education* 15, 2 (6 2005), 83–102.
- [11] ALEXA INTERNET. Edx.org Traffic, Demographics and Competitors - Alexa. <http://www.alexa.com/siteinfo/edx.org>, 2020. Retrieved 2020-10-15.
- [12] BECKER, B. A., AND QUILLE, K. 50 years of CS1 at SIGCSE: A review of the evolution of introductory programming education research. In *SIGCSE 2019 - Proceedings of the 50th ACM Technical Symposium on Computer Science Education* (New York, NY, USA, 2 2019), ACM, pp. 338–344.
- [13] BIRSAN, D. On Plug-ins and Extensible Architectures. *Queue* 3, 2 (3 2005), 40–46.
- [14] CODEACADEMY. What is an IDE? | Codecademy. <https://www.codecademy.com/articles/what-is-an-ide>, 2020. Retrieved 2020-10-15.
- [15] COURSERA. About | Coursera Blog. <https://about.coursera.org>, 2020. Retrieved 2020-10-15.
- [16] GROVER, S., AND KORHONEN, A. Unlocking the potential of learning analytics in computing education. *ACM Transactions on Computing Education* 17, 3 (8 2017), 1–4.
- [17] HUNDHAUSEN, C. D., OLIVARES, D. M., AND CARTER, A. S. IDE-based learning analytics for computing education: A process model, critical review, and research agenda. *ACM Transactions on Computing Education* 17, 3 (8 2017), 1–26.
- [18] JETBRAINS. Code With Me - plugin for IntelliJ IDEs | JetBrains. <https://plugins.jetbrains.com/plugin/14896-code-with-me>, 2020. Retrieved 2020-10-15.

- [19] JETBRAINS. EduTools - plugin for IntelliJ IDEs | JetBrains. <https://github.com/JetBrains/educational-plugin>, 2020. Retrieved 2020-10-16.
- [20] JETBRAINS. IntelliJ Platform: Open Source Platform for Building Developer Tools. <https://www.jetbrains.com/opensource/idea>, 2020. Retrieved 2020-10-19.
- [21] JETBRAINS. Java Programming - The State of Developer Ecosystem in 2020 Infographic | JetBrains: Developer Tools for Professionals and Teams. <https://www.jetbrains.com/lp/devecosystem-2020/java>, 2020. Retrieved 2020-10-16.
- [22] JETBRAINS. Plugin Class Loaders | IntelliJ Platform SDK DevGuide. https://jetbrains.org/intellij/sdk/docs/basics/plugin_structure/plugin_class_loaders.html, 2020. Retrieved 2020-10-19.
- [23] JETBRAINS. Plugin Structure | IntelliJ Platform SDK DevGuide. https://jetbrains.org/intellij/sdk/docs/basics/plugin_structure.html, 2020. Retrieved 2020-10-19.
- [24] JETBRAINS. Scala - plugin for IntelliJ IDEA and Android Studio | JetBrains. <https://plugins.jetbrains.com/plugin/1347-scala>, 2020. Retrieved 2020-10-19.
- [25] JETBRAINS. The State of Developer Ecosystem in 2020 Infographic | JetBrains: Developer Tools for Professionals and Teams. <https://www.jetbrains.com/lp/devecosystem-2020/scala>, 2020. Retrieved 2020-10-16.
- [26] JETBRAINS. What is the IntelliJ Platform? https://jetbrains.org/intellij/sdk/docs/intro/intellij_platform.html, 2020. Retrieved 2020-10-19.
- [27] JETBRAINS. Work with Scala code in the editor - Help | IntelliJ IDEA. <https://www.jetbrains.com/help/idea/edit-scala-code.html>, 2020. Retrieved 2020-10-19.
- [28] KAUPPINEN, T., AND MALMI, L. Aalto Online Learning – a pathway to reforming education at the Aalto University. *EUNIS 2017 – Shaping the future of universities* (2017), 212–221.
- [29] KHARRAT, D., AND QADRI, S. S. Self-registering plug-ins: An architecture for extensible software. In *Canadian Conference on Electrical and Computer Engineering* (2005), vol. 2005, IEEE, pp. 1324–1327.

- [30] LIGHTBEND. Lightbend Case Studies. <https://www.lightbend.com/case-studies>, 2020. Retrieved 2020-10-19.
- [31] LUXTON-REILLY, A., SIMON, ALBLUWI, I., BECKER, B. A., GIANNAKOS, M., KUMAR, A. N., OTT, L., PATERSON, J., SCOTT, M. J., SHEARD, J., AND SZABO, C. Introductory programming: A systematic literature review. In *Annual Conference on Innovation and Technology in Computer Science Education, ITiCSE* (2018), ACM, pp. 55–106.
- [32] MCCAULEY, R., FITZGERALD, S., LEWANDOWSKI, G., MURPHY, L., SIMON, B., THOMAS, L., AND ZANDER, C. Debugging: A review of the literature from an educational perspective. *Computer Science Education* 18, 2 (6 2008), 67–92.
- [33] MCDIRMIID, S., AND ODESKY, M. The scala plugin for eclipse. In *Proceedings of Workshop on Eclipse Technology eXchange (ETX)* (2006).
- [34] METALS. Scala (Metals) - Visual Studio Marketplace. <https://marketplace.visualstudio.com/items?itemName=scalameta.metals>, 2020. Retrieved 2020-10-19.
- [35] MICROSOFT. Introducing Visual Studio Live Share. <https://code.visualstudio.com/blogs/2017/11/15/live-share>, 2020. Retrieved 2020-10-19.
- [36] PECINOVSKÝ, R. BlueJ as the NetBeans Plugin. In *Advances in Intelligent Systems and Computing* (2017), vol. 511 AISC, pp. 264–270.
- [37] PIETERSE, V. Automated Assessment of Programming Assignments. In *Proceedings of the 3rd Computer Science Education Research Conference on Computer Science Education Research* (2013), Open Universiteit, Heerlen, pp. 45–56.
- [38] PROGRAMMING 1. Frequently Asked Questions | Ohjelmointi 1 | A+. <https://plus.cs.aalto.fi/o1/2020/wNN/faq/#the-scala-language>, 2020. Retrieved 2020-10-28.
- [39] RED HAT. What is an IDE? <https://www.redhat.com/en/topics/middleware/what-is-ide>, 2020. Retrieved 2020-10-15.
- [40] REIS, C., AND CARTWRIGHT, R. Taming a professional IDE for the classroom. In *Proceedings of the SIGCSE Technical Symposium on Computer Science Education* (New York, New York, USA, 2004), ACM Press, pp. 156–160.

- [41] RICHARDS, M. *Software Architecture Patterns*, 1st ed. O'Reilly Media, 2015.
- [42] RICHARDS, M., AND FORD, N. *Fundamentals of Software Architecture: An Engineering Approach*, 1st ed. O'Reilly Media, 2020.
- [43] SCALA CENTER. Metals and Scala 3 - Scala Center Updates - Scala Contributors. <https://contributors.scala-lang.org/t/metals-and-scala-3/4274>, 2020. Retrieved 2020-12-04.
- [44] SCALA IDE. Scala IDE (for Eclipse) 4.7.0 is out! <http://scala-ide.org/blog/release-notes-4.7.0-vfinal.html>, 2018. Retrieved 2020-10-19.
- [45] SORVA, J., KARAVIRTA, V., AND MALMI, L. A review of generic program visualization systems for introductory programming education. *ACM Transactions on Computing Education* 13, 4 (11 2013), 1–64.
- [46] SORVA, J., AND SEPPÄLÄ, O. Research-Based Design of the First Weeks of CS1. In *ACM International Conference Proceeding Series* (New York, New York, USA, 2014), vol. 2014-Novem, ACM Press, pp. 71–80.
- [47] SPACCO, J., HOVEMEYER, D., AND PUGH, W. An eclipse-based course project snapshot and submission system. In *eclipse 2004 - Proceedings of the 2004 OOPSLA Workshop on Eclipse Technology eXchange* (New York, New York, USA, 2004), ACM Press, pp. 52–56.
- [48] STACK EXCHANGE. Stack Overflow Developer Survey 2012. <https://insights.stackoverflow.com/survey/2019>, 2019. Retrieved 2020-10-16.
- [49] STACK EXCHANGE. Stack Overflow Developer Survey 2020. <https://insights.stackoverflow.com/survey/2020>, 2020. Retrieved 2020-10-19.
- [50] TESTMYCODE DEVELOPMENT TEAM. **BETA** TestMyCode plugin for IntelliJ IDEA. <https://github.com/testmycode/tmc-intellij>, 2020. Retrieved 2020-10-28.
- [51] TESTMYCODE DEVELOPMENT TEAM. TestMyCode NetBeans plugin. <https://github.com/testmycode/tmc-netbeans>, 2020. Retrieved 2020-10-28.

- [52] TRÆTTEBERG, H., AND AALBERG, T. JExercise: A specification-based and test-driven exercise support plugin for Eclipse. In *Proceedings of the 2006 OOPSLA Workshop on Eclipse Technology eXchange, ETX 2006* (New York, New York, USA, 2006), ACM Press, pp. 70–74.
- [53] VIHAVAINEN, A., AIRAKSINEN, J., AND WATSON, C. A systematic review of approaches for teaching introductory programming and their influence on success. In *ICER 2014 - Proceedings of the 10th Annual International Conference on International Computing Education Research* (2014), pp. 19–26.
- [54] VIHAVAINEN, A., VIKBERG, T., LUUKKAINEN, M., AND PÄRTEL, M. Scaffolding students’ learning using Test My Code. In *Annual Conference on Innovation and Technology in Computer Science Education, ITiCSE* (New York, New York, USA, 2013), vol. 13, ACM Press, pp. 117–122.
- [55] VLISSIDES, J. Pattern Hatching PLUGGABLE FACTORY, Part II. *C++ Report 11*, February (1999), 1–10.
- [56] WOLFINGER, R., DHUNGANA, D., PRÄHOFFER, H., AND MÖSSENBOCK, H. A component plug-in architecture for the.NET platform. In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)* (2006), vol. 4228 LNCS, pp. 287–305.
- [57] ÉCOLE POLYTECHNIQUE FÉDÉRALE LAUSANNE. The Scala Programming Language. <http://www.scala-lang.org>, 2011. Retrieved 2020-10-19.
- [58] ÉCOLE POLYTECHNIQUE FÉDÉRALE LAUSANNE. Scala 3- A community powered release | The Scala Programming Language. <https://www.scala-lang.org/blog/2020/09/15/scala-3-the-community-powered-release.html>, 2020. Retrieved 2020-10-19.

Chapter 8

Appendices

8.1 Appendix 1. Semi-structured interview questions

Semi-structured interview questions:

- What are the strong sides of CS1 at the Aalto University?
- What challenges & needs (technical, conceptual, financial, etc.) does the primary programming education at Aalto pose?
- Which future developments do you expect to see in the novice programming teaching at the Aalto University?
- What reasons motivated you to get involved in the development of the educational IDE plugins?
- How would you rate the success of the A+ Courses plugin project so far?

8.2 Appendix 2. Technologies and tools

Technologies:

- Java 8¹
- IntelliJ Platform
- Swing
- JUnit4²
- Scala 2.x
- ScalaTest³
- Bash⁴
- Gradle

Tools:

- CheckStyle
- Travis CI
- GitHub Projects
- GitHub Wiki
- IntelliJ IDEA
- Snyk
- Fossa
- SonarCloud
- Slack

¹oracle.com/java8

²junit.org/junit4

³scalatest.org

⁴gnu.org/bash

8.3 Appendix 3. Scala Plugin repo branches



The image shows a screenshot of the 'protected branches' section of a GitHub repository. It lists seven protected branches, each with a shield icon, a branch name in a blue box, and update information. The branches are listed in descending order of their last update time.




	<code>idea201.release</code>	Updated 3 months ago by semkagtn
	<code>idea193.release</code>	Updated 6 months ago by Dmitrii Naumenko
	<code>idea192.release</code>	Updated 11 months ago by mutcianm
	<code>idea191.release</code>	Updated 2 years ago by Dmitrii Naumenko
	<code>idea183.release</code>	Updated 2 years ago by niktrop
	<code>idea182.release</code>	Updated 2 years ago by adkozlov
	<code>idea181.release</code>	Updated 2 years ago by niktrop

Figure 8.1: Scala Plugin GitHub repository protected branches.

8.4 Appendix 4. A+ Courses plugin UI/UX

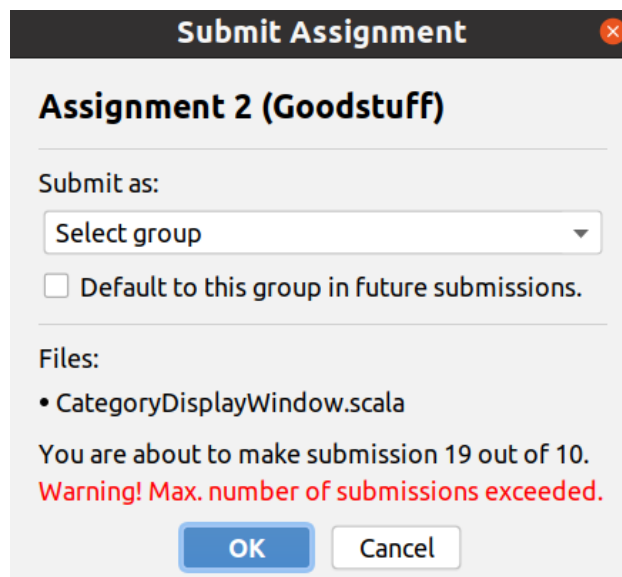


Figure 8.2: Assignment submission dialog.

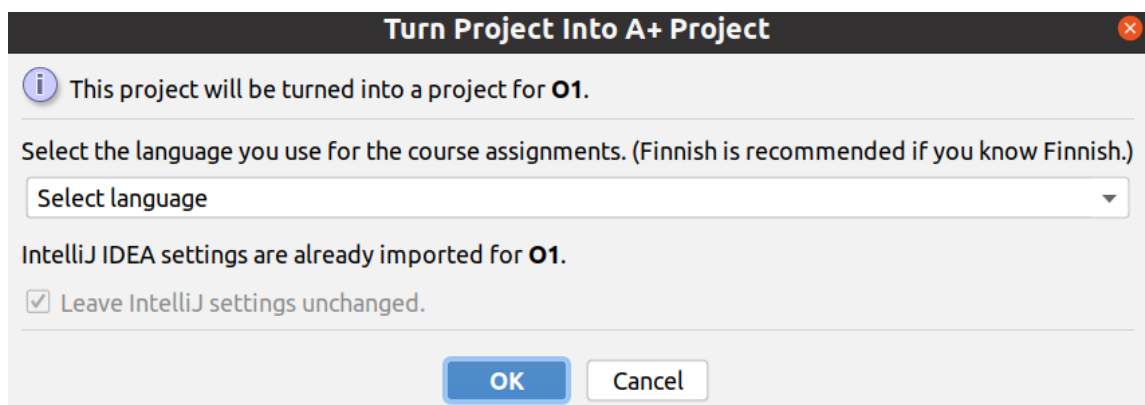


Figure 8.3: Turn Project into A+ Project dialog.

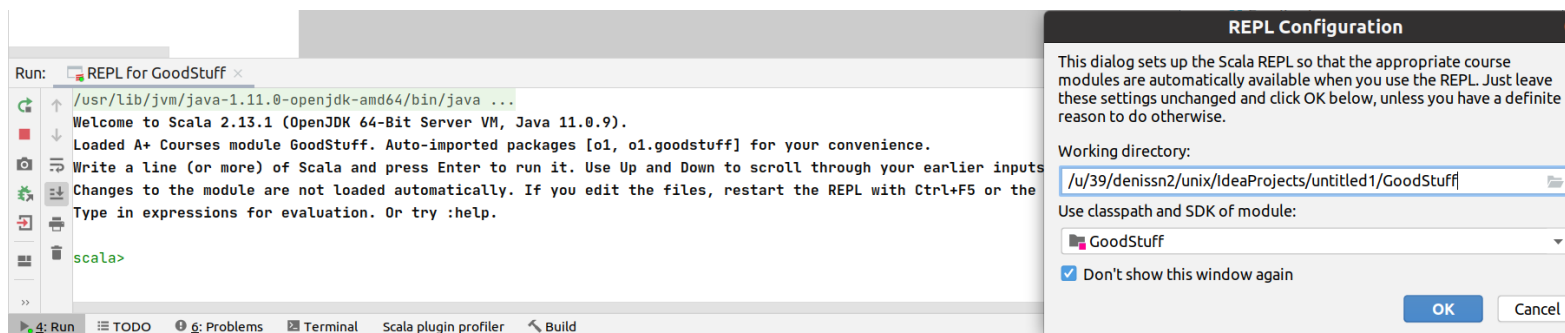


Figure 8.4: REPL and its configuration dialog.

8.5 Appendix 5. Student feedback

Feedback given for the A+ Courses plugin through A+ LMS:

Student #1

The automation around the course (IntelliJ plugin, autochecking) is the best I've ever seen in programming courses!

Student #2

IntelliJ'n A+ lisäosa on todella selkeä ja kätevä tapa palauttaa tehtäviä.

IntelliJ's A+ extension is a really clear and handy way to return the assignments [Author's translated].

Student #3

Palautus suoraan IntelliJ:stä on erittäin näppärä. Vielä kun tuo muistaisi valinnan yksin palauttamisesta.

Returning the assignments directly from IntelliJ is especially nifty. If only it remembered the "submit alone" choice [Author's translated].

Student #4

Tosi hieno systeemi tämä A+ plugini IntelliJ:hin, eipä tarvitse enää uploadata yksittäisiä tiedostoja A+:aan/TIM:iin.

This A + plugin for IntelliJ is a really great system, no more need to upload individual files to A+ (LMS)/TIM⁵ [Author's translated].

⁵tim.aalto.fi

8.6 Appendix 6. Final feedback

An excerpt from the Programming 1 (2020) course's final feedback.

"Fetching modules into IntelliJ and submitting solutions to A+ worked well."

- 0) cannot say / no comments
- 1) strongly disagree
- 2) disagree
- 3) agree
- 4) strongly agree

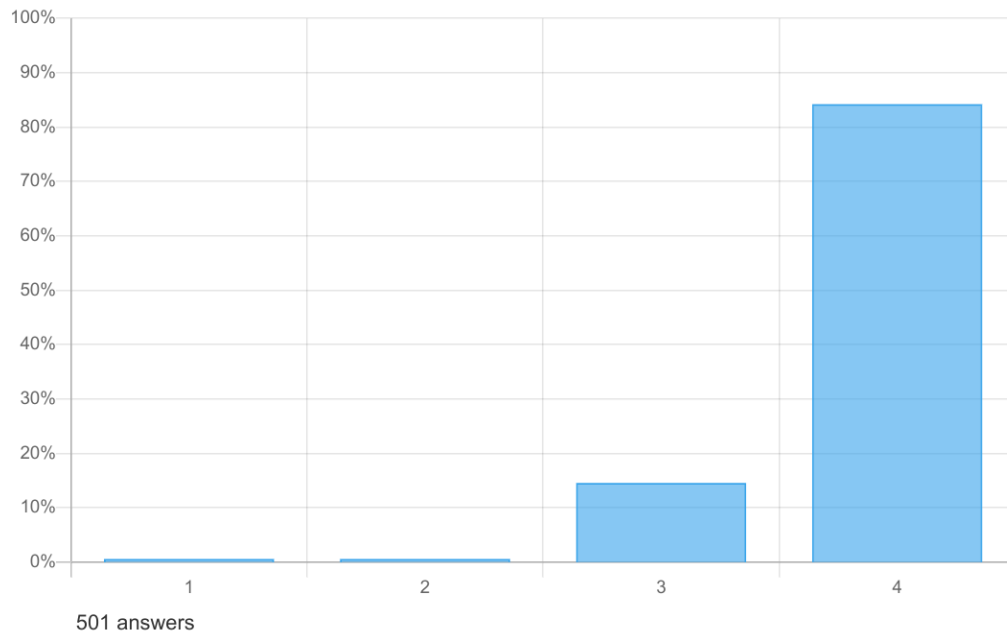


Figure 8.5: Student feedback chart.