Lappeenranta-Lahti University of Technology LUT
School of Engineering Science
Software Engineering

Roope Luukkainen

ASPA: A STATIC ANALYSER TO SUPPORT LEARNING AND CONTINUOUS FEEDBACK ON THE FIRST PROGRAMMING COURSE

Examiners: Associate Professor Uolevi Nikula
Associate Professor Jussi Kasurinen

TIIVISTELMÄ

Lappeenrannan-Lahden teknillinen yliopisto LUT
School of Engineering Science
Tietotekniikan koulutusohjelma
Roope Luukkainen

ASPA: Staattinen koodianalysaattori oppimisen ja jatkuvan palauteen tueksi ohjelmoinnin peruskurssille
Diplomityö 2020

97 sivua, 20 kuvaa, 26 taulukkoa, 2 liitettä

Tarkastajat: Tutkijaopettaja Uolevi Nikula
Apulaisprofessori Jussi Kasurinen

Hakusanat: ohjelmoinnin perusteet, Python, ohjelmointi, staattinen analyysi, abstrakti syntaksipuu, itseopiskelu, palaute.
Keywords: CS1, Python, programming, static analysis, AST, self-study, feedback.

Ohjelmoinnin perusopetuksesta oikeasta totetustavasta on argumentoitu jo vuosikymmeniä. Lisäksi keinoja parantaa opiskelijoiden oppimista ja opetushenkilöstön töiden tarkastusprosessia on tutkittu laajalti. Monet menetelmät toimivat jompaankumpaan tarpeeseen, mutta vain harvat menetelmät edistävät kumpaakin. Näistä harvoista menetelmistä suurin osa hyödyntää jonkinlaista ohjelmistoa. Moni työkalu on saatavilla vain englanniksi ja on lisäksi suunnattu vain tietylle kurssille. LUT-yliopiston Ohjelmoinnin peruskurssin opetuskieli on kuitenkin suomi. Ongelma ratkaistiin kehitettämällä uusi työkalu, ASPA (Abstrakti SyntaksiPuu Analysaattori), joka mahdollistaa palautteen saamisen sekä suomeksi että englanniksi. ASPA on staattinen koodianalysaattori, joka hyödyntää abstraktia syntaksipuuta analysoidessaan opiskelijoiden Python tiedostoja. Palaute on hyödyllistä sekä opiskelijan oppimisen että kurssihenkilöstön arviointiprosessin kannalta. ASPAn toiminnallisuuden validointi suoritettiin asiantuntija- ja testikäyttäjä-arvioinneilla sekä aiempia kurssisuoritteita analysoimalla.

ABSTRACT

Lappeenranta-Lahti University of Technology LUT
School of Engineering Science
Software Engineering
Roope Luukkainen

ASPA: A static analyser to support learning and continuous feedback on the first programming course
Master's Thesis 2020

97 pages, 20 figures, 26 table, 2 appendices.

Examiners: Associate Professor Uolevi Nikula
Associate Professor Jussi Kasurinen

Keywords: CS1, Python, programming, static analysis, AST, self-study, feedback.

For decades there have been arguments how to teach programming in the first programming course (CS1). In addition, supportive intervention methods to improve students' learning and methods to improve assessment process have been widely studied. There are various successful methods to each topic separately, but only a few of them fit for both. In general, varying software tools have been beneficial for both, but they are usually in English and dedicated for a specific course. In LUT University CS1 course with Python is lectured in Finnish. Therefore, ASPA (Abstrakti SyntaksiPuu Analysaattori) was created to satisfy needs for both Finnish and English feedback. ASPA is a static analyser tool, which utilises abstract syntax trees to detect coding convention violations from students' Python file submissions. Feedback is beneficial for the student while studying programming topics and for course staff while assessing student submissions. ASPA and its functionality were validated by expert evaluations, user testing and analysing course submissions.

ACKNOWLEDGEMENTS

I would like to thank all educational experts, teaching assistants and user testers who helped me to complete this study by participating testing sessions.

And most importantly, I would like to thank my fiancée Essi for all the support you provided me during this process.

Lappeenranta, June 18, 2020

Roope Luukkainen

TABLE OF CONTENTS

LIST OF ABBREVIATIONS

ACM	Association for Computing Machinery
AEP	Academic Enhancement Program
AST	Abstract Syntax Tree
BKT	Bayesian Knowledge Tracing
CS	Computer Science
CS1	First Computer Science course
CSV	Comma Separated Values
GUI	Graphical User Interface
IDE	Integrated Development Environment
IEEE	Institute of Electrical and Electronics Engineers
IS	Information System
IT	Information Technology
JSON	JavaScript Object Notation
LOC	Lines Of Code
MOOC	Massive Open Online Course
OO	Object-Oriented, referring to object-oriented programming
PCRS	Python Classroom Response System
PDF	Portable Document Format
PI	Peer Instruction
RQ	Research Question
SE	Software Engineering
WDF	Grades W, D and F, referring to withdraw, lowest passing grade and fail

1 INTRODUCTION

Traditionally programming has been studied at the beginning of a computer science (CS) degree (Jenkins, 2002). But for decades there have been discussions on how programming should be taught and assessed. There is no agreement on which programming language or even which paradigm should be used. (Becker and Quille, 2019; Luxton-Reilly et al., 2018). This study focuses on introductory level programming course, i.e. the first computer science course (CS1) and how to support self-study and improve feedback during the course. CS1 has been lectured in our university ever since the computer science department was established in 1986. The course has had varying implementations over the years. The current course implementation in LUT University, henceforth LUT, is based on systematic rehabilitation and two major changes.

According to our records, between 2002 and 2009 the course pass rates varied from 36 % to 68 %. During this time constant revisions to the learning process and the course contents were done, a process which is documented in Nikula et al. (2009) and Nikula et al. (2011). The major changes were an integration of an automatic code checker in 2001, and a change of programming language from C to Python in 2006 (Kasurinen, 2006). Due to the rehabilitation process and continuous improvements during the last four years pass rate has stabilised between 60 % and 70 %.

Currently, LUT CS1 can be completed twice in every academic year. The traditional version during an autumn semester and as a self-study course during a summer period, with around 550 and 100 participants, respectively. For the majority of the participants, the course is a compulsory part of major or minor studies. There are students from science and business degree programmes. The used programming language is Python and there are four assignment types. Weekly assignments and quizzes which are graded automatically as pass or fail, as well as, a course project and an electronic examination, which are manually graded from 0 to 5, 1 being the lowest passing grade.

1.1 Problem statement

Currently, the greatest threat to pass the course, is having fatal coding convention violations either in the project or in the examination. The required coding conventions of the course are aggregated into a programming guide, which is used as a reference during the lectures, tutorial videos and exercise sessions.

Nevertheless, students do not follow required coding conventions, for which several reasons are identified. Firstly, the students do not know they are violating recommended practices until course staff give feedback or student independently verifies conventions from the programming guide. Secondly, written feedback requires a lot of manual effort from the course staff, and therefore written feedback is given only to the course project and a single weekly assignment.

The second problem is possible inconsistency between multiple teaching assistants in course project grading. While there are agreed guidelines, each teaching assistant make individual decisions for overall grade. On the other hand, examinations are graded by the lecturer, which reduce the inconsistency, but causes significant workload for the lecturer.

1.2 Goals and delimitations

The primary goal is to find or implement a suitable tool to support grading of course projects and electronic examinations. The support means making grading faster, more efficient and most importantly as consistent as possible. If there is no existing tool which satisfy requirements, defined in more detail in Section 4.4, either the best tool available is modified or a new tool is implemented.

The secondary goal is to have same tool for students to assist them with their programming tasks as well as support their learning and self-study opportunities. The usage of the tool should be convenient for students who are new to programming. The tertiary goal is to have configuration options for the tool, to enable advanced selection between coding convention violation detections as well as feedback options. The research goals were used to derive formal research questions (RQ), which are explained in details in Section 3.2.

RQ1: Can a static analyser be used to standardise assessment feedback of programming assignments?

- **RQ1a:** What is a suitable static analysis method for detecting coding convention violations?
- **RQ1b:** What kind of feedback students need for their programming assignments?

RQ2: Can static analyser be used to assist students with programming assignments?

There are delimitations which are due to the nature of CS1 course implementation. Tool might be applicable for more advanced programming courses than CS1, but without mod-

ification tool will not work on other programming languages. Tool is not tested in LUT CS1 -course during the thesis study, because thesis is written in spring semester. In addition, due the time constraint of the thesis, tool is developed only to a proof-of-concept state during the thesis process.

1.3 Structure of the thesis

The thesis contains seven sections. The first section is an introduction with the background and the motivation for the study. Section 2 contains related work, including literature review and discussion of related software artefacts. Used research methodology is presented in Section 3 and in Section 4 use case scenarios, requirements and analysed data are presented as well as design and development choices derived from these.

Demonstration examples of the developed artefact are shown and discussed in Section 5, followed by four separate evaluations of the artefact. Section 6 contains result and discussion, including summary of evaluation results with the key findings. Finally, Section 7 contains conclusion of the thesis.

2 RELATED WORK

Related studies are investigated for understanding the best suitable solution for improving learning at LUT CS1 course. Related studies are ones which target to detect common problems and possible solutions on programming education, especially within CS1 context. Especially interesting are techniques which enabled overcoming learning related problems. Every CS1 course is a unique case, therefore it is not possible to generalize that all these methods are used in every course. However, it is possible to study learning related problems and examples solutions, which are often in use.

2.1 How programming is taught

Traditionally programming has been as a fundamental subject which has been studied at the beginning of a CS degree (Jenkins, 2002). Many traditional CS1 courses have physical or electronic textbook, teaching assistant mentoring in exercise sessions, paper-based or electronic examination done individually or in small groups. (Luxton-Reilly et al., 2018). In addition, on both procedural and object-oriented (OO) programming courses teaching focuses on coding and programming language itself, not an abstraction and a design (Schulte and Bennedsen, 2006). However, there has not been a consensus what to teach at CS1 courses (Hertz, 2010), even if there are many curricula recommendations, e.g. by ACM and IEEE (Joint Task Force for Computing Curricula, 2005).

According to Luxton-Reilly et al. (2018) there are self-paced, exploratory, inverted classrooms, and online course approaches on CS1 implementation. The lectured information is delivered with varying formats, and peer to peer collaboration has been utilised in many courses. Used pedagogical techniques include e.g. topic ordering, testing, exercises, leveraging students' prior knowledge, visualisation and videos. In addition, education oriented tools have been developed and existing tools are used to improve learning. They concluded that there were so large variety of curriculum implementations, that it was impossible to state one complete set to teach. In other words, there was not the one and only basis and basics of the programming. However, Shuhidan et al. (2011) noted that regardless of how programming is taught, for a novice programmer the first weeks of programming course are the most essential in order to learn programming.

According to Schulte and Bennedsen (2006) the most used languages in CS1 course were Java, C++ and Pascal, and according to (Aleksić and Ivanovic, 2016) in Europe top five

lectured programming languages were C, C++, Java, Pascal and Python. On the other hand, Goldwasser and Letscher (2008) argued that Java and especially C++ were too complex for CS1 course, while Python was suitable due to the simple syntax and possibility to use either object-oriented or procedural paradigm.

CS1 classes at university level have on average 101-200 participants (Schulte and Bennedsen, 2006), therefore, automatic code checkers, i.e. tools giving instant feedback, have been key to assessment process on larger programming courses (Choudhury et al., 2016).

2.2 Known problems

From literature six problem categories, which are the most relevant for this study, can be identified. These categories are pass rate, learning programming, self-study, assessment and feedback, grading and motivation.

Pass rate of traditional class centred computer science and software engineering courses have long been a discussed topic. Bennedsen and Caspersen (2007) conducted a global questionnaire study showing that 67 % of students pass their CS1 course. And 12 years later the result was 72 % (Bennedsen and Caspersen, 2019). Watson and Li (2014) conducted a systematic review of introductory programming literature and resulted 67.7 % pass rate on CS1 courses worldwide. Vihavainen et al. (2014) studied 60 data entries from 32 articles and prior to any new intervention technique the WDF rate, i.e. rate of student not withdrawing, getting the lowest passing grade, or a failing, was 61.4 %. The usage of WDF rate differ from other studies, but percentages are still quite close to other studies. Finally, Simon et al. (2019) gathered data from 17 universities with five-year timespan with result of 73 % pass rate.

It has been stated that learning programming is a difficult task (Figueiredo and García-Peñalvo, 2018; Karvelas, 2019; Luxton-Reilly et al., 2018; Shuhidan et al., 2011), and programming is a hierarchical set of skills, which are not learned at once, but slowly with repetitions. (Jenkins, 2002). Edwards et al. (2017) mentioned that students have often been using a trial and error strategy instead of decomposing the problem. Naturally this led to result, that commonly many errors occur before students succeed to create a working program. On the other hand, Bruce et al. (2001) noted that in some cases, the concept or structure itself was not actually the problem, but available techniques were hard for novices. However, Keuning et al. (2017) found that novice programmers focus on passing the tests and getting program output correct, not making good code. In addition,

Jenkins (2002) argued that not all students wanted or were able use all possibilities to get help during the course.

Figueiredo and García-Peñalvo (2018) pointed out that university courses are designed for local students via local in-class activities instead of remote student. In addition, to avoid misunderstanding programming concepts teaching assistant explanations are needed (Choudhury et al., 2016; Chow et al., 2017). However, Smith et al. (2018) mentioned that there also remote students, who often are part-time students with very limited time available.

The formative assessment help student and teacher keep track of students learning, while summative assessment is a test of learning Simon et al. (2012). On the other hand, it can be hard to measure, if student has actually learned programming (Tew and Guzdial, 2011). Nevertheless, to succeed in cognitive learning, students, i.e. learners, should receive enough formative feedback to deepen the knowledge and understanding (Choudhury et al., 2016; Chow et al., 2017). In addition, Gerdes et al. (2010) noted that the student might not remember the problem, if feedback is too late. However, getting formative or summative assessment feedback reasonably fast requires a usage of computers (Tempelaar et al., 2013), or it should be completely automated (Ju et al., 2018). But Liu and Petersen (2019) pointed out that e.g. Python interpreter gives feedback which is technical and does not provide guidance to fix an error. Likewise Becker (2016) and Becker et al. (2019) found that compiler and interpreter messages are considered hard for the novice programmers without sufficient set of background knowledge.

In an interview study by Kinnunen and Malmi (2006), main reasons to drop out CS1 were lack of time, lack of motivation and difficulty of the course. Background reason creating such feeling were high-stress, lack of help and unsatisfactory scores from assignments. A subsequent survey found that "I did not get enough help" was the seventh common reason from all 22 reasons and the third common reason when workload and time related issues are excluded. (Kinnunen and Malmi, 2008).

2.3 Solution strategies

While there are identified problems, also possible solution methods have been studied. As previous section studies showed by providing channels to get help for assignments, student could be less demotivated and would more likely pass the course.

From literature five solution categories, which are most relevant to the previously discussed problem categories, can be identified. These categories are adding intervention, modifying assessment and feedback methods, modifying course assignments, modifying infrastructure or curriculum, and platforms and tools. Categorised findings are shortly discussed in following subsections. At the end there is a short synopsis of solution strategies with their topic of the interests.

2.3.1 Adding intervention

Studies with pre-test and post-test with a new intervention method have shown that on average, a new intervention method will improve the pass rate. Vihavainen et al. (2014) have done comparison of the effects of different methods. They compared methods with realized improvements in WDF-rates before and after the methods was utilised. By realized improvement they mean "the absolute improvement divided by the potential, by which the room for improvement that varies between different institutions is taken into account". The change was from 38.4 % to 25.6 %, i.e. 33.3 % of realized improvement.

Cukierman et al. (2019) studied the effects of Academic Enhancement Program (AEP) between two groups, one using it and another not using it. The AEP includes varying intervention activities improving students' academic skills and overall wellbeing. They got an increasing result between pre-test and post-test. Group with the AEP got 7.9 % increase, while without the AEP students increased scores only 3.0 %.

Porter et al. (2011) utilised intervention of peers, when they tested feasibility of peer instruction (PI) method. In two advanced level CS courses, PI was added to deepen students understanding about CS concepts. By students group discussion was seen valuable by 80 % and 86 %, while only 44 % on both courses though that group member knowledge was key factor of value. Weighted learning gain calculation result was as high as 89 % and 85 % of potential students learned. Zingaro and Porter (2014) reported similar results from CS1 course, with Python 3, exam results. Students were learning better with PI and PI results were collating with exam results.

Olsen and Fox (2019) tested a new "buy a hint"-technique on Software Engineering (SE) course to tackle problem of cascading penalties in assignments with multiple subsequent parts, as well as, to help weaker students to get at least some points. Contrary to the hypothesis hints did not help weaker students.

2.3.2 Modifying assessment and feedback methods

Ju et al. (2018) argued, that current trend is to utilise autograders as a part of formative feedback for weekly take-home assignments. Examinations are still mainly traditional individual tasks in a classroom, without any resources, but only couple hours of time. On the other hand, Shuhidan et al. (2009) compared and categorised students' answers to evaluate their examinations and understand in which topics student needed most help. For categorisation BLOOM taxonomy and SOLO taxonomy were used.

Choudhury et al. (2016) noted, that rapid feedback, e.g. from Intelligent Tutoring System (ITS), improved novice student's code quality starting at the beginning of learning. Therefore, it will be permanently better at the end, i.e. while doing programming work for the software industry. However, as Liu and Petersen (2019) mentioned it is crucial to think how the feedback is provided to the students. Their study showed that even when optional help was available only in 28.6 % of cases user checked it. Frequent users of optional output from PyTA tool achieved better scores than average user, and average user performed better than infrequent user.

Figueiredo et al. (2019) studied predictive neural network model to detect which students will be in trouble later during the course. This method was reported to be extremely efficient by categorising only three students out of total 85 incorrectly. Fu et al. (2017) had similar idea to provide tool for teacher to detect students who are struggling. They compared students programming time and number of compiling errors fixed. Real-time tool LAPLE has a collection of the most common errors which can be then utilised for new exercises and other lecturing material.

2.3.3 Modifying course assignments

When assessment and feedback is not enough, the whole assignment could be changed. Allen et al. (2019) moved from one large program to many small programs (MSP). This led to happier students with less stress, even though they did more assignments than before.

Navrat and Tvarozek (2014) tested an online learning environment for submitting weekly assignments, which was used to increase motivation by showing rankings such as total number of solved assignments and fastest running times per algorithm assignment. Results were used to predict final grade from weekly assignments. Grade prediction was

reported to be feasible replacement for course examinations. Smith et al. (2018) created an open summer camp to support existing programming skills. Camp had daily puzzles with infinite submissions and the best counted, to encourage trying. Student were reported to be very interested in opportunity to develop skills and understanding.

Ju et al. (2018) argued, that it is more reasonable to require students to do examination coding in natural environment, which contains elements such as writing programs with IDE, possibility to test them and access to materials. Similarly Olsen and Fox (2019, p. 839) said: "in paper exams students are flying-blind". Ju et al. (2018) continued, that it is due the limitation of paper exam grading is even harder on paper exams. In-class coding-based summative assessment with third-year software engineering students was successful for course staff and students were reported to be satisfied as well. A year later also Olsen and Fox (2019) found the identical exam environment successful on advanced SE course.

2.3.4 Platforms and tools

Already in 1988 tool called Ceilidh was used to automatically assess programming exercises (Higgins et al., 2005). A decade later it was replaced by another tool, CourseMaster, later called CourseMarker (Foxley et al., 2001). Systematic literature review by Luxton-Reilly et al. (2018) showed that between 2003 and 2017, a total of 265 studies focused on tools and their utilisation as part of introductory programming. From 11 categories the most popular category was *Learning programming*.

Higgins et al. (2005) studied CourseMarker in varying programming courses. When they combined CourseMarker's feedback with possibility to submit three times instead of one, 94 % of the students got consistently better grades. Total improvement from first to last submission was 63 % on average and the tool saved hundreds of working hours from academic staff without lowering the feedback quality. Garg and Keen (2018) studied a tool called Earthworm, which give suggestions to decompose function to reduce complexity, i.e. refactoring feedback. Tool searches simpler solution based on cyclomatic complexity, which is calculated from control flow graph.

Autograder tools, e.g. zyBooks, have been utilised to enable e.g. an automated assessment, an effective usage of multiple submissions per assignment and occasionally code templates (Allen et al., 2018). Allen et al. (2019) combined it with previously mentioned MSP, which led to reduced student's stress and better scores on subsequent CS course.

Croft and England (2020) conducted a study showing that Moodle CodeRunner can be used very successfully at introductory course with Python 3 and at the first object-oriented course with C++14. Both courses had students with varying backgrounds and prior programming knowledge was not required. Usage of the CodeRunner increased average exam grade by 7 % and pass rate by 10 %. In addition student satisfaction to assessment and feedback was increased by 7 %.

Sorva et al. (2013) studied 46 visualisation tools and concluded that there are multiple visualisation systems built to solve various problems. However, these systems tended to be research prototypes with a short life span, but e.g. ViLLE system has been in use since year 2005. (Kaila et al., 2018). Annamaa (2015) studied another successful visualisation tool, an educational Python IDE Thonny, which uses visualisations to show code structure for students.

Guo (2013) created a purely web-based solution, Online Python Tutor, to visualise Python 2 and 3 programs. System used a standard Python debugger module, bdb, and execution trace to visualisation and sandboxing to prevent malicious usage. Zingaro et al. (2013) utilised Online Python Tutor and created a web-based tool called Python Classroom Response System (PCRS), which enabled effective and extended usage of iterative four step PI method. PCRS supports interactive reading and writing code, lecturer feedback and questionnaires.

For first-year functional programming course Gerdes et al. (2010) created non-test-based strategies and program transformation to functionally assess students' Haskell programs, as well as, give explained correction guidance. Authors argue test-based method has various problems, such as test coverage and missing detection for good conventions. They used methods like strategy language and lambda calculus to check correctness, good conventions and design of the program. The tool reduced teacher's workload from 94 programs to 8 programs.

Chow et al. (2017) argued, that software tools can also create problems, e.g. grading and tutoring systems check program code against test cases do not usually give guidance to fix detected errors. Keuning et al. (2017) noted another problem, tool or IDE cause confusion for novice programmer, because options or feedback were too advanced.

There are great number of tools developed to reduce aforementioned problems. Therefore, more detailed look of recent and relevant tools as well as platforms utilised in Finnish Universities is done in Section 2.4.

2.3.5 Modifying infrastructure or curriculum

Changing programming language in CS1 course from Java to Python did improve learning of programming concepts (Koulouri et al., 2015, pp. 23-24). Goldwasser and Letscher (2008) mentioned that the change of the language might have a significant effects in curriculum, because differences between programming languages and paradigms. However, authors did manage to move from C++ and Java to Python in CS1 course, but still managing to keep object-oriented paradigm.

Lawhead et al. (2002) and Fagin and Merkle (2003) considered Lego Mindstorms Robots as an option to learn programming. The aim of Lawhead et al. was to tackle problem of abstraction code objects. Therefore, they gave concrete objects, i.e. Lego Mindstorms Robots, for novice programmers to help perceive and understand objects oriented programming. In addition, they stated that using robots is not a new and unique idea in 2002.

Fagin and Merkle (2003) did comparative study between student groups using Lego Mindstorms Robots and groups using traditional lab sessions. Both groups used Ada language for programming, but robot group used API which translates and assembles code for Lego Mindstorms RCX module. When groups were compared with midterm exam scores, final exam scores and class rank, non-robot group achieved better total scores on all measured scales. However, robots were described as interesting, fun, challenging and relevant in the feedback session. Nevertheless, robots did no effect to which major students selected after the course.

2.3.6 Synopsis of the solution strategies

Synopsis of the solution strategies are shown in Table 1. The first column contains categorised methods. The second column briefly summarises the interest area in the study. The third column contains reference to the studies.

Used method categories are slightly more detailed than section headings. From the table we can see that there are 12 different tools – marked with superscript 1, and two platforms – marked with superscript 2. However, all tools and platforms are located under other topics to highlight a topic of interest.

Table 1. Synopsis of the solutions methods.

Method / Method category	Topic of interest	Reference
New intervention method		
Peer instruction (PI)	Support students who do not learn from lectures	Porter et al. (2011)
Web-based tool – Online Python Tutor[1]	Improve visual teaching of Python 2 and 3	Guo (2013)
Buy a hint in summative assessment	Pass rate, cascading penalties	Olsen and Fox (2019)
Academic Enhancement Program for CS students[3]	Students' academic skills, overall wellbeing	Cukierman et al. (2019)
Infrastructure modification		
Roadmap for Lego Mindstorms Robots	Concretize abstract topics with robots	Lawhead et al. (2002)
Programming with Lego Mindstorms Robots	Students' interest towards CS and SE	Fagin and Merkle (2003)
Python Classroom Response System[1]	Enhance PI in CS courses	Zingaro et al. (2013)
New online learning environment[1]	Student motivation and test feasibility to grade based on weekly assignments	Navrat and Tvarozek (2014)
Remote summer camp for remote students[2]	Support remote students to improve existing skills	Smith et al. (2018)
Moodle CodeRunner environment on introductory programming course[2]	Pass rate, exam grade and student satisfaction	Croft and England (2020)
Curriculum modification		
Moving from C++ and Java to Python	Reduce programming language complexity to enable focus on OO concept	Goldwasser and Letscher (2008)
Moving from Java to Python	Reduce complexity of programming language	Koulouri et al. (2015)
Assessment and feedback modification		
Automatic assessment tool Ceilidh[1] and its successor CourseMarker[1]	Establish automatic assessment of assignments	Higgins et al. (2005) & Foxley et al. (2001)
Various non-test-based static checks[1]	Detect common student mistakes and give correction guidance	Gerdes et al. (2010)
Educational IDE, Thonny[1]	Visualising code structure	Annamaa (2015)
Intelligent Tutoring System (ITS)[1]	Faster feedback to improve code quality	Choudhury et al. (2016)
Cyclomatic complexity based feedback tool Earthworm[1]	Support students with code complexity and problem decomposition	Garg and Keen (2018)
zyBooks autograder tool[1]	Students' high stress, poor performance, and negative evaluations	Allen et al. (2018)
Static analysis tool PyTA[1]	Effect of an optional assignment feedback	Liu and Petersen (2019)
Assignment modification		
Reasonably natural examination environment	Effects of assessment environment	Ju et al. (2018)
Change course assignments to MSP	Student motivation and pass rate	Allen et al. (2019)
Analyse and predict student performance		
Categorise student answers with BLOOM and SOLO taxonomies	Identify the hardest topics for students	Shuhidan et al. (2009)
LAPLE, the real-time log-based support system for C language[1]	Detect students in trouble, create effective education material	Fu et al. (2017)
Predictive neural network[1]	Predict who will need help, focus feedback	Figueiredo et al. (2019)

[1] A tool

[2] A new platform

[3] Both infrastructure modification and new intervention method

2.4 Competitive analysis

The goal of the study is to find a solution for supporting self-study as well as increasing assessment quality of teaching assistants and lecturer. Therefore, effects on formative and summative assessments as well as suitability for student usage are compared from studied literature. Based on those the most optimal solution is to utilise a tool or a platform. A dedicated tool or platform could significantly increase quality, consistency and throughput of the assessments.

To select the most optimal tool type, different types were compared based configurability and successful use cases. Based on selected factors a static analysis tool would be suitable. The more detailed explanation of design choices is presented in Section 4.6. To select between a tool, a platform or both, existing static analysis tools as well as programming education platforms used in other Finnish universities are investigated. Furthermore, option of developing a new tool, instead of utilising existing one, is considered based on further analysis.

2.4.1 Static analysis tools

Edwards et al. (2017) mentioned that static analysis is used to detect errors on syntactically correct programs, therefore naturally static analysis tool, i.e. static analyser, is a software program designed to do so. Static analyser do not run programs, they do analysis by examining the source code. Authors did static analysis in CS1, CS2 and CS3 courses with Java, by using existing open-source tools Checkstyle and PMD. In over a million analysed files, the most common individual errors were missing documentation, wrong indention and extra whitespaces. However, while student were decreasing amount of errors nearly 85 % from initial submission to final submission, the distribution of error types remained the same regardless of number of errors.

Chow et al. (2017) developed ITS system using AST to detect differences of unsuccessful attempt to successful attempts to generate a code-based hint. ITS was used to generate input hints, concept hints and pre-emptive hints. The data used in the study was from Grok learning platform (Grok Learning, 2020), which is an educational platform with varying programming languages and difficulty levels.

Yulianto and Liem (2014) compared source code analysers and further tested five of them. Different tools were designed to detect different things, such as bugs, flaws and violations

of coding conventions, but none of them detected all of them. Therefore, with both artificial test files and students' programs, all the errors were not detected by a single tool. Tested tools were Cppcheck for C and C++ languages, PMD and Checkstyle for Java, and PHPCheckstyle and PHPMD for PHP.

Rogers et al. (2014) studied an Automatic Coding Composition Evaluator (ACCE), which is used for automatic code reviews. It aims to reduce bad programming conventions among students, improve assessment consistency and expedite the assessment process, by giving highly detailed and targeted feedback. Tool uses AST to get edit distances of all solutions and clustering to group similar solutions in common clusters, which were visualised with open source program Gephi (Bastian et al., 2009).

Choudhury et al. (2016) and Wiese et al. (2017) studied AutoStyle tool which uses AST and clustering to detect similarities to give syntactic hints, approach hint and code skeletons. Automatic feedback saved workload of 40 teaching assistants on thousands of submissions. With hints 70 % students achieved the best solution, while only 13 % of a control group did the same. Approach hint, a code skeletons and syntactic hint were most relevant for weaker, average and advanced student, respectively.

Liu and Petersen (2019) created PyTA which is used to detect various coding errors, which are common among CS1 students. This tool was tested on previously mentioned PCRS platform. Data of 750 students were compared to 504 students from previous year and students with PyTA needed less submissions to fix individual errors and to pass an assignment. In addition, among students using PyTA, infrequent user needed over 25 % more submissions than frequent user.

In general, AST is such an interesting and useful topic that there is symposium for it, Static Analysis Symposium, which have had e.g. studies for data flow analysis (Rapoport et al., 2015) and string manipulations in C programs (Journault et al., 2018).

2.4.2 Platforms in Finnish universities

In Finland there are 14 universities. While introductory programming is taught on many of these, there is no single countrywide platform, tool, environment or technique, henceforth platform, to do so. From all the found platforms seven were selected for further analysis. Selection was done based on two factors. Platform is used to teach CS1 and has automatic grading capability. The second factor is usage in more than one universities, because they

are therefore assumed to be suitable for general usage not just for single university with specific goals. Selected platforms are studied to see general outlines and comparison what they are capable of. In addition, comparison exclude all 24 universities of applied sciences in Finland. Studied platforms are listed below, henceforth only short name or abbreviations are used.

1. Viope
2. ViLLE, ViLLE - Collaborative education tool
3. Lovelace
4. A+ LMS, A+ Learning Management System
5. TRAKLA2
6. TIM, The Interactive Material
7. TMC, Test My Code - Programming assignment evaluator

In this study Viope platform is used as a reference platform, because it is used for programming task submissions in LUT University. Viope platform have been developed by Viope (2020). Viope can be used for various types of assignments, such as quizzes, mathematic questions and programming tasks. In addition, it possible to add reading material, forums, tutoring questions etc. teaching material. For teacher, there are possibilities to add, modify and remove exams and assignments as well as observing statistic about courses and individual students. For student, there are possibilities to see open assignments, personal assignment progression and upcoming deadlines. Each assignment submission can be tested, and when all tests are passed it can be submitted for grading. In addition to LUT university, Viope is used in several universities of applied sciences for various degrees, even if they are not covered in this study.

ViLLE platform has been developed in University of Turku (UTU) (Kaila et al., 2018; Rajala et al., 2020). Platform is used in UTU and in variety of lower level educational institutions. ViLLE provides a learning environment for multiple subjects such as programming, mathematics and languages. This is possible with more than one hundred exercise types. ViLLE can be used to record course activities and keep track on lecture participants.

Salminen (2020) platform has been used to teach various programming courses in university of Oulu. Documentation of usage elsewhere was not found, but most of the material is fully available to anyone interested. CS1 course with Python has various assignment types to teach programming, such as multichoice, answer fields and file submissions. Platform also provides great amount of theory material and resources used on the course.

Karavirta et al. (2013) introduced an open source learning management system A+, henceforth A+ LMS. It is open platform used for automatic and semiautomatic grading in programming courses. In addition, the system stores exercises, their submissions and feedback stored. A+ LMS can be used with multiple programming languages, e.g. Python, JavaScript, Scala and HTML. A+ LMS has a dedicated grader for assignments and eleven other smaller components and the whole system is complementary to Moodle (A+ LMS, 2019). In addition to Aalto University, A+ LSM is used in Tampere University, as a local application called Plussa (Niemela and Hyyro, 2019).

Malmi et al. (2004) developed an online learning environment TRAKLA2. In addition, TRAKLA2 has been studied e.g. by Hakulinen et al. (2013). In general TRAKLA2 is used to improve teaching of algorithms and data structures by providing visualisation for them. TRAKLA2 can generate pseudocode-based assignments automatically for each of supported algorithms, e.g. binary trees, many sorting algorithms, graph algorithms, etc. and submissions are automatically graded. For teacher overall trends of the course are available, but individual student statistics are not. For student it provides theory of algorithms, visual simulation of algorithms and possibility to test algorithms with much less abstraction than previously. Also instant feedback, achievement badges, model answers and course progression are available.

TIM (2020) has been developed and used in University of Jyväskylä. Idea of the TIM is to embed all the course materials, i.e. theory, assignments, notes, etc., in single interactive page. Therefore, TIM can be used and is used for various levels and study programs such as languages, mathematics and programming. On programming field, it is used to teach e.g. SQL databases and R programming. TIM support multiple assignment types and can provide a feedback that something is wrong in submission. It also shows expected value and submission value, e.g. programming tasks are tested by unit tests and if the unit test fails due to the erroneous submission, the line number of the error is given with predefined feedback.

TMC (2019a) and TMC (2019b) explain TMC platform in general. TMC has been developed and used in Helsinki University, in addition it is used for Finnish MOOC courses and in dozens of other organisations. TMC is a set of multiple smaller tools, similar to A+. Integration to NetBeans IDE is common way to use it. This allows test code directly in IDE with predefined tests. TMC gather variety of data which is used for research purposes.

3 RESEARCH METHODOLOGY

This section describes the chosen research methodology in details and justifications behind the choice. Multiple research methodologies have been developed and studied on information system (IS) and information technology (IT) fields. A subset of these methodologies are considered. Based on practical suitability and the goal of the thesis, which is to implement a software artefact. Artefact implementation a major part of the design science. "The result of design-science research in IS is, by definition, a purposeful IT artifact created to address an important organizational problem" (Hevner et al., 2004, p. 82). Therefore only research methodologies related to design science are considered.

One of the early days methodology is proposed by Nunamaker et al. (1990). Authors proposed a multimethodological approach which had following four stages: theory building, experimentation, observation and system development. Hevner et al. (2004) created a concise conceptual framework with seven guidelines. Guideline titles are design as an artifact, problem relevance, design evaluation, research contributions, research rigor, design as a search process, communication of research. Wieringa (2009) changed this framework with three modifications. The first was emphasis to business need elicitation using regulative cycle. The second modification was layering the knowledge base, and the third was utilising a nested problem structure. The nested structure had knowledge questions and practical problems as major classes. The research resulted a new collection of eight guidelines.

Peffers et al. (2007) introduced a design science research methodology (DSRM) and a process model for it. The DSRM process model consists of six steps which are problem identification and motivation, objectives of a solution, design and development, demonstration, evaluation and communication. The DSRM improves earlier methodologies as Geerts (2011, p. 143) noted "the DSRM aims at improving the production, presentation, and evaluation of design science research while being consistent with the principles and guidelines of design science research established in previous research studies". The previous studies refer e.g. Hevner et al. (2004).

Another branch of design science research is represented by study of Baskerville et al. (2009). Authors introduced a soft design science methodology which combined two earlier methodologies. The design-build artefact-evaluate process and the iterative soft systems methodology. The soft system methodology added a social aspect for the design-build artefact-evaluate process.

From these methodologies and guidelines the practically most suitable one is selected for the thesis. Pure guidelines are not as practical as process models. The soft design science methodology is focusing on improving the human organisation, which is also a case in this study as form of a university course. However, the thesis is focusing on creating the artefact not yet testing it with students due the time limitations discussed in Section 1.2. Therefore this thesis study is conducted by the DSRM process model, introduced by Peffers et al. (2007). The adapted presentation of an original DSRM process model is shown in Figure 1.

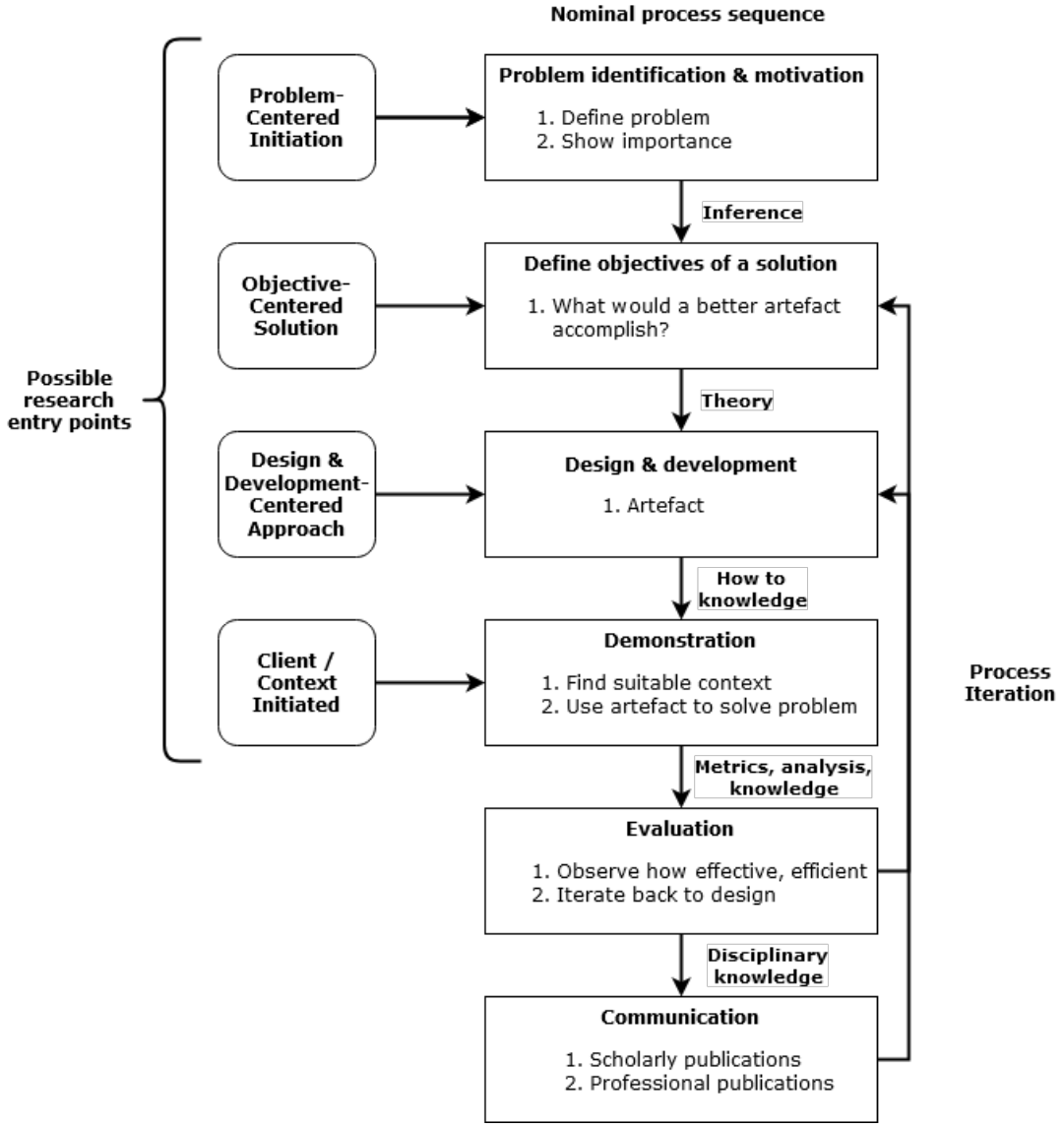

Figure 1. DSRM process model, presentation adapted from Peffers et al. (2007).

3.1 Design science research methodology

The DSRM process model is used as a baseline model. The order of presented steps is nominal and there are multiple possible entry points. In this study the objective centred solution approach is used. The adapted process model used in this study is shown in Figure 2. Steps are executed parallel with each other, and back and forth transitions between steps are done when needed.

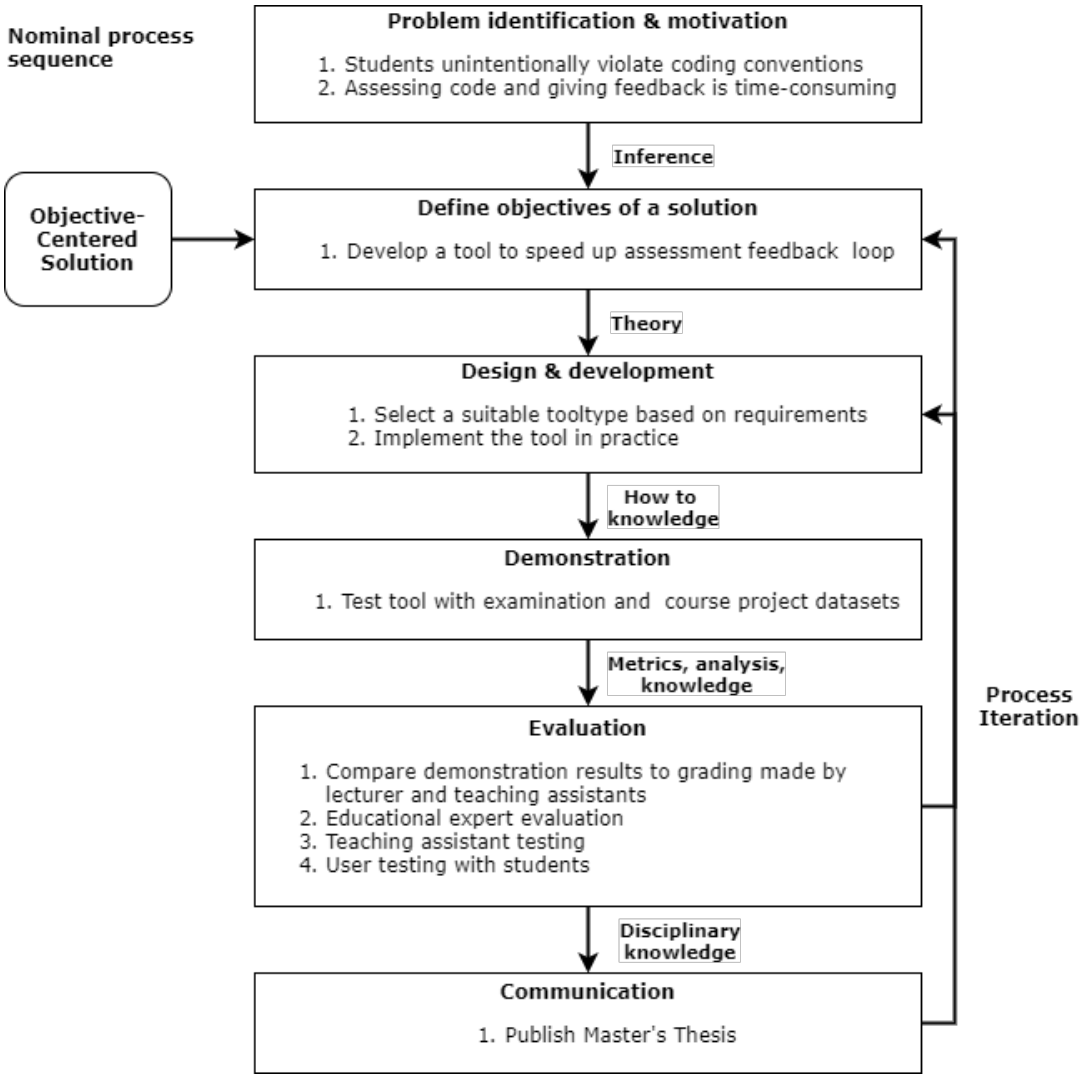

Figure 2. Adapted objective-centered solution of DSRM, which is used in this study.

The identified problems are related to the taught coding conventions and assessment feedback given to the students. In many cases students are unintentionally violating recommended conventions. They might not be aware of conventions or how to utilise conventions in their solutions. The primary way to get confirmation is by checking recommended

conventions from the course material, such as written programming guide, lecture slides and tutorial videos. The secondary way is via feedback given by the course staff, and even if it is the secondary option it requires a lot of human effort. In addition, feedback is not always consistent between teaching assistants. These problems motivate a research of an artefact, i.e. a tool or set of tools, for an improved feedback.

The objective is to solve aforesaid problems, which in practice it means developing an artefact which supports self-study and enables an improved and more standardized assessment feedback. Based on literature review, the self-study support can be e.g. tests with automated feedback and students can run them independently. On the other hand, the possible assessment feedback improvements can be speeding up feedback and grading processes, removing human errors and increasing a quality of the feedback.

The design of the artefact is based on academic literature, course-specific objectives and two datasets from LUT CS1 course, which are analysed. The first dataset is an electronic examination dataset and the second dataset contains rest of the course's programming assignments, i.e. weekly assignments as well as a course project. Functional requirements are elicited from information gathered by literature review, objectives and other course-specific requirements given by the lecturer of the course. Non-functional requirements and constraints are derived from selected technologies, datasets and assumption that most of the end users will be novice programmers. The literature review process is presented in Section 3.3, literature review in Section 2 and the elicited requirements are listed in Section 4.4. These are the foundation for the selection of tool type and development of the artefact. All the design and development choices are presented in Section 4.

The artefact functionality is demonstrated with previously mentioned datasets. Demonstration means detecting coding convention violations from course project data as well as from electronic examination dataset. Primary detection method is a static analysis, which allows detection of predefined recommended conventions and solutions violating them. Detected coding convention violations should be displayed to the student clearly enough. Therefore, based on literature the most suitable feedback format is selected as a part of design choices. The demonstration of the artefact is presented with examples in Section 5. Course staff creates another user group for the artefact. They will be using artefact for grading student submissions. Therefore, the possibility to use analysis results as basis for automatic or semi-automatic grading is tested. Furthermore, analysis combined with configuration options should enable possibility to focus on particular practices, e.g. new practices learned during a single week. Therefore, different configurations are tested too, but only the most suitable one will be used.

Evaluation of the artefact is done with four methods. Initially, static analysis results from the demonstration step are validated against existing feedback, because submissions are already graded by teaching assistants and lecturer. After result validation an educational expert evaluation and a teaching assistant testing are used to evaluate the artefact performance, and result correctness. In other words, while result validation is based on the effectiveness and efficiency of the artefact, the expert evaluation will focus more on practical perspective. For example, understand how artefact affects each user groups workflow, as well as, identify possible problem in order to refine the artefact. Furthermore, test subjects which are not testable using datasets are evaluated based on literature, expert evaluation and user testing, which is done by former CS1 students. The artefact evaluation methods and results are presented in Sections 3.4 and 5, respectively.

Communication step is related to future research and thesis publication. The discussion of future work, including future research and future development of the artefact, is presented in Section 6.3. The first step of the publication process is applying for imprimatur, and when the publication licence is granted, the thesis is published in the LUTPub open access database.

The overview of actual timeline is shown in Figure 3, which includes all the nominal process steps. The design and development step was divided into two major sections, and the fist section contained design of the whole solution, development of analyser with all the modules, the GUI and the configuration possibilities. The second section contained bug fixing and designing the future development based on evaluations.

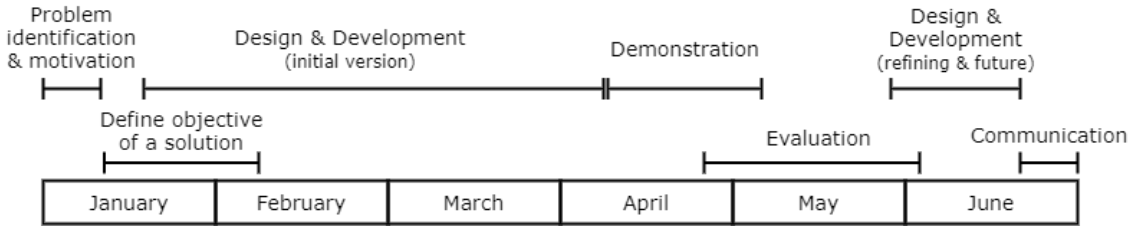

Figure 3. Overview timeline for DSRM process steps.

3.2 Research questions

The research in this study is done to answer to two main research questions. Both questions are related to the designed artefact and its utilization. In addition, there are supportive subquestions, which are answered before the main question. RQs are as follows:

RQ1: Can a static analyser be used to standardise assessment feedback of programming assignments?

- **RQ1a:** What is a suitable static analysis method for detecting coding convention violations?
- **RQ1b:** What kind of feedback students need for their programming assignments?

RQ2: Can static analyser be used to assist students with programming assignments?

The RQ1 focus on course staff point of view, i.e. how could a static analyser tool be utilised to do a part of or the whole grading process. This has two major components, the first being a usage of the static analyser, i.e. the RQ1a is a practical question focusing on actual implementation, technologies and usage of the static analyser tool. The RQ1b contains a theoretical and a practical part. The theoretical part is focusing on selection of feedback format and how it is presented to the user. The practical part focus on evaluating selected format and refine it based on feedback. The whole RQ1 is combining the static analyser with the assessment feedback, i.e. with what technology coding conventions from student submissions can be detected and how to gather them to convenient format for course staff.

The RQ1, RQ1a and RQ1b are answered by combining results of four different methods. Firstly, the best practices are searched from the literature and secondly by studying subset of tools described in the competitive analysis section, Section 2.4. Thirdly the artefact is implemented and tested with real dataset and finally educational experts and teaching assistants evaluate suitability of the tool.

The RQ2 is focusing on student users point of view, i.e. how could they utilise tool while programming their weekly assignments. Similarly as a static analyser detects coding convention violations for course staff, it is studied if student could benefit from the feedback also. Therefore, answers of RQ1b is being utilised in RQ2, but the RQ2 is answered mainly based on the analysis result comparison and the user testing.

3.3 Literature review methodology

The initial search terms used in the literature review are shown in Table 2. In the initial searches two databases were used – Association for Computing Machinery (ACM) and Institute of Electrical and Electronics Engineers (IEEE). All the search terms were limited to the abstract and for the highest resulting searches year range is used to limit results.

From the results suitable studies were selected for each topic based on the abstract and manual overview of the study to evaluate its relevance. Around 10 most suitable studies for each topic were selected.

In addition, references from selected search results are used when needed, i.e. as the academic manner require the original academic literature is referred when possible. The supplementary searches for the other than literature review section were made based on new keywords from initial references. For the supplementary searches ACM, IEEE and Google Scholar databases were used.

Table 2. Initial literature review searches.

Search term	Since	ACM	IEEE
abstract AND syntax AND tree AND static AND analy* source AND code AND static AND analy* AND python	2009	26	123
code AND style AND cs1	2010	20	895
computer AND science AND education AND error AND detection	-	106	32
software AND engineering AND education AND error AND detection	-	51	35
computer AND science AND education AND cs1 AND curriculum AND python	-	35	0
summative AND assessment AND programming	-	55	63
formative AND assessment AND programming	2010	292	98
style AND guide AND python	2010	37	1

3.4 Evaluation methodologies

The created artefact is being evaluated by four different methods. The first method is to compare analysis results of artefact to human grading feedback. The second and third methods are an educational expert interview and teaching assistant interview. These method are conducted with multiple expert groups to gather expert feedback from experts with varying backgrounds. The experts are only asked to participate the interview session, i.e. no prior or post actions are required.

The fourth method is to test the artefact in practice by conducting an user testing, i.e. it focus on student users point of view. All four evaluations methods are discussed with details in following subsections.

3.4.1 Result comparison methodology

The result comparison uses the same datasets as used for the artefact demonstration and the assessment feedback is taken from Moodle. Result comparisons is done manually by comparing the feedback given by a human grader to feedback given by the artefact and every clear difference is verified by checking the analysed code. enabled detection of code structures which are not detected by the artefact as well as verification of humane errors in assessment feedback.

Because both feedback and student submission datasets contain only achieved data, a goal of this method is to verify functionality of the artefact before other evaluations. Used datasets are explained with general analysis in Sections 4.1 and 4.2, while the results of these systematic comparisons are presented in Section 5.3.

3.4.2 Educational expert interview methodology

Wohlin et al. (2012, pp. 62-63) divided interviews to three categories based on their structure. In addition, they suggested dividing interview into phases. These were used as general guidelines while planning the interview structure. The educational expert interview is a semi-structured interview with multiple phases. Interview is conducted as an two hour online meeting where the artefact will be demonstrated to the participants via shared screen.

Interview has a planned structure and predefined questions. However, to enable free discussion and focus on relevant topics which experts are the most interested in, questions are not asked in strict order. The planned structure of the educational experts interview session is shown in Figure 4.

3.4.3 Teaching assistant testing and interview methodology

Teaching assistant testing is conducted by two teaching assistants, TA1 and TA2. Both teaching assistant have separate testing sessions, but the material and the task are identical. General instructions, the artefact and anonymous submissions are given to each teaching assistant before the testing. After the testing session both of them are interviewed. The full structure of the testing and interview session is shown in Figure 5.

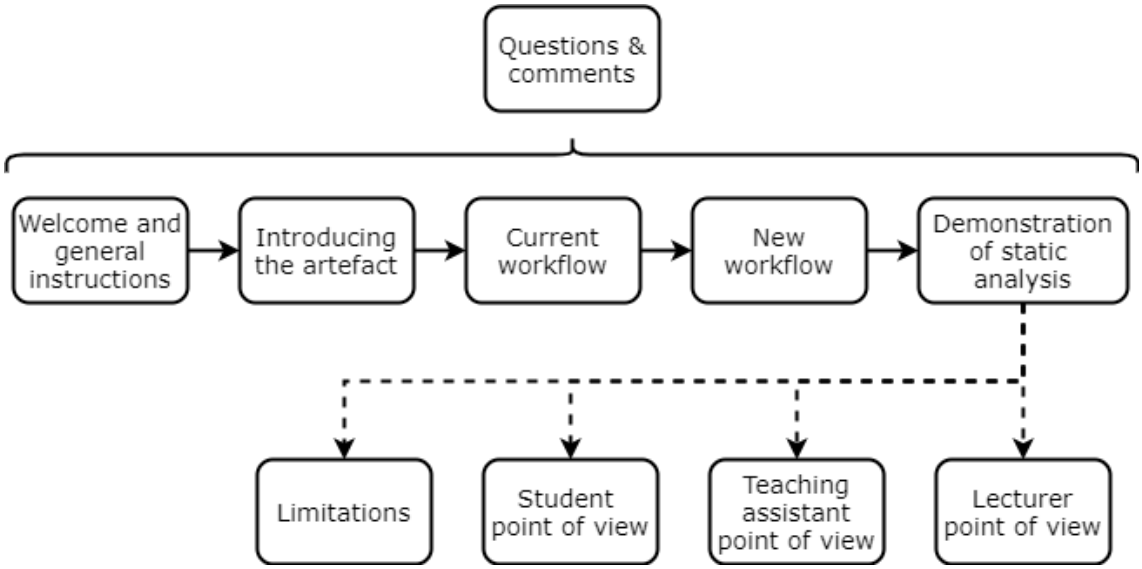

Figure 4. Structure of the educational experts interview session.

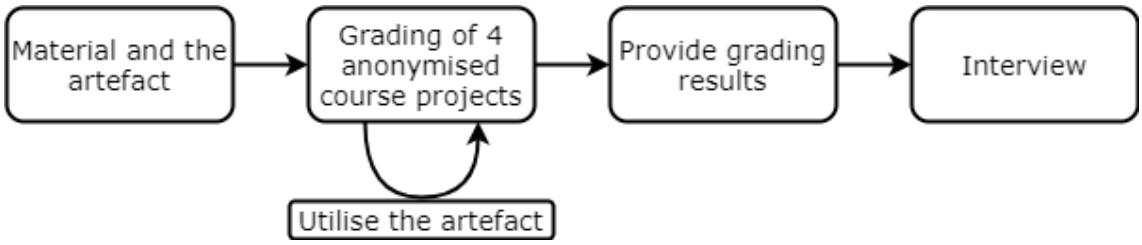

Figure 5. Structure of the teaching assistant testing and interview session.

A testing task is to grade four anonymised course project submissions and use the artefact while doing so. The testing is conducted as a self-paced remote work, i.e. similarly as normal grading process is conducted. The session will be recorded to reveal possible problems or usage patterns which would not be informed in the interview. The fully structured interview for every single subject from Wohlin et al. (2012, pp. 62-63) was used as a baseline for interview structure. The interview contains nine questions, which are asked in defined order from each teaching assistant directly after they have finished the grading process. Interview is conducted in Finnish, but questions translated into English are shown in Appendix 1.

3.4.4 User testing methodology

Nielsen and Landauer (1993) presented an equation for number of usability problems found at least once by i evaluators. The equation is following

$$\text{Found}(i) = N(1 - (1 - \lambda)^i) \quad (1)$$

where i is number of evaluators, N is the total number of usability problems, and λ is the proportion of found usability problems while using a single evaluator. However, to correlate with equation and for user testing to succeed, suitable testers must be recruited and they should represent user group as well as possible.

Based on equation 1, Nielsen (2000) suggested that with λ value of 31 % the suitable number of user tester would be 3-5. However, he also stated "the formula only holds for comparable users who will be using the site in fairly similar ways". Regardless of the study being about web pages, similar user testing can be conducted with a desktop application too. On the other hand, as there are different student groups with varying programming experience, more than five testers could be used.

Therefore, the recruitment target is four experienced and four inexperienced Python programmers, who have completed LUT CS1 course. Participants are tried to select such that they represent different student profiles. Initial selection is done based on major discipline, i.e. CS major or non-CS major. More detailed demographic information will be asked separately from actual user testing questionnaire to keep data as anonymous as possible.

Users will be asked to do a remote self-paced testing and report results via questionnaire. The user testing includes completing two weekly assignments, using the artefact and answering feedback questionnaire. To ensure as natural experience as possible participants are instructed to complete given weekly assignments and use the artefact when ever they feel appropriate, and as many times as they want. Also actions based on ASPA feedback are totally self-imposed. The user testing questionnaire is conducted in Finnish, including answers. The questions translated into English are presented in Appendix 2. Both open ended and closed questions will be used, and with closed questions, choices are shown below the question. Rest of the questions are open ended and ones marked with asterisk are mandatory.

4 ARTEFACT DESIGN AND DEVELOPMENT

The weekly assignment submission workflow is shown in Figure 6. More detailed steps are listed below.

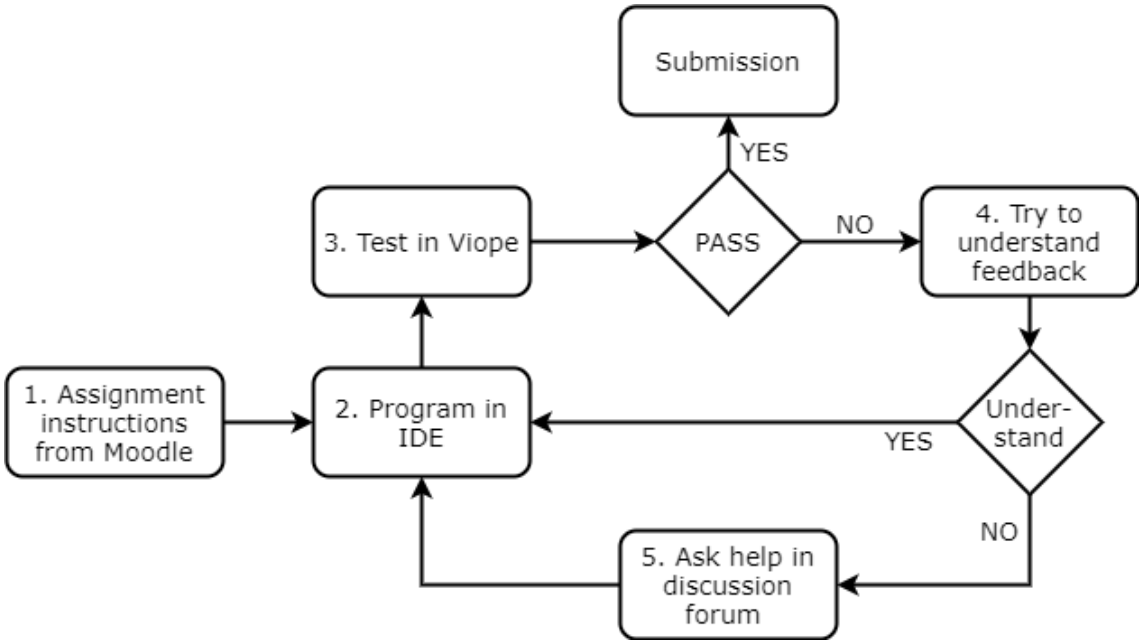

Figure 6. Current student workflow to complete a weekly assignment.

1. Student get the weekly assignment instructions from Moodle.
2. Student implement program to match assignment instructions.
3. Student use automatic code checker to submit the program.
    a) If successful, student can submit and move to the next assignment.
    b) If unsuccessful, student get highlight on part where outputs differ.
4. Student try to understand feedback and figure out how to get output to match.
    a) If successful, student can return to step 2.
    b) If unsuccessful, student does the 5th step: ask help.

This process is far from the optimal because student get little feedback. The initial feedback students get is only syntactical notifications or error messages in IDE. Then while trying to submit the code they get feedback about program output compared to expected output. This lacks guidance how to fix any errors and coding conventions are only evaluated in the course project and the examination when course staff gives written feedback.

4.1 Examination data

In LUT University electronic examinations in dedicated exam room have been utilised in programming courses for 3 years. While the environment is different the basic concept is similar to one studied by Ju et al. (2018). Electronic examination setup enables access to students answers which is key to conduct analysis on examination data in this study.

Dataset of one year electronic examinations is used in this study for initial calibration. Usage of the data is discussed more in Section 5. The dataset contains 413 unique submission by around 380 unique students within two batches. The first batch was the recommended one and the second batch was for those students who failed in the first examination or did not attempted the examination at all in the first time. Examination was graded with numerical scale from 0 to 5. The examination had five different tasks corresponding to each possible passing grade. Students were allowed to submit only one of the five tasks corresponding to the grade they were aiming. General analysis of student submissions in the first batch is shown in Table 3 and analysis results of the second batch are shown in Table 4. Levels from 1 to 5 have individual columns and the rightmost column is statistics from the all submissions.

Table 3. General statistics of students' examination 1 submissions.

Category	**L1**	**L2**	**L3**	**L4**	**L5**	**All**
Number of submissions	64	118	82	30	48	342
Correct filename	53	106	72	29	85	345
Incorrect filename	11	12	10	1	11	45
Total LOC	1 813	5 837	6 584	3 300	6 773	24 443
Average LOC	28	50	80	110	71	63
Maximum LOC	71	134	171	191	164	191
Minimum LOC	14	24	48	66	18	14
Pass	44	100	77	24	33	278
Pass with lower grade than submission level	-	9	2	5	14	30
Fail	20	9	3	1	1	34

Usage of library file is required only in the level 5 examination. Therefore, level 5 submissions are the only submissions containing more than one file per student. Examination instructions included naming schema for the files. To get data on how many students did not follow the instructions, filenames were analysed.

Table 4. General statistics of students' examination 2 submissions.

Category	L1	L2	L3	L4	L5	All
Number of submissions	34	26	5	0	6	71
Correct filename	32	23	4	-	10	69
Incorrect filename	2	3	1	-	2	8
Total LOC	859	1 174	341	-	882	3 256
Average LOC	25	45	68	-	74	42
Maximum LOC	54	96	76	-	146	146
Minimum LOC	12	18	62	-	22	12
Pass	33	24	5	-	5	66
Pass with lower grade than submission level	-	2	0	-	1	3
Fail	1	0	0	-	0	1

In LOC analysis, level 5 library files and main file are not combined, but they are analysed as separate files. Therefore the actual average line count for level 5 submission is around 141 lines. Naturally, also minimum and maximum LOC would be larger if the main file and the library files would be combined. All lines including comments and empty lines are included to the LOC calculations. Grades of the submissions were not included to the dataset. Number of passed, passed with lower grade and failed were taken from Moodle platform after the previous analysis. Examination were graded by the course lecturer.

The second examination batch is analysed similarly to the first one. When comparing the first examination to the second examination the greatest difference is the smaller number of submissions in the second and is has overall slightly smaller numbers in the LOC results, excluding level 5 average and the minimum in levels 3 and 5. These can be due the more students in the second batch who are aiming purely to pass with aimed level. Therefore, they might add very little or no comments as well as do very little extra, such as try-except or if statement checks when they are not required by the instructions.

4.2 Course assignment data

In this study, also data of course assignments collected by Viope platform is used. The course assignment structure is following. There are total of 60 weekly assignments and one course project which is available with two levels. The minimum level which is used to support student, who has trouble to pass the course, usually a subset of non-CS major attendees. The target level is designed in such a way that it is directly continuing from

the minimum level, but it requires more advanced skills such as usage of matrices or two dimensional lists and more complicated data-analysis algorithms. In total dataset contains 24 872 files submitted by 535 users, which inlcudes teaching assistants, course lecturer and Viope administrators. However, while there are non-student users included, they have not been completing assignments by passing all the tests and then submitting the files like their student counterparts. Instead non-student users use their accounts for administrative purposes.

From the whole set 22 441 are passed and rest of them have failed the submission, i.e. there are 2 431 file submissions which have not finished until deadline. System only stores the latest submission data, i.e. while student's test runs are unlimited until the set deadline, the dataset only contains maximum of one submission per student per one assignment. However, one submission may contain multiple files, in the analysed course only at the most two, a main file and a library file. In total 1 707 of them are library files.

To enable convenient navigation during both general and static analysis these files are stored in nested directory structure. The structure need to support analysis for each exam set and for each individual task separately. Therefore, the following hierarchy is chosen `course/exam/assignment/student/files`. Exam is a weekly set of assignments in the system, including 3-5 assignment per week. After parsing the data to the chosen directory structure the more detailed general analysis of weekly assignment and course project submissions are done.

There are huge variations in weekly assignment data across all 60 assignments. From the first Hello World -programs to almost course project sized assignment on the last weeks. On average each weekly assignment has around 400 submissions. The general weekly assignment statistics are shown in Table 5. In addition to the total statistics, results from lecture weeks before and after the midterm week are shown.

It is important to note that similarly as with examination data, library files and a main file are not combined, but they are analysed as separate files. Statistics of weekly assignment show that number of submissions is significantly decreasing from the beginning to the end. However, it is partly due the fact that exams on weeks from one to eighth have five assignments and rest of the weeks have three or four. On the other hand, the second set has 10 % more failures that the first set. It is clear that not every student submit all the assignments. Firstly, they do not even try all of them and secondly not all the students are able to do all the assignments. These statistics are in line with aforementioned results of e.g. Kinnunen and Malmi (2008) study showing that student struggle with difficulty

Table 5. General statistics of weekly assignments.

Category	Weeks 1-7	Weeks 8-14	Total
Number of files	16 581	6 892	23 473
Number of libraries	-	1 007	1 007
Pass	15 580	5 791	21 371
Fail	1 001	1 101	2 102
Total LOC	361 044	330 480	691 524
Average LOC	23	57	32
Maximum LOC	334	372	372
Minimum LOC	1	6	1

of the programming tasks on CS1 course. Due to the weekly assignment grading system used on the course, some students stop completing assignments when reaching desired amount of points and grade.

The course project data is more interesting because it is used as the first level summative evaluation before an electronic examination and these files are evaluated by teaching assistants. Teaching assistant grading allows comparison of human grading and automated grading discussed in Section 5. The general statistic of course project submissions for both target (T) and minimum (M) levels at three different deadlines, 1-3, are shown in Table 6.

Table 6. General statistics of course project.

Category	T1	T2	T all	M1	M2	M3	B all	Both All
# of files	228	74	302	752	300	46	1 098	1 400
# of libraries	114	37	151	376	150	23	549	700
Pass	134	56	190	574	264	42	880	1 070
Fail	94	18	112	178	36	4	218	330
Total LOC	29 999	12 793	42 792	59 209	26 715	4 016	89 940	132 732
Avg. LOC	224	229	225	103	101	96	102	124
Max. LOC	952	558	952	658	265	222	658	952
Min. LOC	41	22	22	12	12	25	12	12

Number of files, number of libraries, passed and failed are calculated based on every submission. Number of submissions includes all files from every student last attempt, in case of no attempt there is no submission. Each course project submission has a main file and a library file with it, therefore, in this specific case number of submissions is number of files divided by two. For line number calculation only passed file submissions are

included, i.e. submission has passed all automated tests. This is due to the fact that failed submissions have both, very short submissions, i.e. problem has only partially solved, and extremely long submissions, i.e. solution is having extremely extensive code elements or files include something else than source code. For example there were submissions where students misunderstood purpose of the library file, by including an analysed data file as a library submission. The data file had over 8 700 lines which would distort all statistics.

Again as with weekly assignment and examination data, library files and a main file are not combined, but they are analysed as separate files. This lead to minimum number to be as low as 12 for a passing file. However, this particular file is a main file which is simply importing a library file and calling the main function from there. Solution of this kind can pass the tests in automatic code checker, because the system only run files in a sandbox and compare an expected output to a generated output. However, a solution like this will be graded as a fail during the teaching assistant evaluation.

Finally, when student got failed grade they were required to submit a fixed version to the next submission round in order to pass the course. Therefore, some students have submitted files to multiple deadline categories too. On the other hand it was not required to submit to the first deadline in order to be allowed to submit to the second one, i.e. submitting only to the second deadline was possible also.

In addition to submissions, Viope provides usage data. Weekly usage times are shown in Figure 7. The system was used a lot during the course, the global maximum is before the course project submission, i.e. week 10. Correspondingly, there are a local peak at week 13th before the second submission.

When comparing usage times to the number of files and total LOC from tables 5 and 6, one can see that during the first seven weeks students created 16 581 different files with 361 044 LOC and during the last seven weeks 8 292 files with 463 212 LOC. For the first half they used 19 053 hours, but the later half 17 563 hours, i.e. almost 8 % less time to create over 28 % more code lines.

Nevertheless, the Figure 7 shows that while less submissions are made, i.e. less students contribute, the remaining students have learned to either produce or submit code much more efficiently than at the beginning. Another interesting information is usage per each weekday as a percentage of total, shown in Figure 8. Deadline was on Tuesday 6 o'clock in the morning and same time new assignment were published. From the graph we can see that during Monday, i.e. day before the deadline, most feedback is needed.

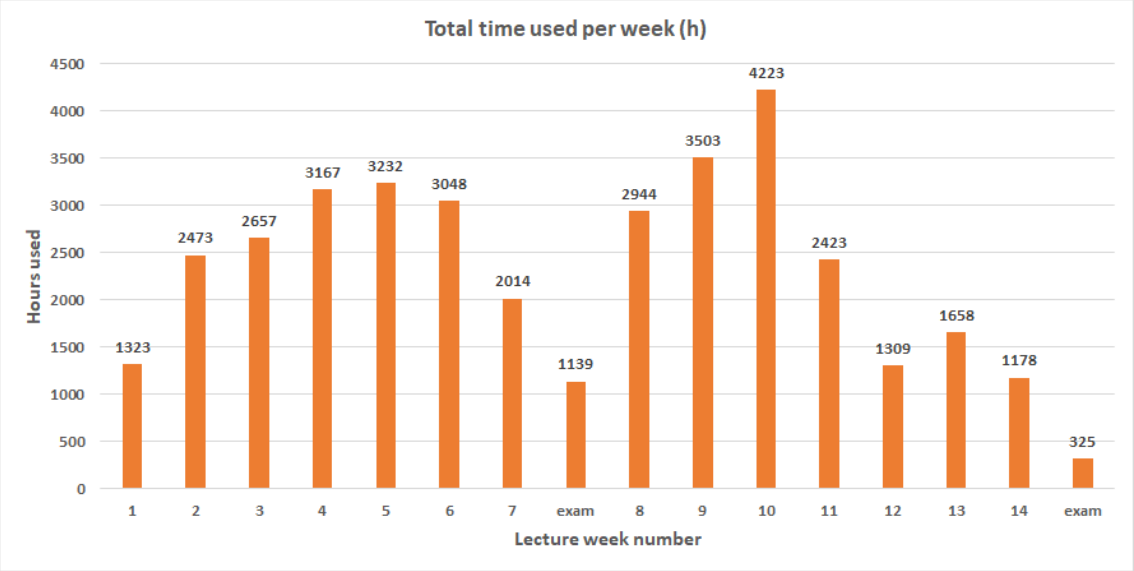

Figure 7. Total usage time of Viope per lecture week.

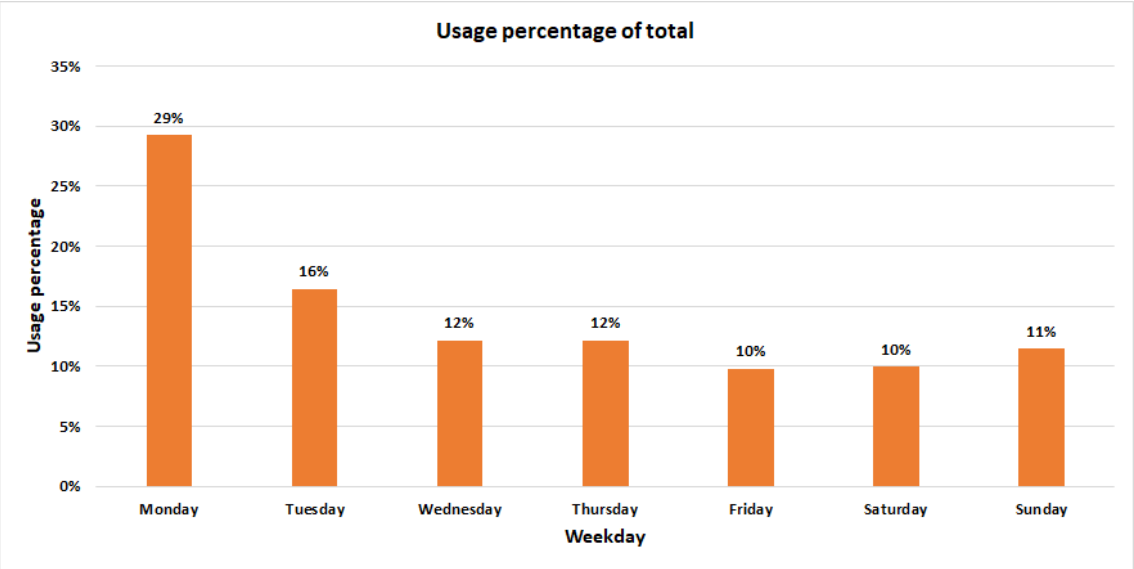

Figure 8. Submission system usage as a percentage of total per weekday.

4.3 Scenarios

The solution will have at least three user groups which are students, teaching assistants and lecturers. The student group is also intended as the primary user group for the artefacts, while course staff is the secondary.

Desired benefits for the user groups using the tool are presented with five scenarios in following subsections. Scenarios are not limiting the solution to work only in these cases, but they give reasonable overview of practical usage possibilities of the artefacts.

4.3.1 Student checking course project

Example student called Brita is doing the course project. She has created some of the core functionalities and wants to test if there are coding convention violations in the code. Brita runs the static analyser program. She selects one of default options, project minimum level, because she wants the analyser to complete analysis related to the minimum level of the course project. In addition, in the case of the course project, she selects two files to be analysed which are the main file and the local library file. After setting analysis options and selecting files, program runs the analysis. Result of the analysis shows Brita that there are no coding convention violations found. Therefore, she can continue her work and implement next feature for the project.

After that she can test if her current implementation passes the target level tests. This process would be identical to previous one, except she selects option project target level. This time analyser tells that Brita has forgot to close her second file handle and there is also a single write operation which is not inside an exception handler. She can now fix her code based on feedback.

4.3.2 Student checking weekly assignment

Another example student called Brian is doing one of the weekly assignments at the first half of the course, i.e. before all the coding conventions have been taught. Brian has written his very first function and he wants to test if there are any coding convention violations. He selects functions option and the file he is working with.

Tests are run and program give two messages as an output. Brian checks the feedback and notices that he forgot one of his variables outside the main function. Therefore, he is using it as a global variable which is violating coding conventions of the course. He moves the variable inside the main function. Secondly, he sees from the feedback that return statement of the function is missing the return value. Because his function does not return anything, he checks from the course material what is the value used in case like that. He learns that Python 3 has a keyword `None` which should be used in these kinds

of situations, therefore he adds `None` as the return value. After running tests again, he is happy to see that no coding convention violations were detected. Now he can move to the next weekly assignment.

4.3.3 Student at the exercise session

Example student Brita is visiting at the exercise session to get help for her weekly assignments. She is having issues with her data analysis' result file. All the written data objects have same values. Brita is asking help from the teaching assistant, who then checks Brita's code. Teaching assistant notices that Brita is accidentally using her data class as a global variable by changing values of class variables directly and not via objects. Static analyser didn't give any notification due the fact that Brita only selected file handling tests. She wants to run static analyser again before fixing the error.

When running static analysis again with more tests, Brita gets two output messages. Firstly, she is using class as a global variable and, secondly, she has one nested import statement which is outside a global scope. Brita fixes the first error, but she doesn't understand the second one and therefore, asks teaching assistant to help her again. Teaching assistant explains what a global scope means and guides her to study recommended file structure from the course's coding convention guide.

4.3.4 Teaching assistant evaluating project

Teaching assistant Simon is evaluating dozens of students' projects. He has two main usages for the program analyser. Firstly, he wants to know general statistics of the projects. He selects a general analysis option, output mode and the root folder before running analysis. As an output he gets a list of general statistics data such as number of submissions, total lines of code (LOC), average LOC and submission distribution between levels. Now Simon has data with which estimate his workload and plan the evaluation schedule.

When Simon starts the evaluation process, he wants to run all the static analysis tests, so he selects the course project analysis option and the folder of the first student's project. After running the tests, he gets a warning feedback about recursive function call and no other messages. Now he can start checking the program manually. However, with this setup he already knows that at least the recursive function call usage needs to be checked, but there are no other common mistakes. Therefore, manual check can be done at overview level

which was not the case before since no general view of the program was available. When checking the recursion, he notices it is used reasonably, so it was a false alarm. During the fast overview check he still finds another problem; a very inconvenient naming schema is used to name variables. After a quick evaluation process, Simon can give feedback stating that every test was passed successfully but naming schema should be more reasonable.

4.3.5 Lecturer evaluating electronic examination

Lecturer is evaluating all of the students' electronic examinations. He has very similar main usages for the program analyser as teaching assistant. Firstly, lecturer wants to know general statistics of the electronic examinations. He selects the exam general analysis option, output mode and the root folder before running the analysis.

Each exam submission is analysed as an individual program, but this can be done in batches for each of the five grades. It is possible to select exam grade 4 and output as a file. Lecturer gets a comma-separated values (CSV) file containing results of the submissions targeting to grade 4 and he can then verify the results by going through the submissions manually on the overview level. For each of these submissions he can give grade 4 with minimal manual checking, when there are no coding convention violations detected. Or lower grade if there are coding convention violations. In addition lecturer can copy the generated feedback to Moodle and move to next submission.

4.4 Requirements

The implementation is naturally affected by the goal of the thesis presented in Section 1.2. In short the goal is to enable better to improve feedback from course staff to the students, such that it supports learning. In addition, there are requirements, which affect the implementation details of the artefact. In this subsection both functional and non-functional requirements are presented in categories. The functional requirements are mostly desired functionalities to the artefact, while the non-functional requirements are mainly constraints derived from the target user groups. Categorised presentation of the requirements is not comparable to the software requirements specification or other formal requirement documentation, but they are used as guideline what needed to be done.

Most of the requirements are included or highly related to the goal itself, but majority are functional requirements from the course lecturer. This initial set of analysis requirements

were a baseline what the artefact needs to do. This set was supplemented with well-known common practices or practices derived from the literature. In addition, during the implementation there were minor changes done as well as addition of new constraints which were not documented until then. Used requirement categories are as follows:

1. General analysis
2. Basic commands static analysis
3. Function usage static analysis
4. Library usage static analysis
5. Data structures static analysis
6. File handling static analysis
7. Exception handling static analysis
8. Other functionalities
9. Configurations
10. GUI.

This is a presentational order, i.e. requirements were not elicited in this order and they were not completed based on this order. Each category has a table with all identified requirements related to the category. Every requirement has an id, a description and a priority presented as separate columns in requirement tables. The id is unique identifier derived from the category name and incremented number. In cases with highly related requirements they are using same major number but an adjustment have minor number added. The description is shortly describing the practical content of the requirement. The used priority scale is high-medium-low and it defines importance of the requirement as well as an order in which requirements are fulfilled. In addition, there are category specific columns, which are only used with subset of categories, e.g. exam column is used with general analysis and static analysis categories. An exam column indicates for which examination grade's task the analysis requirement is related, while a course project column shows the related level of the course project related – minimum, target or both.

The first category is general analysis. Related requirements are shown in Table 7. During the implementation the general analysis was decided to implement as an individual standalone tool. Therefore, further discussion of general analysis requirements and implementation is out of the scope of this study. However, results of the general analysis were presented in the sections 4.1 and 4.2.

Requirements for static analysis categories basic commands, function usage, library usage, data structures, file handling and exception handling are shown in Tables 8, 9, 10, 11,

Table 7. Requirements for general analysis.

ID	Description	Priority	Exam
GA REQ-1	Total number of files	High	1-5
GA REQ-2	Total line count of all analysed files	High	1-5
GA REQ-3	Maximum line count of a file from all analysed files	High	1-5
GA REQ-4	Minimum line count of a file from all analysed files	High	1-5
GA REQ-5	Average line count of all analysed files	High	1-5
GA REQ-6	Library count	Medium	5

12 and 13, respectively. Basic commands static analysis are all purely for checking that different commands are in use. These checks do not give an error, if something is not in use, because for example not all the assignments require use of `for`-loop. However, they should give a note which commands are used.

Table 8. Requirements for static analysis of basic commands.

ID	Description	Priority	Exam
BC REQ-1	`print` command usage check	Low	1-5
BC REQ-2	`input` command usage check	Low	1-5
BC REQ-3	`if` statement usage check	Medium	1-5
BC REQ-3.1	`if-else` structure usage check	Low	1-5
BC REQ-3.2	`if-elif-else` structure usage check	Low	1-5
BC REQ-4.1	`for`-loop usage check	Medium	1-5
BC REQ-4.2	`while`-loop usage check	Medium	1-5

Static analysis of functions is focusing on global and local namespace as well as return value check. As a CS1 course, the course aimed for novice programmers. Therefore, basics such as using functions and using more than one function are checked too. More advanced concepts e.g. nested functions, recursion, `*args`, `**kwargs` and `yield` are detected to give proper warning.

The global variables have been issue in the course. Therefore, there is high priority to detect those. However, global constants are fine, but in this case false positives are infinitely better than false negatives. Therefore, there is only low priority to detect global constants. `return` command with a return value should be used at the end of each function, and if no other value is needed `None` should be used.

Static analysis of library usage is focusing on the file structure, i.e. how larger programs are split into a main file and a library file. The code in global scope refers to other but classes, imports and functions which are required to be at global scope. In addition

Table 9. Requirements for static analysis of function usage.

ID	Description	Priority	Exam
FU REQ-1	Main function usage check	High	2-5
FU REQ-2	Check that other functions are used too	High	2-5
FU REQ-2.1	Nested function detection	Medium	2-5
FU REQ-3	Global variable detection	High	2-5
FU REQ-3.1	Global constant detection	Low	2-5
FU REQ-4	Recursive function call detection	Medium	2-5
FU REQ-5	Function parameters usage check	Medium	2-5
FU REQ-5.1	`*args` and `**kwargs` detection	Low	2-5
FU REQ-6	`return` command usage check	High	2-5
FU REQ-6.1	`yield` and `yield from` detection	Low	2-5
FU REQ-6.2	`return` at the middle of the function detection	Medium	2-5
FU REQ-6.3	`return` without a return value detection	Medium	2-5
FU REQ-6.4	`return` with a constant value detection	Medium	2-5

Table 10. Requirements for static analysis of library usage.

ID	Description	Priority	Exam
LIB REQ-1	Identify library files	High	5
LIB REQ-1.1	Detection of code in global scope in the library	High	5
LIB REQ-1.2	One or more functions in the library files	High	5
LIB REQ-2	Identify a main file	High	5
LIB REQ-2.1	Detection of main function in the main file	Medium	(2-4), 5
LIB REQ-3	Detection of header comments in all files	Low	1-5

detection of header comments is included here as the they are mandatory in all larger programs. Numbers in parenthesis in the exam column mean that there is no library file in these levels. However, the main file could be detected even with one file.

Static analysis of data structures is focusing on checking that data structures are used as well as detecting that they are not used incorrectly. Checked data structures are list, tuple, dictionary, class and object. Detection of incorrect usage are mainly for classes and object because they have been issue on the course. The DS REQ-6 is checked only if it is possible with selected technology. However, the priority is low regardless of the selected technology.

Numbers in parenthesis in exam column mean that it is not required to use specified data structure in these levels. However it is required to use them in levels without parenthesis, e.g. in Table 11 requirements, class is not required in exams 1-3 but in case they are used it is mentioned to the user. This means that class and objects are allowed before level 4 examination, but when used their usage is checked as it would be level 4 examination.

Table 11. Requirements for static analysis of data structures.

ID	Description	Priority	Exam
DS REQ-1	List usage check	Medium	(1), 2-5
DS REQ-2	Class usage check	High	(1-3), 4-5
DS REQ-2.1	Class used as global variable detection	High	1-5
DS REQ-2.2	Non-global class detection	High	1-5
DS REQ-3	Object usage check	High	(1-3), 4-5
DS REQ-3.1	Temporal object usage detection	Medium	1-5
DS REQ-3.2	Number of attributes in object detection	Medium	1-5
DS REQ-4	Dictionary usage check	Low	(1-5)
DS REQ-5	Tuple usage check	Low	(1-5)
DS REQ-6	Check that with 5 or more similar variables, they are put in the data structure	Low	1-5

Static analysis of file handling is focusing on checking usage of required file operations as well as detecting incorrect usage of them. The most important check is to detect left open files because they have previously been issue in the course. Reading and writing checks are high priority requirements, but to detect how the file is read or written is less important. All the file operations are relevant to same examination levels, because usual workflow is to read data from single file and do some analysis and then write results into a result file.

Table 12. Requirements for static analysis of file handling.

ID	Description	Priority	Exam
FH REQ-1	Left-open files detection, i.e. open-close pair	High	3-5
FH REQ-1.1	`with open(...) as` detection	Medium	3-5
FH REQ-2	File reading check	High	3-5
FH REQ-2.1	`file.readline()` command usage check	Medium	3-5
FH REQ-2.2	`file.readlines()` detection	Medium	3-5
FH REQ-2.3	`file.read()` detection	Medium	3-5
FH REQ-2.4	File reading with iterator detection	Medium	3-5
FH REQ-3	File writing check	High	3-5
FH REQ-3.1	Writing mode (a, w, x) check	Low	3-5
FH REQ-3.2	`file.writelines()` detection	Low	3-5
FH REQ-3.3	`sys.stdout` detection	Low	3-5

Static analysis of exception handling is focusing on checking correct usage of Python's `try` statement as well as detecting that required operations are inside the exception handling. Correct structure of `try` statement can vary a lot and therefore initial checks are focusing on exception handling with file operations with high priority.

Table 13. Requirements for static analysis of exception handling.

ID	Description	Priority	Project	Exam
EH REQ-1	Exception handling detection	High	Both	(1-4), 5
EH REQ-2	"Unnecessary lines inside `try`-branch" detection	Low	Both	(2-4), 5
EH REQ-2.1	"All lines in the function inside `try`-branch" detection	Low	Both	(1-4), 5
EH REQ-2.2	"Exception handling in a caller function" detection	Medium	Both	(2-4), 5
EH REQ-3	Exception handling in all file openings	High	Both	(3-4), 5
EH REQ-4	Exception handling in all file readings	High	Both	(3-4), 5
EH REQ-5	Exception handling in all file writings	High	Both	(3-4), 5
EH REQ-6	Exception handling in user input check	Low	Target	5
EH REQ-7	Exception handling in data row manipulation check	Low	Target	5

In previous years there have been problems related to the placement of the exception handling. There have been solutions such as whole program inside one `try` statement, every library function call inside `try` statement, and every function starts with a `try` and ends with an empty `except` branch. Therefore, there are detections for such solutions, but with low and medium priority. Finally, user input and data row manipulations could be detected in target level of the course project. This is also the only difference between minimum and target levels in static analysis point of view.

In addition to all static analysis requirements, there are requirements related to other functionalities, configurations and GUI which are shown in Tables 14, 15 and 16, respectively. The first three major functional requirements are specifying the main usage of the ASPA, i.e. selecting analysed files, analysis options and after the analysis results for each file. Requirements from OF REQ-4 to OF REQ-7 are specifying on which environment and with what knowledge ASPA should be usable.

The OF REQ-8 is ensuring that ASPA is usable with assignments currently used in the course. In other words, there should not be assignment done for ASPA but configuration options in ASPAto suit for assignments. Requirements from OF REQ-8.1 to OF REQ-8.4 are specifying that student must have freedom to create a program with functions, classes and variables as they wish. However, in multiple assignments e.g. in the course project, it is compulsory to have at least one class and more than one functions.

Requirements from OF REQ-8.5 to OF REQ-8.8 are specifying that there must not be any requirements to name e.g. all file handles as `file_handle` or classes with only three letter long name. However, this does not mean that good coding conventions can be broken, e.g. Python keywords should not be used as variable names and some logical naming convention should be used. Finally, OF REQ-9 ensures that at least basic statistics, e.g. number of each detected coding convention violation, are available for research purposes.

Table 14. Requirements for other functionalities and constraints.

ID	Description	Priority
OF REQ-1	Must have possibility to select a file to be analysed	High
OF REQ-1.1	Must have possibility to select whole folder to be analysed	Medium
OF REQ-2	Must have possibility to select suitable analysis options	High
OF REQ-3	Analysis results must be shown for each file separately	High
OF REQ-4	Usage must not require knowledge of programming[1]	High
OF REQ-5	Tool must be usable on major operating systems[2]	High
OF REQ-6	Tool must be usable at least with Finnish language	High
OF REQ-7	Must be usable offline	Low
OF REQ-8	Completion requirements of the assignments must not be changed	High
OF REQ-8.1	Student must not be restricted to use fixed number of functions	High
OF REQ-8.2	Student must not be restricted to use fixed number of classes	High
OF REQ-8.3	Student must not be restricted to use fixed number of data structures	High
OF REQ-8.4	Student must not be restricted to use fixed number of variables	High
OF REQ-8.5	Student must be able to name functions freely	High
OF REQ-8.6	Student must be able to name classes freely	High
OF REQ-8.7	Student must be able to name variables freely	High
OF REQ-8.8	Student must be able to name a main file freely	High
OF REQ-9	Must store basic statistics of analysis results	High

[1] Excluding what is taught during the course. To analyse own programs students need to have basic understanding what they are programming in order to understand the feedback.
[2] Targeted major operating systems are Windows 10, macOS and Linux. However, due the existence of hundreds of Linux distributions, the supported ones will be specified later.

Configuration requirements are specifying main configuration options for ASPA. While many of these requirements are just on-off options for different configurations, there are requirements to have default configurations, dedicated JavaScript Object Notation (JSON) file and option to change name of the result file. Decision which user groups have access to which configurations will be done earliest when representatives from all the user groups have tested ASPA. Therefore, on-off option requirements are not specifying how these options are used.

The final requirement category is GUI. These requirements are specifying what GUI should contain. The GUI REQ-1, i.e. GUI in Finnish, is strongly linked to the requirement OF REQ-6, i.e. tool in Finnish. They are marked as separate requirements to highlight that also the GUI must be available at least in Finnish. Other highest priority requirements are to have GUI element to each major ASPA activities, i.e. selecting analysis, selecting files, showing selections, and showing static analysis results. With lower priority the GUI could have help section, preset analysis options and directory selector. On the other hand, there are no requirements to visuals e.g. colour schema or button layout, because of proof-of-concept nature of the initial GUI.

Table 15. Requirements for configurations.

ID	Description	Priority
CFG REQ-1	On-off option for console print	High
CFG REQ-2	On-off option for result file writing	Medium
CFG REQ-3	On-off option for each static analysis category	High
CFG REQ-4	Default name for results file	Low
CFG REQ-5	Default configurations	High
CFG REQ-6	All configurations in JSON file	Medium
CFG REQ-7	Configuration options for statistics	High
CFG REQ-7.1	On-off option for statistics	Medium

Table 16. Requirements for GUI.

ID	Description	Priority
GUI REQ-1	GUI must be available at least in Finnish language	High
GUI REQ-2	GUI must have selection for each static analysis category	High
GUI REQ-3	GUI must have a file selector	High
GUI REQ-3.1	GUI must have a directory selector	Medium
GUI REQ-3.2	GUI must show selected files and directories	High
GUI REQ-4	GUI must have section for help	Low
GUI REQ-5	GUI must have presets for analysis selections	Low
GUI REQ-6	GUI must have section for static analysis results	High

4.5 Platform comparison

Aforementioned educational programming platforms were studied from seven Finnish universities. Every studied university is using one or more dedicated platform for code submissions. Most of the platforms have been developed inside the university, which is using it, but e.g. Viope is developed by an external company. Every studied platform is still being used or it is used recently and all the comparison are done based on available information. Usage of the seven platforms is shown in Table 17, and universities are as follows:

1. LUT, LUT University
2. TAU, Tampere University
3. UH, University of Helsinki
4. AU, Aalto University
5. UO, University of Oulu
6. UTU, University of Turku
7. JYU, University of Jyväskylä.

Table 17. Usage of seven code submission platforms in Finnish universities.

Platform	LUT	TAU	UH	AU	UO	UTU	JYU
Viope	x						
A+ LMS		x		x			
TRAKLA2			x	x			
TMC			x	x			
Lovelace					x		
ViLLE						x	
TIM				x			x

From Table 17 we can see that only Aalto University and University of Helsinki are using more than one platform. On the other hand, TRAKLA2, A+ LMS and TMC are only tools used in more than one university. However, it is important to remember that some of the platforms are used in multiple other educational institutions outside these universities.

All the platforms have also more advanced functionalities than just accept code submissions as text or as a file. There are functionalities such as allowing to write code directly to submission field, varying task types, testing the submitted code and so on. Assignment submission types and programming question correctness check in different platforms are presented in Table 18. All shown data is based on available information, e.g. Moodle integration information was found only from two platforms. Naturally, platforms seldom explicitly state that there is no Moodle integration.

Programming exercises varied a lot, there are at least following exercise types available code submission as a text or as a file, code submission via IDE plugin, submit answers to given question as a text, drag code element in correct order or to correct locations. In addition, programming exercises platforms provided a varying set of different exercise types e.g. multichoice questions, click or drag items, open ended questions, create or modify graph, match pairs, fill in table and simulate an algorithm from given elements. It is clear that not a single platform provide all of these functionalities, but this gives a good overview of methods in use. Each platform do validation checks for submissions. The programming exercises are validated by using either unit tests, comparison of expected and submission output, or exercise specific tests. Again not a single platform used all of these. Therefore, there might be limitations in exercise types due the limited possibilities of submission validation techniques. From these platform only Viope and A+ LMS can transfer internally stored data to Moodle.

Table 18. Exercise submission and programming exercise correctness check types in the platforms.

Platform	Programming exercises	Other exercise examples	Check submission correctness	Moodle integration
Viope	Code to text field	Multiple choice quiz, open ended questions	Comparing output	Yes[3]
Lovelace	Submit output, file submission	Multiple choice quiz	Comparing output	No[1]
TRAKLA2	Based on pseudocode, no programming	Click or drag items e.g. nodes and references	Check algorithm result	No[1]
A+ LMS	Submit output, file submission	Open ended and multichoice questions	Unit tests	Yes[3]
TIM	Drag lines of code, code to text field	Open ended, graph and multichoice questions	Unit tests	No[1]
ViLLE	Drag lines of code, code to text field	Match pairs, fill in tables, multichoice	Compare output	No[1]
TMC	Submit to server via NetBeans plugin	-	Run checks via NetBeans plugin	No[2]

[1] Scores are internally stored.
[2] Scores are stored to server via plugin.
[3] Scores are stored internally, but they can be automatically transferred to Moodle.

In context of this study, the most interesting functionalities are features related to grading and feedback given to students, they are presented in Table 19. We can see that all but TRAKLA2 show created tests, while TRAKLA2 do comparison to model answer, but the test is not shown to the user. There were two major methods used to visualise test results, either expected output and submission output, or expected results and current unit test results are shown. Error highlight was done with various methods e.g. by giving error message, highlighting incorrect output, showing line number where error occurred, showing violated rules. While Lovelace and ViLLE did not highlight errors during the comparison, they showed tests which can be an effective way to identify the error.

Lovelace was the only platform which gave direct improvement advice, however, not all the exercises had them. For example, there was an advice to use period instead of comma in decimal calculation. Finally, all the platforms had built in grading system which utilised tests. There were three methods used, which are pass or fail, points only if all the tests are passed and scores from each successfully passed test. The number of possible answers was usually unlimited, but this feature was varying also between exercises so it is not included to the comparison.

Table 19. Exercise grading and feedback given to the students.

Platform	Show tests	Highlight error	Improvement advice	Grade answer
Viope	Yes[1]	Highlight output errors, few different errors messages	-	Points if pass
Lovelace	Yes[1]	-	Few different advice	Points if pass
TRAKLA2	No	Violated algorithm rules are shown	-	Pass or fail
A+ LMS	Yes[2]	Give error message and show which line produced error	-	Score based on succeed tests
TIM	Yes[2]	Line number of error is shown	-	Pass or fail
ViLLE	Yes[1]	-	-	Points if pass
TMC	Yes[1]	Highlight error from output or code, if highlight is used in the exercise	-	Pass or fail

[1] Expected output and submission output shown.
[2] Show expected results and current result of unit test.

4.6 Design and development choices

There are many design and development choices done during the development of current version of ASPA. In this section the major choices are presented. They are based on requirements, literature and other available information.

Design and development choices are divided into four subsections, which are used intervention type, used technologies, what kind of analysis options there are, and in which format feedback is given to the users.

4.6.1 Intervention type

At the beginning the suitable intervention method type was selected. Selection is based on user groups and presented results in related literature. Edwards et al. (2017) found that non-CS majors do more mistakes than CS majors, which correlate with observations we have had in LUT CS1 course where non-CS majors ask more help than CS majors. Moreover, in recent attendee distributions CS majors are significant minority. Study results of Porter et al. (2011), Sorva et al. (2013), Vihavainen et al. (2014), Zingaro and Porter (2014), Cukierman et al. (2019) and Olsen and Fox (2019) indicate, that adding a new intervention methods to the course could have positive effect on student performance and motivation. Therefore, it is reasonable to try new intervention technique to help students.

However, intervention methods like Lego Mindstorms Robots require more investments than many pure software options (Fagin and Merkle, 2003; Lawhead et al., 2002). In addition, with successful software solutions it is possible to greatly improve programming outcome with automated feedback and dedicated environment (Allen et al., 2018; Annamaa, 2015; Choudhury et al., 2016; Garg and Keen, 2018; Higgins et al., 2005; Ju et al., 2018).

On the other hand, Yulianto and Liem (2014) tested many existing tools analyser tools. They found that none of the tools was able to detect all the errors, but they were focusing on subset of error types they detect. Therefore, it is reasonable to use many tools, modify existing one or create a new one to fit exactly needs of the course. In addition, while there are wide variety of great tools for major programming languages. None of the currently known tool support feedback in Finnish, which is part of the requirement OF REQ-6.

Finally, there are LUT and Moodle platform related requirements. Based on Moodle integration possibilities of Finnish university platforms shown in Table 18, Viope and A+ LMS are options for the course. Another interesting observation is that none of the found CS1 courses used Moodle platform's CodeRunner module, which was recently studied and used with success by Croft and England (2020).

All these arguments combined, a new intervention method is an autograder-like standalone tool. Due to the standalone nature of the selected tool, no further changes to the current platforms are done or planned.

4.6.2 Analysis options and feedback format

Edwards et al. (2017) studied Java programs with open-source tools and found that the most common error categories were mostly cosmetic issues in Java language. However, found errors are much more erroneous in case of Python 3, e.g. indention formatting is crucial in Python 3. In addition, Edwards et al. (2017) argued that the most time-consuming errors seem to be non-functional errors, e.g. excessive coding. Excessive coding includes size issues such as large classes and large functions. Therefore, structural style violations should be detected too. On the other hand, excessive coding might be related to a decomposition problem among novice programmers, which was mentioned by Garg and Keen (2018). Based on literature also advanced structures should be detected as well as many formatting violations which are crucial in Python 3 syntax. In addition, based on requirement CFG REQ-3 there should be option select which static analyses

are executed. This enable user to select which analyses are executed first, e.g. novice programmer can first do everything else correctly but exception handling. Then after adding the exception handling, it can be analysed with other options.

Sheard et al. (2013) argued that completing a programming task might take long time, even up to weeks. Of course, not all the programming tasks take such a long time. But some tasks might, because problems occur and resolving them takes time. Fixing a bug and trying to spot the semantic error might be a great challenge for a novice programmer. Therefore, formative feedback should be provided to support their learning and possibly same time by giving feedback related to the bugs.

On the other hand, summative assessment is needed to give grades for students. As Simon et al. (2012) concluded there are different weights for course elements in summative assessment. Because LUT CS1 has multiple programming elements it is reasonable to provide similar standardised feedback and formative assessment to the students during the course, as the summative assessment would be at the end. The design choice is to have formative feedback as the main usage for students.

Choudhury et al. (2016) argued that students with different programming knowledge benefit from different types of hints. Based on the result varying guidance is needed to result the best possible programming solution. Therefore, solution should include checks for violations which might be trivial to student with more programming experience. On the other hand, it should also give feedback to the experienced user. Furthermore, based on requirements CFG REQ-1 and CFG REQ-2 there are option to get output as a console print or as a file. Moreover analysis results are shown in GUI by default. This allows different users to select the most suitable output format for their needs.

Various configurations options are the key to fit tool for needs of three different user groups. Therefore, based on requirement CFG REQ-6 there is a separate JSON file for all the advanced configurations. The default configurations will be targeted towards the student user needs, because they are not expected to understand configuration options or tweak them according to their needs. Based on requirement OF REQ-3, results for each analysed file are shown separately. Initially all the output channels use similar format, which contains full file path, filename and static analysis categories separately. Violation messages contain line number and feedback message, and they are listed below the respective categories. Results are separated with horizontal line of hyphens between analysed files. Output format is shown in next sections figure, Figure 10. Some of the violations are more severe than others. Therefore, three detection severity levels are defined:

1. Error – The most severe violation type. Must be fixed.
2. Warning – Less severe than error. Warns about advanced structure or possibly incorrect usage.
3. Note – Neutral mention that something is detected, i.e. nothing is actually violated but something is either fulfilled or neutrally mentioned.

4.6.3 Selected technologies and developed solution

There are various technologies to create a standalone tool, but due to the requirement OF REQ-7 a web application is not a suitable option. Therefore, a desktop application was selected. Artefact need to work at least with Python 3, because it is language used in the LUT CS1 course. In addition, popularity of Python is growing in GitHub (GitHub, 2019). While GitHub repository users do not represent CS1 students and statistics are based on automatically detected languages, this report support the observation of growth.

The convenient way to do static analysis and detect syntactic structure of the student code is needed, in order to detect coding convention errors. Pure string manipulation is way too complicated due the varying combinations. Therefore, proper library or other strategy is was searched. Guo (2013) utilised execution trace and Python's debugger module bdb to visualise Python programs. Gerdes et al. (2010) used non-test-based strategies for assessing Haskell programs. On the other hand, AST was successfully utilised for similar educational purposes by Choudhury et al. (2016), Liu and Petersen (2019), Rogers et al. (2014), Wiese et al. (2017) and Chow et al. (2017). However, AST has known limitations e.g. as a static analyser it is not able to detect run time values for variables or conditions.

Based on literature and requirements the static analysis tool is developed with Python 3.7.2, i.e. the same version as the course is taught. The Python's ast library is used to enable convenient analysis of syntax tree and a dedicated graphical user interface (GUI) is created to enable convenient user experience. The proof-of-concept version of the main page GUI is shown in Figure 9. The tool was named as ASPA. It is an acronym of Abstrakti SyntaksiPuu Analysaattori, i.e. Abstract syntax tree analyser in Finnish. The source code is licensed as GNU General Public License v3.0 (GPLv3) and can be found from Luukkainen (2020).

The GUI of ASPA is implemented with Python's de-facto standard GUI package, Tkinter. Tkinter is an interface to utilise tool command language (Tcl) and its standard GUI, Tk. Tlc and Tk enable application with GUI to work across common operating systems e.g.

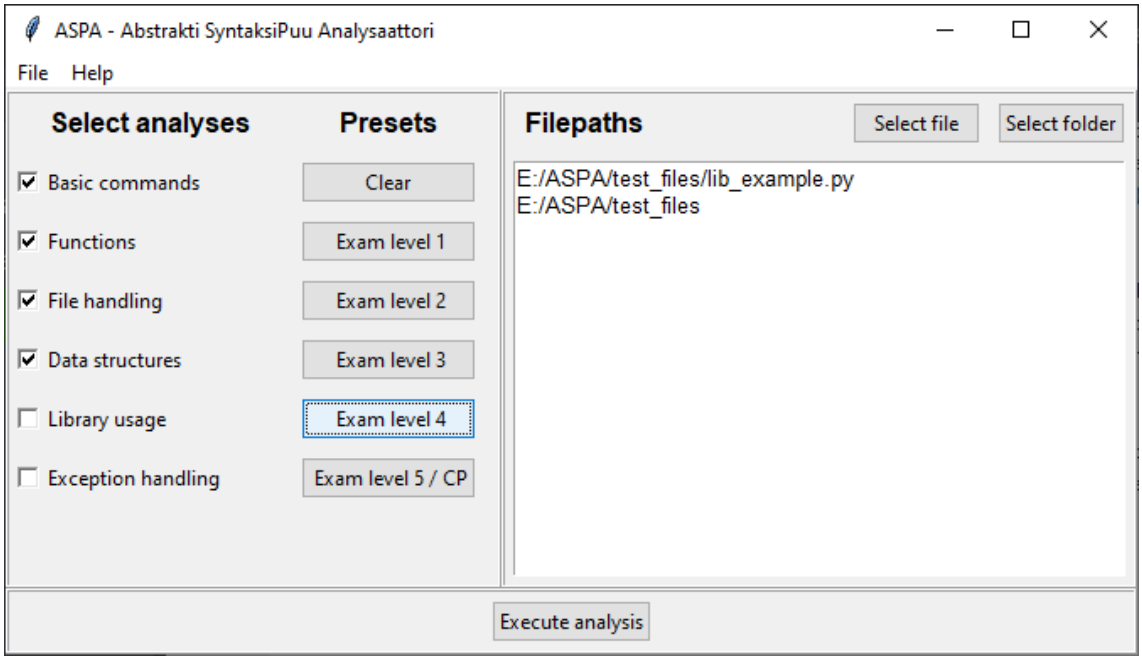

Figure 9. Proof of concept prototype of ASPA GUI main page on Windows 10.

Windows 10, macOS and multiple Linux distributions. The GUI of ASPA is used to ease the usage of the tool for novice programmers. However, during this study GUI is developed only to proof-of-concept level. However, all GUI requirements shown in previous sections Table 16 are fulfilled. The GUI contains two main sections which are frame on the left to select static analysis tests and frame on the right to select files and directories. In addition, GUI contains help section, controller section and results section. Help section is available via help page, accessible via menubar. The controller section on the bottom contains buttons to execute tests. The results page is used to show analysis results and it is automatically shown when analysis are executed. An example of result page is shown in Figure 10, where results of two files are shown with all the analysis options selected.

ASPA has currently six different static analysis modules. Multiple modules are used to increase performance, because while selecting only some of the analysis options, only selected analysis in corresponding modules are executed. On the other hand, not every individual analysis can be executed separately, due the technical reasons with utilised Python ast library's NodeVisitor class. The electronic examination levels are used as a baseline for static analyser division, and modules represent each static analysis category, which are presented in Section 4.4. Coding convention checks of ASPA are done based on the LUT CS1 course's programming guide.

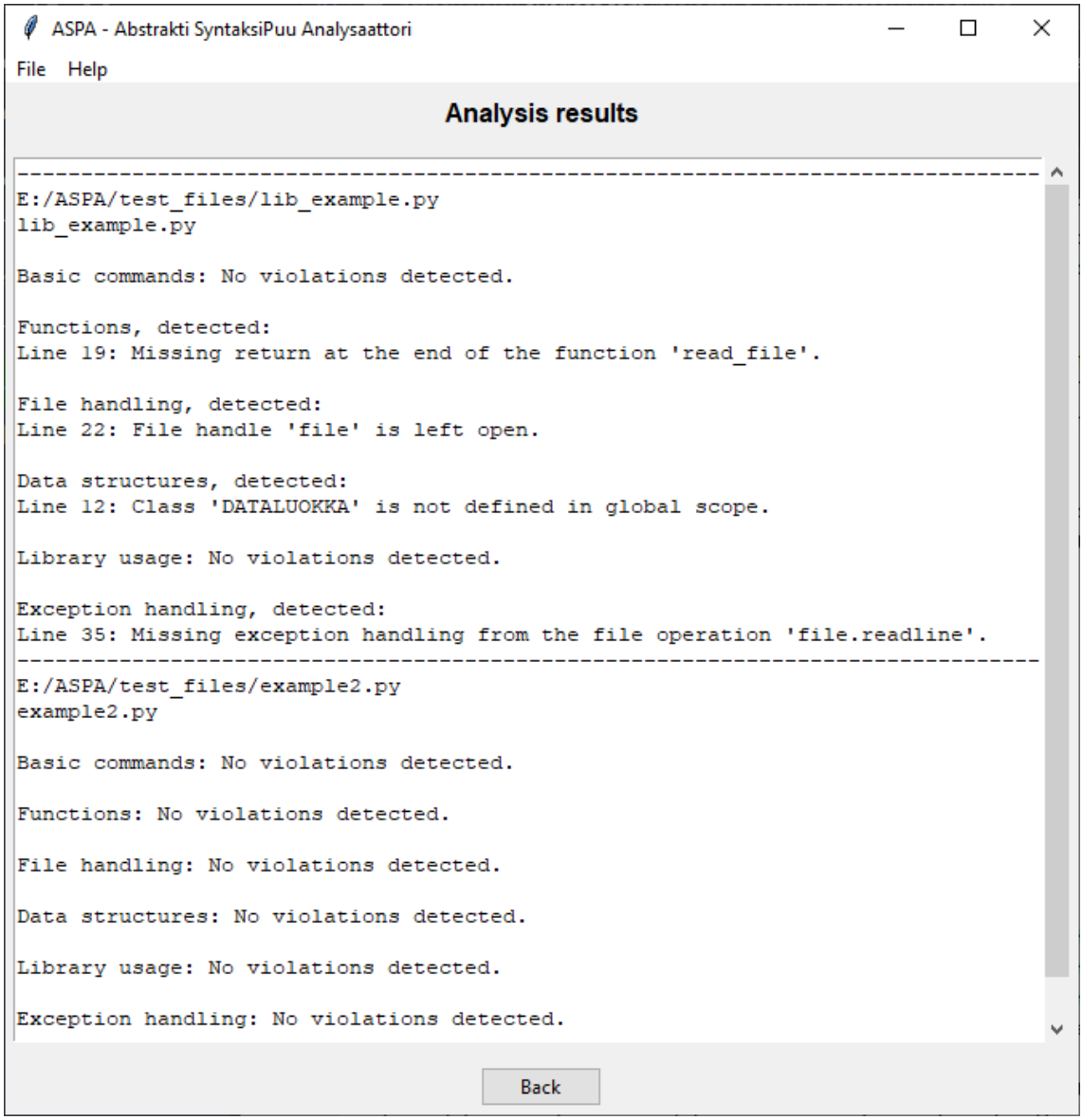

Figure 10. Proof of concept prototype of ASPA GUI result page on Windows 10.

5 ARTEFACT USAGE AND EVALUATION

In this section usage of ASPA is demonstrated with analysis workflow and usage examples of ASPA are presented. The analysis workflow contains intended usage steps while analysing files with ASPA. Usage examples contain examples of detected violations from each static analysis category.

After the demonstration the created artefact is being evaluated by four different methods. The first method is to compare analysis results from ASPA to human grading results. The result comparison is done multiple times to different datasets. A goal of the method is to verify functionality of ASPA before other evaluations. The second and third methods are an educational expert interview and teaching assistant interview.

Experts from two target groups are interviewed to gather expert feedback and improvement suggestions about ASPA. Experts are only asked to participate the interview session, i.e. no prior or post actions are required. All interviews were done after the systematic result comparisons, but before any formal user testing. There were 3 interview sessions, one for educational experts and two separate teaching assistant sessions. All the interviews were conducted in Finnish and comments shown in this study are translated from original comments. The final method is to test ASPA in practice by conducting an user testing. Evaluation results from all major iterations are discussed with details in following subsections.

5.1 Analysis in practice

ASPA was created to do static analysis for Python files, and to enable convenient usability ASPA has a dedicated GUI. Via the GUI, user can do all the major analysis actions and see the results. Workflow to do an analysis with ASPA is shown in Figure 11.

The phase 1 is a starting phase, i.e. default main page. From the main page user can complete phases 2a and 2b in any order. Files can be selected by clicking *Select file* button and use operating system's file explorer to select a file, which is shown in textbox on the right. Entire folders can be selected similarly by using *Select folder* button and analyses can be selected by clicking checkboxes of desired analysis option or by using preset options. After files, folders and analysis options are selected, analysis can be executed. User can check result of the analysis from GUI or optionally by opening a result file. Based on

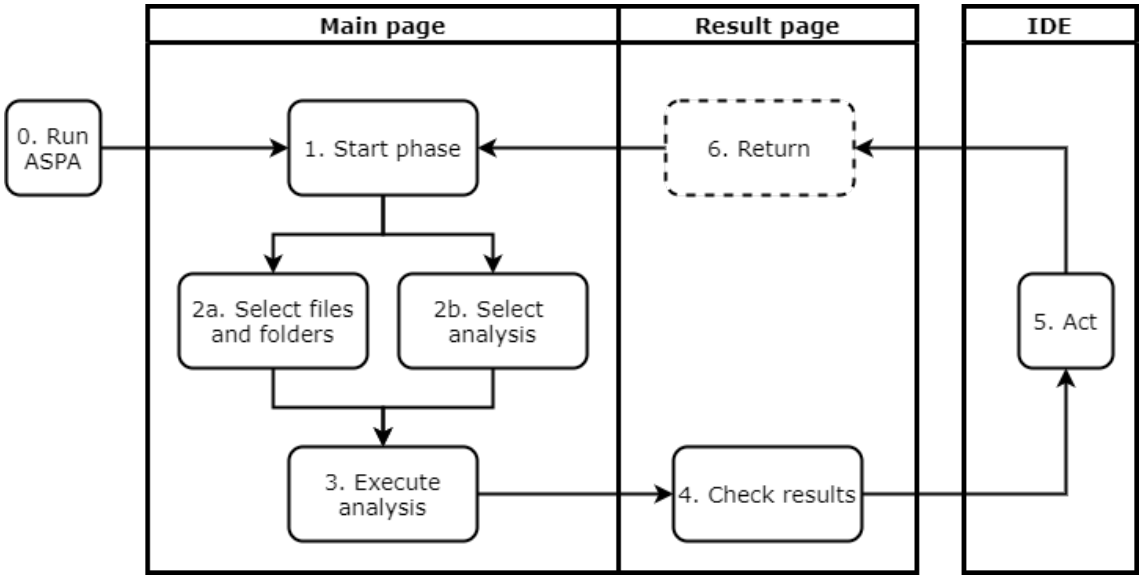

Figure 11. Workflow to do an analysis with ASPA.

the results user can act, e.g. refine code or grade submission, and if another analysis is needed, user can click *Back* button and start from the beginning. ASPA can be closed at any phase.

5.2 Analysis examples

In this section six analysis examples are shown. Examples are grouped by static analysis categories and they include detected coding convention violation errors, warnings and notes. Double plus sign (++) and double asterisk (**) are used to represent notes and warnings, respectively.

Examples cover a subset of all detections and they are selected to represent only couple core features of each analysis category. Each example contains small Python sample and feedback from ASPA. Python samples are fully artificial, but they are based on submitted student solutions which were partially incorrect.

The first example is from the basic commands category. It is shown in Figure 12, and it contains detection of `round` function usage. Feedback is a note, i.e. there is no violation error or warning but a notification that specific function is called. This detection is purely for demonstration purpose to show that single function calls can be detected if needed. This detection is not used in any analysis afterwards. Analysis is done with *Basic commands* and *Functions* analysis options selected.

```
1 def main(var):
2     print(round(var))
3     return None
4 main(1.3)
```

```
Basic commands, detected:
Line 2:  ++Function 'round' is called.
```

Figure 12. Detection of `round` function usage.

The second and the third examples are from the functions category. The second example is shown in Figure 13, and it contains detection of global variable. Analysis is done with *Basic commands* and *Functions* analysis options selected.

The style guide has subsection for namespace, which contains explanation of global and local scopes as well as which elements should be at each scope. Every variable should be at a local scope, excluding global constants. While Python has no actual constant type, values of variables at a global scope should not be altered within program execution.

```
1 data_list = []
2 def main():
3     data_list.append(1)
4     print(data_list)
5     return None
6 main()
```

```
Functions, detected:
Line 1:  Global variable 'data_list'.
```

Figure 13. Detection of a global variable.

The third example is shown in Figure 14. It contains detection of incorrect `return` statement usages as well as a recursive function call. Recursive function call and constant return values are warnings, but other violations are considered as errors. Analysis is done with *Basic commands* and *Functions* analysis options selected. The style guide contains explanation of `return` statement functionality and usage of return values. This include using `return` statement at the end of the function, as well as, using `None` if no other return value is used. ASPA feedback contains two errors for the line 13, because of the missing return value and the `return` statement at the middle of the function. Both are detected and shown separately.

The fourth example is from the data structures category. It is shown in Figure 15. It contains detection of incorrect usages of classes. Analysis is done with *Basic commands*,

```
1  def menu():
2      print("Example Menu...")
3      selection = input()
4      return "selection"
5
6  def fun():
7      print("Example")
8
9  def main():
10     selection = menu()
11     if(selection == "1"):
12         fun()
13         return
14
15     elif(selection == "2"):
16         main()
17     return None
18 main()
```

```
Functions, detected:
Line 4:  **Return value is a constant.
Line 6:  Missing return at the end of the function 'fun'.
Line 13:  Return statement at the middle of the function.
Line 13:  Missing value from the return-statement.
Line 16:  **Recursive function call.
```

Figure 14. Detection of incorrect `return` statement usage and recursive function call.

```
1  class C1:
2      var = 0
3
4  def main():
5      class C2:
6          var = 0
7      obj2 = C2
8      C1.var = 1
9      obj1 = C1()
10     obj1.var = 2
11
12     return None
13 main()
```

```
Data structures, detected:
Line 5:  Class 'C2' is not defined in global scope.
Line 7:  Missing parenthesis from object creation.
Line 8:  Class is being used as global variable 'C1.var'
```

Figure 15. Detection of incorrect usage of classes and objects.

Functions and *Data structures* analysis options selected. Style guide contains explanations of basic class usage, such as defining a class, creating an object and creating a method. This include guide to define class at a global scope. The example in Figure 15 contains three incorrect usage of a classes. The C2 is not defined at the global scope, and an object is created without parenthesis and variable `var` from C1 is altered directly.

The fifth example is from the library usage category. It is shown in Figure 16. It contains incorrect usage of `import` statement, missing header comments and too many function calls at a global scope. In the example only a main file is presented, i.e. correctly implemented library file is excluded. Analysis is done with all but *Exception handling* analysis option selected, however all violations are related to the *Library usage*.

```
1  # Header comments should be here.
2  import lib
3
4  def fun():
5      print("Example")
6      return None
7
8  def main():
9      import sys
10     while True:
11         selection = input()
12         if(selection == "1"):
13             break
14     return None
15 import lib
16 main()
17 fun()
```

```
Library usage, detected:
Line 1:  Missing some or all header comments at 10 first lines.
Line 9:  Import of the library 'sys' is not at the global scope.
Line 15:  Library 'lib' is imported again.
Line 17:  Function call 'fun()' is 2 function call at global scope.
There should be only one (1) function call, calling the main function.
```

Figure 16. Detection of library usage related violations.

Based on style guide and requirement to have header comments, there are four violations in Figure 16 – non global import, multiple function calls at the global scope, importing same library module more than once, and missing header comments at the beginning of the file. The sixth and the last example, shown in Figure 17, is about file and exception handling categories. It contains a file left open and four incorrect exception handling examples. Analysis is done with all analysis option selected, however all violations are related to the *File and exception handling* categories.

```python
# Author:
# Student number:
# Date:
# Cooperation:
def file_reader(filename, data_list):
    fhandle = open(filename, "r")
    try:
        while True:
            row = fhandle.readline()
            data_list.append(row)
    except OSError:
        pass
    except:
        pass
    fhandle.close()
    return data_list

def file_writer(filename, data_list):
    try:
        fhandle = open(filename, "w")
    except:
        pass

    for row in data_list:
        fhandle.write(row)
    return None

def main():
    data_list = []
    data_list = file_reader("filename.txt", data_list)
    file_writer("result.txt", data_list)
    return None
main()
```

```
File handling, notes:
Line 20:  File handle 'fhandle' is left open.

Error handling, detected:
Line 6:  Missing exception handling from the file opening.
Line 19:  **Error handling has only one (1) except.
Line 21:  Missing exception type.
Line 25:  Missing exception handling from the file operation
'fhandle.write'.
```

Figure 17. Detection of a left open file handle and incorrect exception handling.

5.3 Comparing analysis results to human grading

Coding convention violations detected by ASPA and humans were compared multiple times. The first iterations for each new static analysis were done with artificial data. When all artificial cases were detected, the real student data was analysed with systematic comparison method presented in Section 3.4.

Student data analysis were started with sample of examination data because examination programs were smaller than course projects. This iteration focused to verify violation detection and reveal obvious violation cases which were missed in the artificial data.

5.3.1 Electronic examination analysis

The first comparison to real data was done to a subset of the first electronic examination dataset. Subset contained 50 submissions, 10 from each five levels, i.e. 60 files. During the comparison it became clear that level 1 examination submission were such simple that there are not actual coding convention violations to detect. However, it is possible to check that very basic commands and structures, such as `print`, `input`, `if`, `for` and `while` are used.

From the further analysis the level 1 examination submissions were excluded. The results of the comparison are shown in Table 20. There are five columns, one for the detected violation types and four representing detector. Violation type column has subheadings for each level from 2 to 5. Rows for each level represent static analysis which were used to produce ASPA results, e.g. library usage is only relevant for level 5 submission. On the other hand, data structures, e.g. class and list, are only required on the levels 4 and 5, but they are often used in other levels too. Therefore, checks for correct usage are done in all levels.

The column both means that a violation is detected by ASPA and by the human grader, i.e. lecturer. ASPA and human columns show number of violations detected only by ASPA and human grader, respectively. ASPA style column show number of style violations which are detected by ASPA. When student submission violates recommended coding conventions, it contains style violations. The level 4 had one file not following the examination instructions and having both data structure violations. These were detected only by the human grader, and in general there were only three submissions with violations detected by ASPA.

Table 20. Comparison results of the first examination dataset.

Violation category	Both	ASPA	Human	ASPA style
L2				
Functions	0	0	1	8
Data structures	0	0	0	0
L2 total	*0*	*0*	*1*	*8*
L3				
Functions	0	0	0	17
File handling	0	5	0	0
Data structures	0	0	0	0
L3 total	*0*	*5*	*0*	*17*
L4				
Functions	0	0	0	2
File handling	0	4	0	1
Data structures	0	0	2	0
L4 total	*0*	*4*	*2*	*3*
L5				
Functions	3	0	0	13
File handling	0	1	0	0
Data structures	0	0	4	0
Library usage	0	0	1	0
Exception handling	0	8	0	29
L5 total	*3*	*9*	*5*	*42*
Grand total	**3**	**18**	**8**	**70**

Finally, the level 5 had three submissions with minor problems following examination instructions and one submission with major problems. Global variables were detected by both, but all of other instructional violations were detected by human grader only. In addition, ASPA detected a file left open and three submission with file operations missing exception handling. From the detected style violations, 12 were return related violations, one warning of recursive function call and rest of them were related to exception handling, e.g. missing exception type and aforementioned multiple exceptions violation.

Comparison resulted a grand total of 99 detected violations. Only three violations were detected by both and ASPA reported over twice as many non-style violations as human grader. In addition, number of style violations was significantly larger, so there is room for improvement in the codes.

5.3.2 Course project analysis

The first course project submission dataset contained 114 target level submissions and 376 minimum level submissions, in total 980 files. From these 490 submissions, 316 passed Viope tests and were graded by teaching assistants. In total ASPA detected 5 027 coding convention violations, but the same violation can occur multiple times in the same program. For example, missing return at the end of the function occurs as many times as there is a function without `return` statement. Finally, there are both style and non-style violations detected.

As the dataset is significantly larger than the used examination dataset, results are shown in general level and mainly describing violation types not individual violations. Also because of possibility of multiple violations in the same file, only number of different violation types are counted per each file. But the same violations can be detected in each different file separately. As an example submission A has three left open file handles and one global variable detected, and submission B has seven global variables. There will be only two violation types counted in submission A – a left open file and a global variable. In submission B only one type – a global variable. In total there are three types counted, two from A and one from B. Results of the comparison are shown in Table 21. As with the electronic examination data there is one column for the detected violation types and four representing detectors. Detector columns are the same, i.e. both, only ASPA, only human grader and style violation detected by ASPA. Sum of detected types shows sum of all types in individual submissions.

Table 21. Comparison results of the first course project dataset.

Category	Both	ASPA	Human	ASPA style
Sum of detected types	254	60	123	641
Number of submissions	150	49	96	295
1 violation type	78	40	73	79
2 violation types	46	7	19	109
3 violation types	20	2	4	87
4+ violation types	6	0	0	20

The results in Table 21 show that from 316 submissions, 295 had at least one style violation, i.e. over 93 % submissions had something to improve in coding style conventions. On the other hand, both detected 254 violation types. In addition, human graders, i.e. teaching assistants, detected slightly over double the amount of violation types as ASPA,

123 and 60 respectively. From all the detected non-style violation types ASPA detected 72 % and human graders 86 %. On the other hand, if adding all violation types detected only by ASPA to ones detected by human the sum of detected types would increase almost 16 %. The five most common coding convention violations in both minimum level and target level were as follows:

1. Return statement at the middle of the function.
2. Missing exception handling from the file operation.
3. Missing exception type.
4. Missing return at the end of the function.
5. Missing header comments

These are style violations, excluding the missing exception handling from the file operation. This violation type include all read and write operations without exception handling. The vast majority of the non-style violations detected only by ASPA were file left open and missing exception handling from file operations. These violations were usually occurring only in part of the operations and therefore, it is possible that human miss e.g. a single file opening without an exception handling which was then detected by ASPA.

However, there were also four types of false positives and one type false negative verified during the comparison. False negatives are counted from those violations which ASPA should have detected but it did not detect. Finally, in Table 22 are shown 22 violation types, which were detected only by human graders.

Table 22. Identified violation types which are currently not detected by ASPA.

1. Unclear structure	12. Unclear coding convention
2. Library is used incorrectly	13. Hard coding
3. List operations	14. Incorrect data in class
4. Data stored incorrectly to class	15. Class is missing / unused
5. Incorrect usage of objects	16. Naming schema
6. Incorrect usage of main function	17. Missing main function
7. Incorrect parameters	18. Storing return values
8. Missing parenthesis [1]	19. Logical/Functional error
9. Reading file multiple times	20. Try-else branch has used incorrectly
10. Error messages are incorrectly used	21. Unreachable code
11. Try branch has code which does not belong there	22. Except branch has code which does not belong there

[1] Excluding object creating, from where missing parenthesis are detected.

5.4 Educational experts interview

As planned the interview of educational experts was conducted as a recorded online meeting with four experts. The expert participants had backgrounds in multiple different educational institutes as well as in varying industrial positions. They were all current or former university lecturers and one of the experts was representing Viope Education LTD.

The interview session began with general instructions followed by introduction to ASPA. The current and new workflow steps explained how ASPA could simplify user groups workflows. The demonstration step contained four subsections which were limitations and expected usage demonstrations from student, teaching assistant and lecturer point of views. The limitations of ASPA and abstract syntax trees in general were discussed, as well as, cases which are not yet working in ASPA. The student point of view focused on analysing examples from two weekly assignments and course project before and after fixing coding convention violations.

The teaching assistant point of view contained two demonstration examples. The first being a batch analysis of multiple course projects, in which all the submissions from the first course project submission were analysed at once. The second example was more detailed look of feedback, i.e. how a teaching assistant could actually utilise feedback from ASPA when grading course projects. Finally, the lecture point of view was very similar to the teaching assistant point of view. The demonstration had a batch analysis of all examination submissions as well as more detailed look of couple individual submissions. In detailed look the feedback and the submission were compared to see how grading can utilise ASPA feedback.

A set of questions were asked during the session. Each question was asked when the question was relevant to the current discussion and demonstration stage. In addition, at the end of the session all remaining questions were asked to get answer to every question. On the other hand, there were also relevant comments given to topics, which were not directly included to the questions. All ASPA related comments are presented as part of the following summaries. Comments related to Viope's point of view are presented as a coherent entity and the summary of all other comments and answers are presented per topic.

5.4.1 Industry point of view

Expert from industry was really interested in ASPA and said that there is great potential with ASPA. The greatest potential was seen with case when feedback from ASPA would be shown together with current feedback, and there could even be an individual feedback set for each assignment. Then the feedback would be more useful to the students, and dramatically decrease the number of questions asked in discussion forum.

If ASPA would be integrated into current code checker platfrom, Viope, it would add more value to the platform. On the other hand, from working life point of view it was still desired that students would use IDE for programming and code checker should remain as a checker.

5.4.2 Student point of view

The expert panel estimated that ASPA would help students to learn. Effect on quality of code in student submissions and quality of learning, was considered greatly positive. ASPA was seen as a precheck protocol as well as a formative assessment tool. The precheck should be passed before actual submission to code checker, because as Expert 3 stated:

> "This forces to work and program in such a way that it blocks all the craziest misunderstandings and the minimum effort solutions . . . this cut down all such nonsense, therefore, this is definitely a tool to increase quality of learning and the end result." — *Expert 3*

In other words, ASPA guides student to code with good conventions, and warns about bad and minimum effort practices, which includes e.g. everything in the global scope without functions solutions. Similarly as a formative assessment tool, ASPA gives semi-continuous feedback not only grade from the summative assessment. Expert 4 stated:

> "This could be formative assessment by giving feedback about [student's programming] skills . . . so when students can use the tool to wrestle with their code, they have a chance to improve as programmers, which is very useful." — *Expert 4*

The experts agreed that this could ultimately enable student to write similar code every single time, which would ensure that the code is clear and understandable for everyone.

5.4.3 Teaching assistant point of view

The expert panel estimated that ASPA would help teaching to give consistent and more standardised feedback. Expert 1 summarised the usefulness of ASPA as follows:

> "I believe that this [tool] provides a solid foundation for the assessment process of course projects and also speeds up the process." — *Expert 1*

It was also agreed that especially important improvement with ASPA would be same feedback to similar code, i.e. when two students do parts of the course project together they would get similar if not fully identical feedback. Previously there have been inconsistency issues when grading conventions varied between teaching assistants, i.e. similar solutions got different grade and feedback, and students were naturally unhappy and frustrated. In addition, teaching assistant could directly utilise feedback from ASPA as part of the grading feedback, e.g. by copying detected violations from the result file.

5.4.4 Lecturer point of view

From lecturer point of view the most practical usage would be utilise feedback from ASPA as part of the grading feedback. Expert 1 mentioned that following workflow would be the most suitable one. Batch analyse multiple examination submissions at once, e.g. submission from a single level. Then open the first submission, compare feedback and the code, copy feedback from ASPA to Moodle, and add if something seems to be missing. This can be repeated to each submission. The workflow would increase amount of given feedback and ensure consistency. ASPA could have predefined comment for each examination level, this should be a configuration option. In addition from lecturer point of view ASPA provides a possibility to prohibit or discourage coding conventions which are considered unclear for novice programmers. For example, Python enables usage of `else` branch with loops, `try` statement and conditional statement, while many other languages support `else` branch only in conditional statement. Same time it could additional way to guide students, as Expert 1 stated:

> "If we can use ASPA to tell student, don't do this, don't do that, then we indeed guide students towards good and understandable coding conventions. And that is our goal [with student users]." — *Expert 1*

There are also small changes needed to the course implementation in order to enable usage of ASPA. Then consequences of changes and student reactions about aforementioned

examination feedback need to be observed. Finally, the key thing for ASPA would be versatile configuration possibilities, because style guide may change and a new convention could be required to prevent certain student solutions.

5.4.5 Improvement suggestions and open feedback

The expert panel gave many suggestions, firstly, current preset options could be more precise, e.g. drop-down menu for the week and the assignment selection. Secondly, there are currently too many buttons for students. There could be clearer "no violations detected" message when nothing is detected, and the current report when something is detected. These two suggestions were agreed by all the experts, but the third one had multiple viewpoints.

The third suggestion was about selecting static analysis options. The initial expert suggestion was to have configuration options to define static analysis tests for each assignment separately. Other viewpoint was to detect the assignment based on the selected file itself. However, it would not be practical to detect assignment without a proper metadata. Suggested solution was to add a metadata line, which could be provided in assignment description and it would be required in both ASPA and Viope. Then it would be parsed by ASPA to automatically select correct analysis options.

The third expert added that in any case the analysis selection should not be fully manual, but it should somehow prevent student from deselecting every analysis option. Because the best way for student to get rid of all the feedback messages, is to deselect all the static analysis options from ASPA.

There was no silver bullet solution found. However, it was agreed that regardless of the selection method, any automatic configuration which require synchronization of assignments and ASPA is more work for course representatives. Especially, while creating the course, however these options are worth to consider.

The expert panel mentioned, that ASPA is needed, because there are platforms which check only the output of the submission, on the contrary ASPA uses AST to checks structure and usage of the code elements. However, the current version of ASPA has already over 10 elements which can be interacted with, while student version should optimally have only two buttons – select a file and execute an analysis.

On the other hand, they agreed that ASPA can be extremely useful without integration to the code checker, especially for course staff. Therefore, in fundamentals course in student usage it might be required that ASPA would be integrated to some platform or students would get simplified version, while course staff will use the fully configurable version of ASPA. Expert 3 summarised current state of ASPA:

> "Regardless of the limitations due to the nature of non-heuristic analyser . . . I think this [tool] is an improvement to the current practices." — *Expert 3*

In addition, it was discussed utilising similar style guide checks in more advanced CS courses and potential for various other usages were indisputably agreed as well. The considered future implementations were e.g. visual representations, plagiarism detection, more statistics, Bayesian knowledge tracing (BKT) value calculation. Further development ideas are discussed more in Section 6.

5.5 Teaching assistant testing and interview

Teaching assistant testing was conducted by two teaching assistants, TA1 and TA2. Following subsections present summaries of teaching assistant answers for three categories. Similar parts of the answers are shown as a combined summary and when answers had differences they are shown separately.

5.5.1 Effects on grading process

This subsection present summaries of teaching assistant interview answer for questions from 1 to 3. Question were about does ASPA have effect on evaluation time, evaluation quality and evaluation result, respectively. At the beginning and with single submission it was considered that ASPA slow down the process. However, when amount of submissions increases the grading might be faster. TA1 and TA2 emphasised the confidence and reduces assessment iterations:

> "Give confidence, that found all the violations and no need to check [submissions] again." — *TA1*

> "Speed up [the process], because you can check if you found same violations as ASPA. No need to check [submissions] twice." — *TA2*

However, both found violation type, which current ASPA checks did not detect. In addition they agreed that in this small sample ASPA did not effect on the assessment result, however, TA1 stated:

> "With larger sample size the results could be affected and in general become more consistent." — *TA1*

5.5.2 Correctness and usability

This subsection present summaries of teaching assistant interview answer for questions 4, 5 and 8. Question were about anomalies, ease of use and would they use ASPA to support grading process, respectively. TA1 and TA2 mentioned in total of 14 things which were more or less anomalies. Eight of them were included into other answers too, but six clear anomalies were as follows:

1. There are both forward and back slashes in the shown file path
2. Analysis selection checkboxes have quite large hitbox
3. Results page appends the results between analysis
4. The result file is stored in the path where the ASPA is executed, not in the path where the executed file is stored
5. A bug that the first analysis did not show result page
6. The ** marking is not currently explained in the GUI.

In general, both teaching assistants found ASPA clear enough to be used by teaching assistants, as well as, agreed that selection of analysis items and files is intuitive and simple to use. Results were found clear and unambiguous and both would definitely use ASPA to verify their grading results. In addition, TA1 emphasized consistency:

> "Already after 10 [assessments] you become so jaded, that you will not detect everything. It is very hard to remain consistent without a tool." — *TA1*

5.5.3 Improvement suggestions and open feedback

This subsection present summaries of teaching assistant interview answer for questions 6, 7 and 9. Question were about adding or changing functionalities and the open feedback, respectively. TA1 would not add new functionalities, but would add more static analysis tests and improve current features. In total following 12 distinct changes were suggested:

1. In the front page, add button to go directly to the result page
2. Drop down menu for presets
3. Larger font size to the selected file text box
4. Possibility to change font size
5. Colour coding to the GUI's result page
6. Clearer separator between two files
7. Even clearer separator between two students
8. Possibility to select a new file directly from the result page
9. Remove unnecessary exit button and center buttons at the bottom
10. In file menu in the menubar, add option to get back to the front page
11. ASPA should have own icon, not the default Tkinter icon
12. Change the default Windows 10 theme, it is not very good looking.

5.6 User testing

The user testing is conducted to collect feedback from the third user group – students. While workflows of all user groups are highly similar if not identical as discussed in Section 5. However, instead of utilising ASPA during the grading process, an objective of students is expected to be learn and improve based on ASPA feedback.

The user testing followed the plan described in Section 3.4 in all aspects, but the number of participants. Due to the unexpected problems to recruit inexperienced programmers the whole testing was conducted only with five user testers, not eighth as planned. Five students, who have completed LUT CS1 course, were recruited as user testers. Both CS majors and non-CS majors were represented, as well as male and female students. Participants were from second to fifth year students. All the answers are presented as summaries grouped by a question. The first question was about prior programming experience with Python. Answer distribution is shown in Figure 18.

Questions 2 and 3 were about how many times ASPA was used during given weekly assignments. The first assignment was from the fifth week, i.e. when the functions is being taught. The second assignment was from the ninth week, i.e. when the last new topic, exception handling, is being taught. Answers varied from from 1 to 4 uses and from 1 to 6 uses with the first and the second assignment, respectively. Everyone used ASPA at least once with both of the assignments and maximum uses 4 and 6 are quite reasonable if required changes are small.

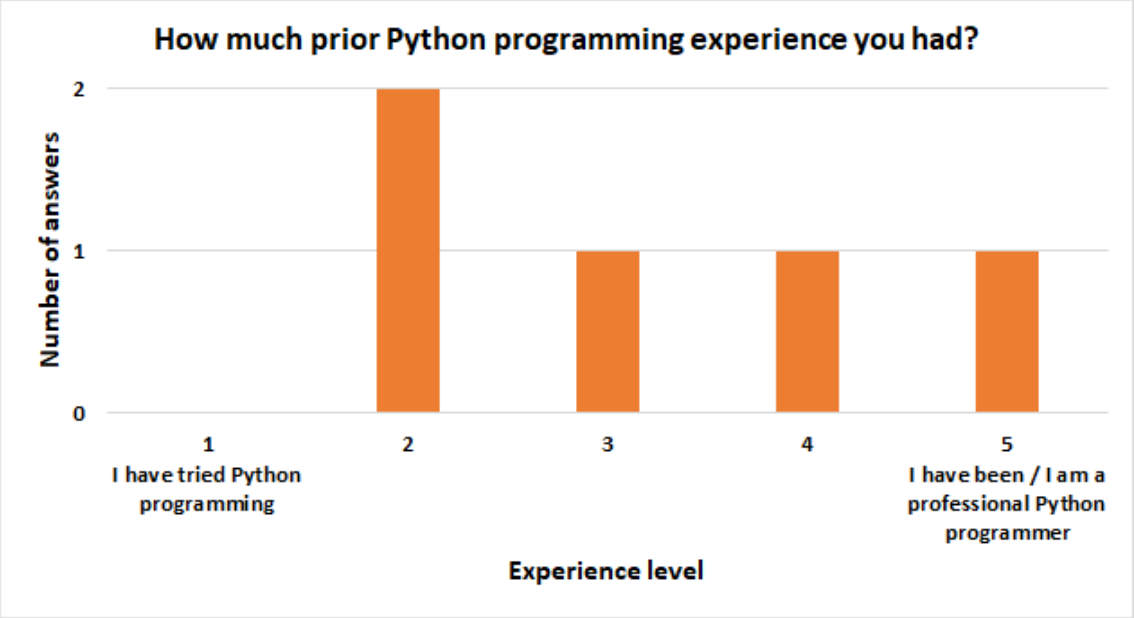

Figure 18. Prior Python programming experience.

The fourth question was about changes made based on the ASPA feedback. While it was not required to make any changes, four testers did changes on following categories:

1. Added header comments
2. Removed global variable
3. Added missing `return` statements and return values
4. Closed left-open files
5. Added exception handling to file operations (opening, reading or writing)
6. Added exception type
7. Added second exception branch

The fifth question was about how ASPA effects on time used for assignments. Both increasing and decreasing arguments were reported. The first argument about increasing time concerned checks which ASPA does, but which are not necessary required in order to complete the weekly assignments, in addition, Tester 3 stated:

> "For me [the tool] increased the used time, but for less experienced programmers the time would likely be shorter" — *Tester 3*

Rest of the reported arguments were with the decreased time, e.g. by highlighting that locating coding convention violations faster would decrease the overall time, and the prior experience could speed up completing of later assignments. Tester 2 summarised:

> "ASPA make it easier to detect violations, and thus speeds up programming." — *Tester 2*

The sixth question focused on effects on self-studying. There was an comment that participant wanted to get rid of all the violation messages, and in case of an unclear feedback message participant would have searched for the solution. Rest of the comments were emphasising that ASPA guides programmer towards better programming style, e.g. Tester 4 commented:

> "ASPA support learning of good style conventions, which might have been missed while reading the programming guide." — *Tester 4*

The seventh question was a question matrix of three likert 5 scale questions. All the questions were about ease and clearness of ASPA usage. Answer distributions are shown in Figure 19.

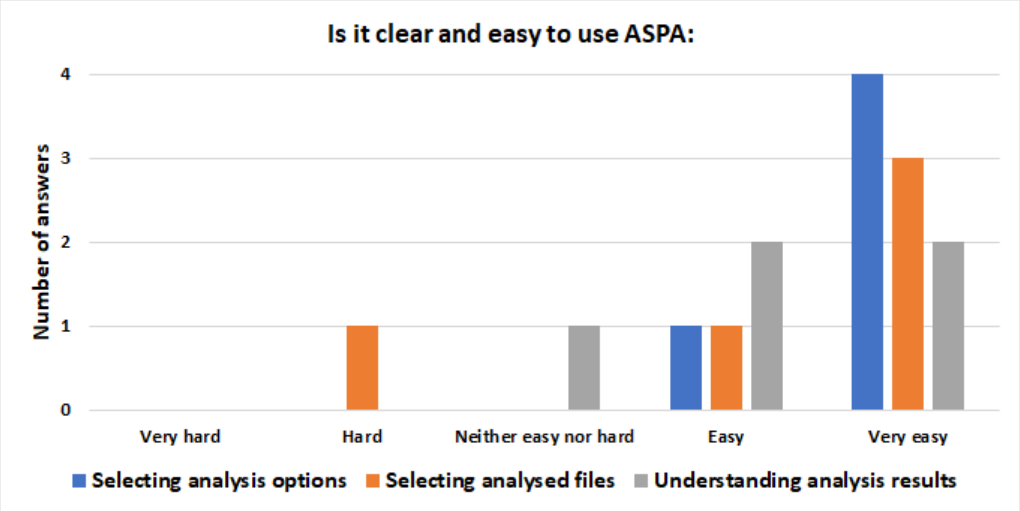

Figure 19. Ease and clearness of ASPA usage.

The eighth question was about any anomalies in ASPA. However, not a single anomaly was reported. The ninth and tenth questions were about adding and changing functionality to ASPA, respectively. All the addition and change suggestions are as follows:

1. Info button to explain ASPA feedback. Would help inexperienced users and would be faster than checking from Google or style guide, i.e. fixing would be faster.
2. More dynamic analysis, e.g. automatic analysis once per minute to give continuous feedback.

3. Previously analysed files should be preserved to make analysis workflow smoother.
4. Colour coding to the feedback results, e.g. based on severity of the violation.
5. Presets could be more cleaner by having e.g. slider instead of buttons.
6. Results could be shown in the same window as settings.

The eleventh question was about using ASPA if given the opportunity. Answer distribution and the translated question are shown in Figure 20.

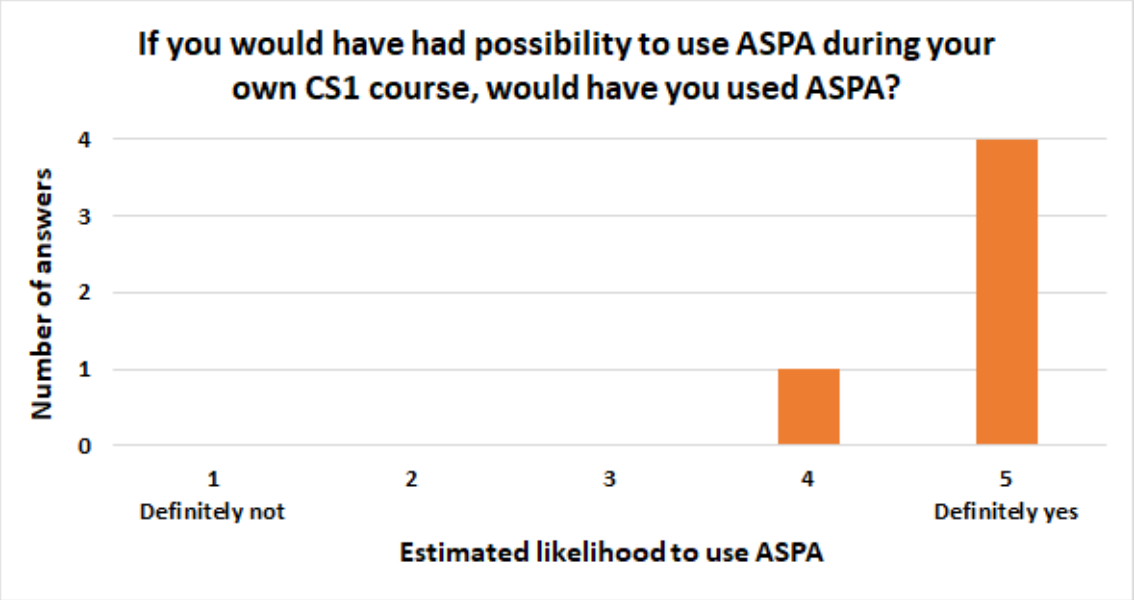

Figure 20. Likelihood to use ASPA if given the opportunity.

The final question was an open feedback. Open feedback was really positive. Usefulness of ASPA in general was praised as well as analysis options. Also positive effects to the next year results were speculated. In addition, Tester 4 said:

> "I would have needed such a tool while doing the course project." — *Tester 4*

On the other hand, more improvement suggestions as well as a note about content were given. Comments were as follows:

1. It would be easier to use ASPA as an IDE plugin, e.g. in Visual Studio Code. However, this might be too complicated for novice programmers.
2. Feedback is very clear, but only if user is familiar with the terminology.
3. Double asterisk at the beginning of the 'there is only one except branch' feedback message was not explained.

6 RESULTS AND DISCUSSION

In this section the summary of evaluation results is presented as well as discussion related to the results. In addition further development of ASPA, future work and future research are discussed.

Used evaluation methods were ASPA analysis and human grading results comparison, educational experts demonstration and interview, former teaching assistants testing and interview, and finally user testing from student point of view. In following tables these data sources are abbreviated as follows – result comparison (RC), educational expert (EE), teaching assistant (TA) and student (S).

6.1 Results and implications

The main results of the study and implications of them are shown in Table 23. The second last and the last results are also answers to the RQ1 and RQ2, respectively.

Table 23. Research results and implications.

Result	Implication
None of the studied code checker platforms support comprehensive guidance for students after a failed check.	A new or an existing tool need to be used to provide guidance for student.
AST based static analyser is suitable way to detect coding convention violations.	AST provides versatile configuration possibilities for detecting code structures and their usage. This is applies also to other programming languages than Python.
None of the found static analyser tools is providing feedback in Finnish.	A new tool need to be implemented in order to get a tool, which fits our course's needs.
A static analyser, such as ASPA, can be used to standardize and improve assessment feedback provided to the students.	A static analyser tool is a good option to improve assessment process on programming courses, because it speed up the process and still remains consistent.
There is evidence that the same static analyser could be used to assist students with their programming assignments, as well as, support learning via semi-continuous feedback.	More testing with students should be done in order to get more comprehensive results. Long run effects for learning need to be studied separately.

In addition, there are summarised evaluation results of ASPA presented. ASPA does have various effects to workflows of each user group. The mentioned and observed effects are shown in Table 24. The teaching assistant testing and user testing resulted comments related only to their respective user groups, while other methods covered all the user groups. There were also fully or partially conflicting effects mentioned.

Table 24. What are effects of ASPA based on evaluations.

ID	Effect	Based on	Is affected
E1	More examination feedback	EE	Lecturer [1]
E2	Consistent examination feedback format	EE	Lecturer [1]
E3	Consistency to examination feedback	RC	Lecturer [1]
E4	Timesavings with examination grading	EE	Lecturer
E5	Consistent course project feedback format	EE	TA[1]
E6	Consistency to course project feedback	TA, EE & RC	TA[1]
E7	Faster course project grading (with larger sample)	TA & EE	TA
E8	Easy to check evaluation result later	TA	TA
E9	No need to go through submissions multiple times	2 TAs	TA
E10	Inexperienced user time decrease [2]	S	Students
E11	Experienced user time increase [2]	S	Students
E12	Guide towards better coding conventions	S & EE	Students
E13	Increase code quality	EE	Students
E14	Improve learning via formative feedback	EE	Students
E15	Tool is easy for students to use [3]	S	Students
E16	Tool is too complicated for students to use [3]	EE	Students

[1] Students are indirectly affected.
[2] Partially conflicting effects.
[3] Conflicting effects.

Then there are the effects for each individual stakeholder. The comparison between analysis results of ASPA and feedback from human graders resulted multiple conclusions, but in short, human graders are currently better with amount of error types but not with consistency. This indicates that by adding new static analysis tests to ASPA it could dramatically improve the grading process. There were various style violations which were detected by ASPA, but not by human. At least human graders did not mention those violations in the feedback.

Expert from industry was interested in ASPA and ASTs in general. In addition, integrating ASPA to code checker platform was seen interesting and reasonable option. Regardless of industrial interest, this did not require any further actions in scope of this study. However, proof-of-concept version of the GUI was asked also in English, which was implemented as part of the general English support.

From lecturer point of view ASPA is ready to be used after adding more static analysis checks and providing copy paste ready feedback, in addition, increased feedback was expected to be improvement for students too. Adding more checks will not be an issue, but the requested feedback format is different from every other user groups format. On the other hand, the result comparison resulted many coding convention violations detected ASPA but not mentioned by lecturer. However, the comments from educational expert interview explain this difference. Style violations and minor non-style violations such as a single file left is seldom major issues in the examination. They do not affect the grading much, which is the main reason that these are not always pointed out by human grader in the feedback.

For the teaching assistant point of view mostly results were mostly promising, i.e. ASPA works for the purpose it was designed. Teaching assistants and educational experts emphasised that ASPA would provide a solid foundation for consistency on violation detection, including faster grading process when the number of submissions increases. Consistency would enable higher quality grading in general, by removing majority of humane outliers. Moreover, as with the lecturer usage one educational expert mentioned consistent feedback format, which could be directly copy pasted as part of the grading feedback.

Furthermore, teaching assistant experts pointed out that ASPA would provide huge time-savings. Firstly, by providing convenient way to store analysis results. Secondly, by removing the need to go through submission multiple times, because results can be checked later on if needed. Both teaching assistant testers would definitely use ASPA to verify their grading results, because it was easy to use, found coding convention violations consistently and would most likely reduce the overall workload. Result correlate with Gerdes et al. (2010) and Higgins et al. (2005), who reported reduced workload while using a tool.

On student point of view, both experience and inexperienced user testers found ASPA rather easy to use in all the three cases. Selecting analysed files was only category with answer one hard, while all others were on easy or very easy. In other words, there were reported problems with confusing feedback which were reported with interpreters and compilers by Liu and Petersen (2019) and Becker et al. (2019). However, sample size was such small that statistical analysis is not feasible. On the other hand, educational experts argued that current version has way too buttons for student users. Optimally student version would have only two buttons – select a file and execute an analysis.

Based on user testing feedback ASPA affects on time but it can be either increasing or decreasing effect. Increase was stated to be due the weekly assignment requirements which

are less than what ASPA checks. In addition one tester stated that ASPA was slowing the process, but added that for inexperienced user tool may be beneficial. It can be hypothesised that this tester might not need such a tool, i.e. there was no need to improve the code and running ASPA was unnecessary overhead. Decreased time was due the faster violation locating. Both educational expert and user testers emphasized that guidance towards better coding conventions was a positive and working feature, which would probable improve learning of good coding conventions towards the end of the course. But, does the ASPA actually support students to find and fix coding convention violations?

In result comparison multiple coding convention violations were found, also teaching assistants found violations and educational expert were expecting that student should find violations with ASPA. User testing showed that code changes were made based on ASPA feedback, i.e. feedback was taken into account and violations were fixed. Therefore, there are strong evidence to support RQ2. In addition, ASPA was agreed to be useful for the weekly assignments, among user testers and every user tester would likely have used ASPA if given the opportunity during the course. Therefore, as educational experts mentioned, ASPA coud be a precheck tool before actual submission, and same time it would work as a formative assessment tool to support learning. Same concept is noted by Chow et al. (2017) and Choudhury et al. (2016), i.e. the formative feedback is beneficial to achieve deep understanding of a programming topic. Finally, it was suggested to have two separate versions, a simplified student version and more advanced lecturer and teaching assistant version, and at least the default GUI wanted to be more simple for student users.

ASPA has known limitations due the static nature of the tool, i.e. it can only detect those violation types it is checking in the first place. All the AST related limitations mentioned in Section 4.6 do apply to ASPA too. Then ASPA specific limitations are e.g. creating a detection that list is cleared at the end of the program is challenging. The list should be detected at first, then tracked in case it is assigned to the different variable, and finally `clear` method should be used for the list. The hard part is detection and tracking of the list.

During the study testers were seem to perform very well, which can be due the human nature that testers were focusing more, because they know being part of the test. This apply to all the test participants and similar behaviour could be seen from the literature. It seems in many previous studies, at average all the new methods were improving course pass rate and student's satisfaction. This raises a question, was it clear to students that they were measured, and therefore they focused on assignments even more and got better results.

Regardless of number of distributed versions, current limitations or well focused testers, the main take away from the evaluations was the great potential of ASPA for the purpose it was designed. To improve ASPA it need to be developed further, therefore, in next subsection implemented and planned actions are presented.

6.2 Suggested improvements

There were few bugs and crucial functionalities which were implemented directly after they were reported. In addition, couple changes to requirements and GUI were done with less high priority. These already implemented actions are shown in Table 25. Expert and tester suggestions, which are not yet implemented are shown in Table 26.

Table 25. Additions and changes implemented to ASPA based on evaluations.

ID	Change or addition already implemented	Based on
I1	Results page appends the results between analysis	TA
I2	The result file is stored in the path from where ASPA is executed, not in the path where the executed file is stored	TA
I3	Exit button was removed	TA
I4	Buttons at the bottom centered	TA
I5	GUI in English	Viope
I6	Basic statistics	2 EEs
I7	Line number fix with except node	EE
I8	Added 10 new static analysis subrequirements	RC
I9	Fixed file closing violation message to point correct file handle	RC
I10	Fixed false positive file left open, when opened and closed in else branch	RC
I11	Fixed false positive file left open in when closed after multiple branches	RC
I12	Fixed false positive missing exception handling with local function names read and write	RC
I13	Changes message when object was named similarly as class	RC
I14	Modified check such that exception handling `else` branch is not exception handling	RC

In addition, based on evaluations it is not reasonable to detect `return` statement at the middle of the function and multiple exceptions violations, while they are not added also to the programming guide. Therefore, there should be better synchronisation between ASPA and programming guide to actually benefit students.

During the educational expert interview great potential of ASPA was recognised and various ideas for further development were given. From new ideas, the basic statistics were

Table 26. What are suggested changes and additions to ASPA based on evaluations.

ID	Change or addition	Proposer
C1	More static analysis tests	EE & 2 TAs
C2	Static analysis tests for each assignment separately	EE
C3	Change preset options to more precise	EE & TA
C4	Change preset options to clearer	S
C5	Clearer "no violations detected" message when nothing is detected	EE
C6	In the front page, add button to go directly to the result page	TA
C7	Larger font size to the selected file text box	TA
C8	Colour coding to the GUI's result page	2 TAs & S
C9	In the front page, add button to go directly to the result page	EE & 2 TAs
C10	Clearer separator between two files	2 TAs
C12	Even clearer separator between two directories	TA
C13	Possibility to select a new file directly from the result page	TA
C14	Use either forward or back slashes in the shown file path, not mix	TA
C15	Reduce size of analysis selection checkboxes	TA
C16	Add possibility to change font size	TA
C17	In menubar under file menu, add option to get back to the front page	TA
C18	Add own icon to ASPA, i.e. not the default Tkinter icon	TA
C19	Change theme, the default Windows 10 theme is not very good looking	TA
C20	Results could be shown in the same window as settings	TA & S
C21	Explain double asterisk at the beginning of the feedback messages	2 TAs & S
C22	Configurable examination feedback format	EE
C23	Better result presentation	2 TAs
C24	For student use, more simplified GUI	2 EEs
C25	Info button to explain ASPA feedback	TA & 2 Ss
C26	For students automatic or semiautomatic analysis selection	3 EEs
C27	More dynamic analysis, e.g. automatic analysis once per minute to give continuous feedback	S
C28	ASPA as an IDE plugin, e.g. in Visual Studio Code	S
C29	ASPA feedback should point to respective section in programming guide	EE

implemented after the interview session. However, all the other further development ideas should be at least considered, while implementing them is not in scope of this study, but they are discussed in the next section, Section 6.3.

6.3 Future work

In this section future work is presented and discussed, including the future development of ASPA and the future research. Because evaluation results were promising, a pilot test with real students users on the summer course is about to start. The pilot test will last until end of the summer period, and after that but before autumn semester, ASPA is planned to be deployed.

The first further development idea is a visual representation of the abstract syntax tree and the used algorithm. The addition could be more useful in advanced courses, e.g. data structures and algorithms course. Visualisation idea is similar to the previously mentioned TRAKLA2 platform, developed by Malmi et al. (2004). The second idea is a plagiarism detection, which could use visual or structural representations of the submissions. While plagiarism does not seem to be an issue all the time, it need to be monitored continuously. To improve monitoring structural analysis is good for such purpose, i.e. solution based on ASPA could be an suitable option for job.

The third idea is to add variety of statistics. In addition to total number of detected violations, ASPA could contain assignment, student and structure specific statistics. For example, how many times each student has used certain structure, and was is correctly or incorrectly used. Statistics would also enable detection of structures and conventions students are actually using and then refine teaching based on those. In extreme cases an experienced programmer or a lecturer may not even think about solutions students come up with. The fourth idea is to developed the third idea further to measure students learning. One method could be BKT analysis. BKT value can be used to measure learning (Corbett and Anderson, 1995). This could be utilised to observe which coding structures and conventions each student has learned, as well as, to get more detailed information which topics are hard for students.

The fifth topic is integration and utilisation of existing tools. Could Python debugger module, bdb, be utilised to analyse individual states of the program, i.e. extend ASPA to also work with run time values. Furthermore, it could be studied how to implement similar static analysis tool to work with other programming languages, such as C and Java, which are used language in advanced LUT CS courses. Or are existing tools, e.g. ones tested by Yulianto and Liem (2014), more feasible for C and Java education? On the other hand, how could industrial tools, e.g. professional IDEs, linter tools, libraries etc., be configured to do similar analyses? For example usage of Pylint's and its pylintrc file for configurations could improve student's knowledge of Pylint and Python in general.

7 CONCLUSION

Software tools are used in education to improve student learning outcomes as well as their satisfaction. On the other hand, course representatives have been utilising them for automatic assessment and analysis purpose for years. The goal of the study was to find or implement a tool to support course staff on assessment process of course project and electronic examination. The secondary goal was a tool to support learning by detecting common coding convention violations. The study found that abstract syntax trees can be used to conduct static analyses for source code without executing the code and it suits for the both purposes.

Therefore, the implemented tool, ASPA, is a static analyser. It utilises Python ast library to analyse syntactic structure of student submissions. ASPA has dedicated GUI to ensure convenient workflow for students. On the contrary to other similar tools ASPA has GUI and given feedback available with Finnish language. Other known tools support either English or other non-Finnish language.

Analysis result comparison indicated that ASPA does detect coding convention violations consistently. However due the static nature of the tool, each coding convention needs a individually implemented static analysis test. On the other hand, based on user testing, educational expert interview and analysis result comparison ASPA satisfy goals, in all but usability aspect. The GUI should be simplified for student users. However, current version of ASPA was considered even more suitable for teaching assistant and lecturers. Therefore, the answer for RQ1 *Can a static analyser be used to standardise assessment feedback of programming assignments?* is strong yes. On the other hand, the answer to the RQ2: *Can static analyser be used to assist students with programming assignments?* is currently unclear but, there is strong evidence that is should be possible.

As mentioned the main weakness is the usability of the GUI, which was estimated to complicated for the student users, by educational experts. However, user testing resulted opposite. In addition, as analysis result comparison and teaching assistant testing resulted, more static analysis checks are needed to cover all core coding conventions taught in LUT CS1 course.

The main take away from the study is the great potential of ASPA for the purpose it was designed. Even the expert from industry was exited about potential of ASPA. It was decided to develop ASPA further for both student self-study and supportive grader use to satisfy needs of all user groups even better.

REFERENCES

A+ LMS (2019). Aalto University, A+ LMS The extendable learning management system. [Online] https://apluslms.github.io/. [Accessed 18 March 2020].

Aleksić, V. and Ivanovic, M. (2016). Introductory Programming Subject in European Higher Education. *INFORMATICS IN EDUCATION*, 15(2):163–182.

Allen, J. M., Vahid, F., Downey, K., and Edgcomb, A. D. (2018). Weekly Programs in a CS1 Class: Experiences with Auto-graded Many-small Programs (MSP). In *2018 ASEE Annual Conference & Exposition*, page 13, Salt Lake City, Utah. ASEE Conferences.

Allen, J. M., Vahid, F., Edgcomb, A., Downey, K., and Miller, K. (2019). An Analysis of Using Many Small Programs in CS1. In *Proceedings of the 50th ACM Technical Symposium on Computer Science Education*, SIGCSE '19, pages 585–591, Minneapolis, MN, USA. Association for Computing Machinery.

Annamaa, A. (2015). Introducing Thonny, a Python IDE for learning programming. In *Proceedings of the 15th Koli Calling Conference on Computing Education Research - Koli Calling '15*, pages 117–121, Koli, Finland. ACM Press.

Baskerville, R., Pries-Heje, J., and Venable, J. (2009). Soft design science methodology. In *Proceedings of the 4th International Conference on Design Science Research in Information Systems and Technology*, DESRIST '09, page 11, Philadelphia, Pennsylvania. Association for Computing Machinery.

Bastian, M., Heymann, S., and Jacomy, M. (2009). Gephi: An open source software for exploring and manipulating networks. In *International AAAI Conference on Weblogs and Social Media*.

Becker, B. A. (2016). An Effective Approach to Enhancing Compiler Error Messages. In *Proceedings of the 47th ACM Technical Symposium on Computing Science Education*, SIGCSE '16, pages 126–131, Memphis, Tennessee, USA. Association for Computing Machinery.

Becker, B. A., Denny, P., Pettit, R., Bouchard, D., Bouvier, D. J., Harrington, B., Kamil, A., Karkare, A., McDonald, C., Osera, P.-M., Pearce, J. L., and Prather, J. (2019). Unexpected Tokens: A Review of Programming Error Messages and Design Guidelines for the Future. In *Proceedings of the 2019 ACM Conference on Innovation and Technology in Computer Science Education*, ITiCSE '19, pages 253–254, Aberdeen, Scotland Uk. Association for Computing Machinery.

Becker, B. A. and Quille, K. (2019). 50 Years of CS1 at SIGCSE: A Review of the Evolution of Introductory Programming Education Research. In *Proceedings of the 50th ACM Technical Symposium on Computer Science Education*, SIGCSE '19, pages 338–344, Minneapolis, MN, USA. Association for Computing Machinery.

Bennedsen, J. and Caspersen, M. E. (2007). Failure rates in introductory programming. *ACM SIGCSE Bulletin*, 39(2):32–36.

Bennedsen, J. and Caspersen, M. E. (2019). Failure rates in introductory programming: 12 years later. *ACM Inroads*, 10(2):30–36.

Bruce, K. B., Danyluk, A. P., and Murtagh, T. P. (2001). Event-driven programming is simple enough for CS1. In *Proceedings of the 6th annual conference on Innovation and technology in computer science education*, ITiCSE '01, pages 1–4, Canterbury, United Kingdom. Association for Computing Machinery.

Choudhury, R. R., Yin, H., and Fox, A. (2016). Scale-Driven Automatic Hint Generation for Coding Style. In *Proceedings of the 13th International Conference on Intelligent Tutoring Systems - Volume 9684*, ITS 2016, pages 122–132, Zagreb, Croatia. Springer-Verlag.

Chow, S., Yacef, K., Koprinska, I., and Curran, J. (2017). Automated Data-Driven Hints for Computer Programming Students. In *Adjunct Publication of the 25th Conference on User Modeling, Adaptation and Personalization*, UMAP '17, pages 5–10, Bratislava, Slovakia. Association for Computing Machinery.

Corbett, A. T. and Anderson, J. R. (1995). Knowledge tracing: Modeling the acquisition of procedural knowledge. *User Modelling and User-Adapted Interaction*, 4(4):253–278.

Croft, D. and England, M. (2020). Computing with CodeRunner at Coventry University: Automated summative assessment of Python and C++ code. In *Proceedings of the 4th Conference on Computing Education Practice 2020*, CEP 2020, pages 1–4, Durham, United Kingdom. Association for Computing Machinery.

Cukierman, D. R., McGee Thompson, D., and Sun, W. (2019). The Academic Enhancement Program: Assessing Programs Designed to Support Student Success. In *Proceedings of the 50th ACM Technical Symposium on Computer Science Education*, SIGCSE '19, pages 686–692, Minneapolis, MN, USA. Association for Computing Machinery.

Edwards, S. H., Kandru, N., and Rajagopal, M. B. (2017). Investigating Static Analysis Errors in Student Java Programs. In *Proceedings of the 2017 ACM Conference on*

International Computing Education Research, ICER '17, pages 65–73, Tacoma, Washington, USA. Association for Computing Machinery.

Fagin, B. and Merkle, L. (2003). Measuring the effectiveness of robots in teaching computer science. In *Proceedings of the 34th SIGCSE technical symposium on Computer science education*, SIGCSE '03, pages 307–311, Reno, Navada, USA. Association for Computing Machinery.

Figueiredo, J. and García-Peñalvo, F. J. (2018). Building Skills in Introductory Programming. In *Proceedings of the Sixth International Conference on Technological Ecosystems for Enhancing Multiculturality*, TEEM'18, pages 46–50, Salamanca, Spain. Association for Computing Machinery.

Figueiredo, J., Lopes, N., and García-Peñalvo, F. J. (2019). Predicting Student Failure in an Introductory Programming Course with Multiple Back-Propagation. In *Proceedings of the Seventh International Conference on Technological Ecosystems for Enhancing Multiculturality*, TEEM'19, pages 44–49, León, Spain. Association for Computing Machinery.

Foxley, E., Higgins, C., Hegazy, T., Symeonidis, P., and Tsintsifas, A. (2001). The CourseMaster CBA System: Improvements over Ceilidh. In *Proceedings of the 5th CAA Conference, Loughborough*, page 13. Loughborough University.

Fu, X., Shimada, A., Ogata, H., Taniguchi, Y., and Suehiro, D. (2017). Real-time learning analytics for C programming language courses. In *Proceedings of the Seventh International Learning Analytics & Knowledge Conference*, LAK '17, pages 280–288, Vancouver, British Columbia, Canada. Association for Computing Machinery.

Garg, N. and Keen, A. W. (2018). Earthworm: Automated Decomposition Suggestions. In *Proceedings of the 18th Koli Calling International Conference on Computing Education Research - Koli Calling '18*, pages 1–5, Koli, Finland. ACM Press.

Geerts, G. L. (2011). A design science research methodology and its application to accounting information systems research. *International Journal of Accounting Information Systems*, 12(2):142–151.

Gerdes, A., Jeuring, J. T., and Heeren, B. J. (2010). Using strategies for assessment of programming exercises. In *Proceedings of the 41st ACM technical symposium on Computer science education - SIGCSE '10*, page 441, Milwaukee, Wisconsin, USA. ACM Press.

GitHub (2019). The State of the Octoverse. [Online] https://octoverse.github.com/. [Accessed 9 March 2020].

Goldwasser, M. H. and Letscher, D. (2008). Teaching an object-oriented CS1 — with Python. In *Proceedings of the 13th annual conference on Innovation and technology in computer science education*, ITiCSE '08, pages 42–46, Madrid, Spain. Association for Computing Machinery.

Grok Learning, P. L. (2020). Grok Learning. Library Catalog: groklearning.com.

Guo, P. J. (2013). Online Python tutor: embeddable web-based program visualization for cs education. In *Proceeding of the 44th ACM technical symposium on Computer science education*, SIGCSE '13, pages 579–584, Denver, Colorado, USA. Association for Computing Machinery.

Hakulinen, L., Auvinen, T., and Korhonen, A. (2013). Empirical Study on the Effect of Achievement Badges in TRAKLA2 Online Learning Environment. In *2013 Learning and Teaching in Computing and Engineering*, pages 47–54.

Hertz, M. (2010). What do "CS1" and "CS2" mean? investigating differences in the early courses. In *Proceedings of the 41st ACM technical symposium on Computer science education*, SIGCSE '10, pages 199–203, Milwaukee, Wisconsin, USA. Association for Computing Machinery.

Hevner, A. R., March, S. T., Park, J., and Ram, S. (2004). Design Science in Information Systems Research. *MIS quarterly*, 28(1):75–105.

Higgins, C. A., Gray, G., Symeonidis, P., and Tsintsifas, A. (2005). Automated assessment and experiences of teaching programming. *Journal on Educational Resources in Computing (JERIC)*, 5(3):5–es.

Jenkins, T. (2002). On the difficulty of learning to program. In *Proceedings of the 3rd Annual Conference of the LTSN*, pages 53–58.

Joint Task Force for Computing Curricula (2005). Computing Curricula 2005 The Overview Report. [Online] https://www.acm.org/binaries/content/assets/education/curricula-recommendations/cc2005-march06final.pdf. [Accessed 9 June 2020].

Journault, M., Miné, A., and Ouadjaout, A. (2018). Modular static analysis of string manipulations in C programs. In *International Static Analysis Symposium*, pages 243–262. Springer.

Ju, A., Mehne, B., Halle, A., and Fox, A. (2018). In-class coding-based summative assessments: tools, challenges, and experience. In *Proceedings of the 23rd Annual ACM Conference on Innovation and Technology in Computer Science Education*, ITiCSE 2018, pages 75–80, Larnaca, Cyprus. Association for Computing Machinery.

Kaila, E., Laakso, M.-J., Rajala, T., and Kurvinen, E. (2018). A model for gamifying programming education: University-level programming course quantified. In *2018 41st International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO)*, pages 0689–0694, Opatija. IEEE.

Karavirta, V., Ihantola, P., and Koskinen, T. (2013). Service-Oriented Approach to Improve Interoperability of E-Learning Systems. In *2013 IEEE 13th International Conference on Advanced Learning Technologies*, pages 341–345. ISSN: 2161-377X.

Karvelas, I. (2019). Investigating Novice Programmers' Interaction with Programming Environments. In *Proceedings of the 2019 ACM Conference on Innovation and Technology in Computer Science Education*, ITiCSE '19, pages 336–337, Aberdeen, Scotland Uk. Association for Computing Machinery.

Kasurinen, J. (2006). Python as a programming language for the introductory programming courses. Bachelor's thesis, Lappeenranta University of Technology.

Keuning, H., Heeren, B., and Jeuring, J. (2017). Code Quality Issues in Student Programs. In *Proceedings of the 2017 ACM Conference on Innovation and Technology in Computer Science Education*, ITiCSE '17, pages 110–115, Bologna, Italy. Association for Computing Machinery.

Kinnunen, P. and Malmi, L. (2006). Why students drop out CS1 course? In *Proceedings of the second international workshop on Computing education research*, ICER '06, pages 97–108, Canterbury, United Kingdom. Association for Computing Machinery.

Kinnunen, P. and Malmi, L. (2008). CS minors in a CS1 course. In *Proceedings of the Fourth international Workshop on Computing Education Research*, ICER '08, pages 79–90, Sydney, Australia. Association for Computing Machinery.

Koulouri, T., Lauria, S., and Macredie, R. D. (2015). Teaching Introductory Programming: A Quantitative Evaluation of Different Approaches. *ACM Transactions on Computing Education (TOCE)*, 14(4):26:1–26:28.

Lawhead, P. B., Duncan, M. E., Bland, C. G., Goldweber, M., Schep, M., Barnes, D. J., and Hollingsworth, R. G. (2002). A road map for teaching introductory programming using LEGO© mindstorms robots. In *Working group reports from ITiCSE on Innovation and technology in computer science education*, ITiCSE-WGR '02, pages 191–201, Aarhus, Denmark. Association for Computing Machinery.

Liu, D. and Petersen, A. (2019). Static Analyses in Python Programming Courses. In *Proceedings of the 50th ACM Technical Symposium on Computer Science Education - SIGCSE '19*, pages 666–671, Minneapolis, MN, USA. ACM Press.

Luukkainen, R. (2020). RoopeLuukkainen/ASPA: ASPA proof-of-concept release. Zenodo http://doi.org/10.5281/zenodo.3898125.

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. (2018). Introductory programming: a systematic literature review. In *Proceedings Companion of the 23rd Annual ACM Conference on Innovation and Technology in Computer Science Education*, ITiCSE 2018 Companion, pages 55–106, Larnaca, Cyprus. Association for Computing Machinery.

Malmi, L., Karavirta, V., Korhonen, A., Nikander, J., Seppälä, O., and Silvasti, P. (2004). Visual algorithm simulation exercise system with automatic assessment: TRAKLA2. *Informatics in education*, 3(2):267. Publisher: Institute of Mathematics and Informatics.

Navrat, P. and Tvarozek, J. (2014). Online programming exercises for summative assessment in university courses. In *Proceedings of the 15th International Conference on Computer Systems and Technologies*, CompSysTech '14, pages 341–348, Ruse, Bulgaria. Association for Computing Machinery.

Nielsen, J. (2000). Nielsen Norman Group, Why You Only Need to Test with 5 Users. [Online] https://www.nngroup.com/articles/why-you-only-need-to-test-with-5-users/. [Accessed 3 April 2020].

Nielsen, J. and Landauer, T. K. (1993). A mathematical model of the finding of usability problems. In *Proceedings of the INTERACT '93 and CHI '93 Conference on Human Factors in Computing Systems*, CHI '93, pages 206–213, Amsterdam, The Netherlands. Association for Computing Machinery.

Niemela, P. and Hyyro, H. (2019). Migrating Learning Management Systems Towards Microservice Architecture. In *Joint Proceedings of the Inforte Summer School on Software Maintenance and Evolution (SSSME-2019)*, page 11, Tampere, Finland.

Nikula, U., Alaoutinen, S., Kasurinen, J., and Pirinen, T. (2009). Improving the Technical Infrastructure of a Programming Course. In *2009 Ninth IEEE International Conference on Advanced Learning Technologies*, pages 374–376. ISSN: 2161-377X.

Nikula, U., Gotel, O., and Kasurinen, J. (2011). A Motivation Guided Holistic Rehabilitation of the First Programming Course. *ACM Transactions on Computing Education*, 11(4):1–38.

Nunamaker, J. F., Chen, M., and Purdin, T. D. (1990). Systems Development in Information Systems Research. *Journal of Management Information Systems*, 7(3):89–106.

Olsen, J. K. and Fox, A. (2019). Usage of Hints on Coding-Based Summative Assessments. In *Proceedings of the 50th ACM Technical Symposium on Computer Science Education*, SIGCSE '19, pages 839–844, Minneapolis, MN, USA. Association for Computing Machinery.

Peffers, K., Tuunanen, T., Rothenberger, M. A., and Chatterjee, S. (2007). A Design Science Research Methodology for Information Systems Research. *Journal of Management Information Systems*, 24(3):45–77.

Porter, L., Bailey Lee, C., Simon, B., and Zingaro, D. (2011). Peer instruction: do students really learn from peer discussion in computing? In *Proceedings of the seventh international workshop on Computing education research*, ICER '11, pages 45–52, Providence, Rhode Island, USA. Association for Computing Machinery.

Rajala, T., Kaila, E., and Laakso, M.-J. (2020). University of Turku, ViLLE Collaborative education tool. [Online] https://ville.utu.fi/. [Accessed 8 April 2020].

Rapoport, M., Tip, F., and Lhoták, O. (2015). Precise Data Flow Analysis in the Presence of Correlated Method Calls. In *Static Analysis - 22nd International Symposium, SAS 2015*, pages 54–71, Berlin, Heidelberg. Springer Berlin Heidelberg. Series Title: Lecture Notes in Computer Science.

Rogers, S., Tang, S., and Canny, J. (2014). ACCE: automatic coding composition evaluator. In *Proceedings of the first ACM conference on Learning @ scale conference*, L@S '14, pages 191–192, Atlanta, Georgia, USA. Association for Computing Machinery.

Salminen, M. (2020). University of Oulu, Lovelace. [Online] https://lovelace.oulu.fi/. [Accessed 18 March 2020].

Schulte, C. and Bennedsen, J. (2006). What do teachers teach in introductory programming? In *Proceedings of the second international workshop on Computing education research*, ICER '06, pages 17–28, Canterbury, United Kingdom. Association for Computing Machinery.

Sheard, J., Simon, Carbone, A., D'Souza, D., and Hamilton, M. (2013). Assessment of programming: pedagogical foundations of exams. In *Proceedings of the 18th ACM conference on Innovation and technology in computer science education*, ITiCSE '13, pages 141–146, Canterbury, England, UK. Association for Computing Machinery.

Shuhidan, S., Hamilton, M., and D'Souza, D. (2009). A taxonomic study of novice programming summative assessment. In *Proceedings of the Eleventh Australasian Conference on Computing Education - Volume 95*, ACE '09, pages 147–156, Wellington, New Zealand. Australian Computer Society, Inc.

Shuhidan, S., Hamilton, M., and D'Souza, D. (2011). Understanding novice programmer difficulties via guided learning. In *Proceedings of the 16th annual joint conference on Innovation and technology in computer science education*, ITiCSE '11, pages 213–217, Darmstadt, Germany. Association for Computing Machinery.

Simon, Chinn, D., de Raadt, M., Philpott, A., Sheard, J., Laakso, M.-J., D'Souza, D., Skene, J., Carbone, A., Clear, T., Lister, R., and Warburton, G. (2012). Introductory programming: examining the exams. In *Proceedings of the Fourteenth Australasian Computing Education Conference - Volume 123*, ACE '12, pages 61–70, Melbourne, Australia. Australian Computer Society, Inc.

Simon, Luxton-Reilly, A., Ajanovski, V. V., Fouh, E., Gonsalvez, C., Leinonen, J., Parkinson, J., Poole, M., and Thota, N. (2019). Pass Rates in Introductory Programming and in other STEM Disciplines. In *Proceedings of the Working Group Reports on Innovation and Technology in Computer Science Education*, ITiCSE-WGR '19, pages 53–71, Aberdeen, Scotland Uk. Association for Computing Machinery.

Smith, N., Richards, M., and Cabrero, D. G. (2018). Summer of code: assisting distance-learning students with open-ended programming tasks. In *Proceedings of the 23rd Annual ACM Conference on Innovation and Technology in Computer Science Education*, ITiCSE 2018, pages 224–229, Larnaca, Cyprus. Association for Computing Machinery.

Sorva, J., Karavirta, V., and Malmi, L. (2013). A Review of Generic Program Visualization Systems for Introductory Programming Education. *ACM Transactions on Computing Education (TOCE)*, 13(4):1–64.

Tempelaar, D. T., Heck, A., Cuypers, H., van der Kooij, H., and van de Vrie, E. (2013). Formative assessment and learning analytics. In *Proceedings of the Third International Conference on Learning Analytics and Knowledge*, LAK '13, pages 205–209, Leuven, Belgium. Association for Computing Machinery.

Tew, A. E. and Guzdial, M. (2011). The FCS1: a language independent assessment of CS1 knowledge. In *Proceedings of the 42nd ACM technical symposium on Computer science education*, SIGCSE '11, pages 111–116, Dallas, TX, USA. Association for Computing Machinery.

TIM (2020). University of Jyväskylä, Introduction to TIM. [Online] https://tim.jyu.fi/view/tim/TIM-esittely/en. [Accessed 18 March 2020].

TMC (2019a). Aalto University, TestMyCode. [Online] http://testmycode.github.io/. [Accessed 18 March 2020].

TMC (2019b). Aalto University, TestMyCode Programming assignment evaluator. [Online] https://tmc.mooc.fi/. [Accessed 18 March 2020].

Vihavainen, A., Airaksinen, J., and Watson, C. (2014). A systematic review of approaches for teaching introductory programming and their influence on success. In *Proceedings of the tenth annual conference on International computing education research*, ICER '14, pages 19–26, Glasgow, Scotland, United Kingdom. Association for Computing Machinery.

Viope (2020). Viope Education LTD, Solutions for blended learning. [Online] https://www.viope.com/viope-en. [Accessed 8 April 2020].

Watson, C. and Li, F. W. (2014). Failure rates in introductory programming revisited. In *Proceedings of the 2014 conference on Innovation & technology in computer science education*, ITiCSE '14, pages 39–44, Uppsala, Sweden. Association for Computing Machinery.

Wieringa, R. (2009). Design Science as Nested Problem Solving. In *Proceedings of the 4th International Conference on Design Science Research in Information Systems and Technology*, DESRIST '09, New York, NY, USA. Association for Computing Machinery. event-place: Philadelphia, Pennsylvania.

Wiese, E. S., Yen, M., Chen, A., Santos, L. A., and Fox, A. (2017). Teaching Students to Recognize and Implement Good Coding Style. In *Proceedings of the Fourth (2017) ACM Conference on Learning @ Scale - L@S '17*, pages 41–50, Cambridge, Massachusetts, USA. ACM Press.

Wohlin, C., Runeson, P., Höst, M., Ohlsson, M. C., Regnell, B., and Wesslén, A. (2012). *Experimentation in Software Engineering*. Springer Berlin Heidelberg, Berlin, Heidelberg.

Yulianto, S. V. and Liem, I. (2014). Automatic grader for programming assignment using source code analyzer. In *2014 International Conference on Data and Software Engineering (ICODSE)*, pages 1–4. ISSN: null.

Zingaro, D., Cherenkova, Y., Karpova, O., and Petersen, A. (2013). Facilitating code-writing in PI classes. In *Proceeding of the 44th ACM technical symposium on Computer science education*, SIGCSE '13, pages 585–590, Denver, Colorado, USA. Association for Computing Machinery.

Zingaro, D. and Porter, L. (2014). Peer instruction: a link to the exam. In *Proceedings of the 2014 conference on Innovation & technology in computer science education*, ITiCSE '14, pages 255–260, Uppsala, Sweden. Association for Computing Machinery.

Appendix 1. Teaching assistant interview questions

1. Do you think ASPA has any effect on evaluation time? If yes, what effect?
2. Do you think ASPA has any effect on evaluation quality? If yes, what effect?
3. Do you think ASPA has any effect on evaluation result? If yes, what effect?
4. Were there any anomalies in ASPA functionality? If yes, what?
5. Was it clear how to use ASPA:
    a) Can correct analysis items be selected intuitively?
    b) Can (batch of) analysed files be selected intuitively?
    c) Are static analysis results clear and unambiguous enough?
6. Is there any functionality you would add to ASPA? If yes, what functionality?
7. Is there anything in ASPA that you would change? If yes, what?
8. Would you use the ASPA to support your grading process?
9. Open feedback, e.g. any other comments related to ASPA and its functionality.

Appendix 2. User testing questionnaire

1. How much prior Python programming experience you had? *

	1	2	3	4	5	
I have tried Python programming	O	O	O	O	O	I have been / I am a professional Python programmer

2. How many times you used ASPA during the L05-T3 assignment? *
3. How many times you used ASPA during the L09-T1 assignment? *
4. Did you do any changes based on the ASPA feedback? If you did, what changes? *
5. Do you think ASPA has any effect on time used for assignments? If yes, what effect? *
6. Do you think ASPA has any effect on self-studying programming topics used in the assignments? If yes, what effect? *
7. Was it clear how to use ASPA: *

	Very hard / Very unclear	Hard / unclear	Neither easy nor hard	Easy / Clear	Very easy / Very clear
Selecting analysis options	O	O	O	O	O
Selecting analysed files	O	O	O	O	O
Understanding analysis results	O	O	O	O	O

8. Were there any anomalies in ASPA functionality? If yes, what?
9. Is there any functionality you would add to the artefact? If yes, what functionality?
10. Is there anything in the ASPA that you would change? If yes, what?
11. If you would have had possibility to use ASPA during your own CS1 course, would have you used ASPA? *

	1	2	3	4	5	
Definitely not	O	O	O	O	O	Definitely yes

12. Open feedback, e.g. any other comments related to ASPA and its functionality.