An Intelligent Tutoring System for a Virtual Reality Procedural Training Project

Master Thesis

Ivans Ribakovs

Madrid, June 2018
This thesis is submitted to the ETSI Informáticos at Universidad Politécnica de Madrid in partial fulfillment of the requirements for the degree of Master of Science in Software Engineering.

Master Thesis
Master Universitario en Ingeniería del Software – European Master in Software Engineering
Thesis Title: An Intelligent Tutoring System for a Virtual Reality Procedural Training Project
Thesis no: EMSE-2018-4
June 2018

Author: Ivans Ribakovs
BSc Computer Science
Newcastle University

Supervisor: Angelica de Antonio Jimenez
Associate Professor
Politécnica de Madrid University
DLSIIS
Computer Science School
Politécnica de Madrid University

Co-supervisor: Jaime Ramirez Rodriguez
Associate Professor
Politécnica de Madrid University

ETSII Informáticos
Universidad Politécnica de Madrid
Campus de Montegancedo, s/n
28660 Boadilla del Monte (Madrid)
Spain
Acknowledgement

I would like to thank my master advisors Prof. Jaime Ramirez and Prof. Angelica de Antonio of the School of Computer Science at UPM. The door to their offices was always open whenever I ran into a trouble or had a question about my work or writing. They consistently allowed this master work to be my own work, but steered me in the right the direction whenever they thought I needed it.

I would also like to thank Diego Riofrío, author of the PhD thesis that this master was hugely base on, for the support he has offered when dealing with results of his work.
# Table of Contents

1  INTRODUCTION ......................................................................................................................... 1

2  INTELLIGENT TUTORING IN VIRTUAL REALITY FOR PROCEDURAL TRAINING – A STATE OF THE ART ......................................................................................................................... 3

  2.1  INTELLIGENT TUTORING SYSTEMS ................................................................................. 3
  2.2  VR SYSTEMS FOR PROCEDURAL TRAINING .................................................................. 4
  2.3  REACTIVE TUTOR ............................................................................................................... 6

3  SIEMA: VIRTUAL REALITY TRAINING OF CAR SEAT ASSEMBLY WORKERS ....................... 8

  3.1  OBJECTIVES ....................................................................................................................... 9
  3.2  DESCRIPTION OF THE GENERIC PROCESS FOR MANUFACTURING A CAR SEAT ................................................................................................................................. 10
  3.3  SYSTEM ARCHITECTURE ................................................................................................. 11
  3.3.1  Component and connector view ..................................................................................... 12
  3.3.1.1  Simulator instance per student .................................................................................. 12
  3.3.1.2  Shared simulator instance ....................................................................................... 13
  3.3.1.3  Comparison ............................................................................................................... 13
  3.3.2  Top modular decomposition of the simulator ................................................................ 13

4  SOFTWARE ENGINEERING PRACTICES FOR A VIRTUAL REALITY PROJECT ............... 16

  4.1  SOFTWARE LIFECYCLE .................................................................................................... 16
  4.1.1  Software lifecycle state of the art ................................................................................. 16
  4.1.2  Application in SIEMA project ..................................................................................... 17
  4.1.2.1  Virtual reality component software development lifecycle .................................. 17
  4.1.2.2  Intelligent tutoring system software development lifecycle .................................. 18
  4.2  CONTINUOUS INTEGRATION ........................................................................................ 19
  4.2.1  Continuous integration state of the art ....................................................................... 19
  4.2.2  Continuous Integration in SIEMA project ................................................................. 20
  4.3  CONFIGURATION MANAGEMENT .................................................................................. 21
  4.3.1  Configuration management state of the art ............................................................... 21
  4.3.2  Configuration management in SIEMA project ....................................................... 22
  4.4  TESTING ............................................................................................................................. 23
  4.4.1  Testing state of the art ............................................................................................... 23
  4.4.2  Testing in SIEMA project .......................................................................................... 23
  4.4.2.1  Testing tutor component ......................................................................................... 23
  4.4.2.2  Line coverage ........................................................................................................ 24
  4.5  CODE STYLE ...................................................................................................................... 24
  4.5.1  Code style state of the art ......................................................................................... 24
  4.5.2  Code style in SIEMA project .................................................................................... 25
  4.6  TEAM MANAGEMENT .................................................................................................... 26

5  AN INTELLIGENT TUTORING SYSTEM FOR SIEMA ......................................................... 27

  5.1  ITS ARCHITECTURE AND CURRENT STATE OF THE PROJECT ..................................... 27
  5.1.1  Reactive tutor ............................................................................................................. 28
  5.1.2  Collective student model ........................................................................................... 29
  5.1.2.1  Student action automaton .................................................................................... 31
  5.1.3  Tutoring coordinator .................................................................................................. 32
  5.2  PROBLEMS FOUND ...................................................................................................... 33
  5.2.1  Problems in the existing reactive tutor design and implementation .......................... 33
  5.2.1.1  Software practices ................................................................................................. 33
  5.2.1.2  Dependency on single configuration file library .................................................. 34
  5.2.1.3  .Net framework version incompatibility .................................................................. 34
  5.2.1.4  Singleton controller classes .................................................................................... 34
  5.2.1.5  Tutor message classification ................................................................................... 35
  5.2.1.6  Action validation .................................................................................................... 35
  5.2.1.7  Unity performance issues & LINQ ....................................................................... 38
  5.2.1.8  Logging .................................................................................................................. 39
  5.2.1.9  Future performance requirements ......................................................................... 39
  5.2.2  Problems in the collective model tutor design and implementation ......................... 40
  5.2.2.1  Non-reusable code for updating cluster automaton ............................................. 40
| 5.2.2.2 | Automaton implementation and issues with path finding | 40 |
| 5.2.2.3 | Pseudocode errors | 42 |
| 5.3 | DESIGN OF THE EXTENDED INTELLIGENT TUTORING SYSTEM | 43 |
| 5.3.1 | Package diagram | 43 |
| 5.3.2 | Class diagrams | 44 |
| 5.3.2.1 | Tutoring module | 44 |
| 5.3.2.2 | Collective model tutor – closer look | 46 |
| 5.3.3 | Sequence diagrams | 48 |
| 5.3.3.1 | Action validation | 48 |
| 5.3.3.2 | Get collective model tutor messages | 49 |
| 5.4 | DESCRIPTION OF CHANGES AND EXTENSIONS | 50 |
| 5.4.1 | Design of the tutoring coordinator | 50 |
| 5.4.2 | Performance improvements | 52 |
| 5.4.3 | Changes in action validation | 53 |
| 5.4.4 | Design of the collective model tutor | 54 |
| 5.4.4.1 | ZPD tutoring | 55 |
| 5.4.4.2 | Error prevention | 56 |
| 5.4.5 | New framework for tutor configuration properties | 60 |
| 6 | CONCLUSIONS AND FUTURE WORK | 62 |
| 6.1 | CONCLUSIONS | 62 |
| 6.1.1 | Unity 3D issues | 62 |
| 6.1.2 | Intelligent tutoring system issues | 63 |
| 6.2 | FUTURE WORK | 63 |
| 7 | REFERENCES | 65 |
1 Introduction

The key topic of this master work lies at the intersection of 2 different areas: intelligent tutoring systems and virtual reality. As such, this document will cover both areas and dive into some of the problems faced when implementing a software system at their intersection.

A real-life project, SIEMA, is the source of context, business needs and constraints for this master work. SIEMA was initiated by a car seat manufacturing company (further in the text - SeatInc), that among other things wants to improve following aspects of training program:

- Improve student knowledge of the domain vocabulary and assembly process steps.
- Increase number of students that can go through training program simultaneously.
- Increase the effective time that each student can spend practicing the assembly process.
- Reduce the number of new candidates that are rejected after attempting practical exercises.
- Reduce the time and effort required to re-train existing employees using new processes.

This master thesis will use above objectives as some of the drivers in order to demonstrate how automated procedural training can be applied to replace or assist human trainers in order to improve training efficiency.

Most of the work done in this master from the domain of intelligent tutoring systems was based on the results of PhD thesis[1] of another UPM alumni – Diego Riofrío. Diego Riofrío proposes a tutoring strategy (Collective Student Model) that applies educational data mining (EDM) to predict student performance and adjust tutoring based on the predictions. In addition, thesis outlines a strategy for combining multiple tutoring strategies in a single tutoring component using Tutoring Coordinator. This document will explain core ideas and concepts behind Collective Student Model and
Tutoring Coordinator and will pay special attention to implementation details such as design and problems faced and how they were solved.

This master work pursues following goals:

- Propose a viable system architecture for SIEMA project based on the goals of the project
- Implement tutoring strategy based on the collective student model
- Produce a viable tutoring coordinator implementation by combining tutoring strategy based on collective student model with existing reactive tutor.

Next, section 2 of this document addresses state of the art in the relevant fields while section 3 will focus on the SIEMA project and set the context in which further work will be performed. Section 4 of this document will present software engineering practices used during implementation of this project. Section 5 will address the area of Intelligent Tutoring Systems, introduce pre-existing components use for development, outline issues faced during the implementation and present solutions and extensions that were implemented in this master work in depth. Finally, section 6 will present conclusions and future work.
2 Intelligent Tutoring in Virtual Reality for Procedural Training – a State of the Art

2.1 Intelligent Tutoring Systems

Term “intelligent tutoring system” refer to computerized educational systems that attempt to mimic the capabilities of human tutors, that is to not only determine whether student responses are correct but also meaningfully analyse responses and provide personalized hints and tips to each of the students.

Intelligent Tutoring Systems is a fairly well-established domain with some of the earliest software artefacts in use since 1970s and critical review publications of a wide range of systems dating back to 1980s. There has been an evolution from a very primitive form of computer-assisted instruction, ranging through various forms of e-learning systems, progressing to form learner adaptive systems, to modern day ITS[2].

Traditional ITS system consists of four core elements: domain model, student model, tutoring model and learning environment (user interface). Existing research on ITS systems can be classified by the aspects of the system that it specialises in. For example, project focusing on domain model realism and intelligence can offer some innovative approach to generating and/or validating problems for students to solve, but may not use best practices in terms of assisting students at learning on providing realistic interaction experience with the system. Similarly, work exploring new ways to provide useful tutoring to students may not necessarily get into intricacies of domain simulation or UI.

Alternative way for classifying ITS projects is by underlying algorithm. One well-known algorithm category that current project can also be attributed to is “model-tracing tutoring”. At the core of this category of algorithms lies the idea of tracking student actions and ensuring that they remain within the boundaries of acceptable solution path.
Common challenge that befalls on all ITS projects regardless of their area of specialization is domain-independence, which is the ability to reuse advances in domain, student or teaching models in other areas of application. While it may appear that domain independence is an inherent quality of intelligence contained within the system models, many experts believe that some of the most essential pedagogical knowledge leading to most effective tutoring strategies is domain dependent. Just think about skills necessary to teach language vs. maths of physics.

2.2 VR Systems for Procedural Training

Virtual reality systems can be especially useful to train users for performing procedures in real scenarios, i.e., sequences of actions that permits to solve a problem or reach a goal. In this context of use, virtual reality systems are used for a wide range of training scenarios, including such critical spheres as medicine and police/military forces. Each of the scenarios comes with its own requirements for the VR component. While it is crucial for medical applications to have high-precision tracking of wide range of hand movements, police force training systems may give more importance to open world-like simulations and freedom of movement.

Several reasons behind developing Intelligent Virtual Environments for Training (IVETs) are:

- Minimizing risks related to performing training in real world
- Minimizing cost of training by using virtual/reusable objects
- Speeding up training for time-consuming domains. For example, agriculture where natural cycle of maintaining a field of crops can take several months

In an attempt to reduce effort needed to produce IVETs for new domains, a software platform called MAEVIF (Multiagent Extension for Virtual Reality Based Intelligent Tutoring Systems) was created[3]. MAEVIF was designed to be easily configurable for wide range of learning applications. MAEVIF architecture was developed following an agent-based approach where each of the components of classical ITS architecture was implemented using one or more software agents.
Figure 1 - MAEVIF's ITS extension
2.3 Reactive Tutor

Reactive Tutor is a type of automatic tutor that was originally implemented to cater the needs of a biotechnology virtual laboratory, which was created in order for students to practice gene encoding a protein responsible for protecting vegetation against diseases.

Biotechnology virtual lab allowed students to roam multiple areas of the building and interact with objects as well as other students in virtual environment. While in the lab, students have an opportunity to carry out several experiments, which would trigger start of the tutoring session. As students go through the various phases and steps of experiment, Reactive Tutor supports learning process with confirmation messages, hints for next possible actions and error messages when students make a mistake. Reactive Tutor can be also configured (domain configuration file) to allow students to make some mistakes and continue with the experiment in order to give them chance to correct their actions or be notified about error later on.

Reactive Tutor also maintains a student model consisting of the sequence of actions performed earlier, which allows tutor to perform types of validations like dependency and incompatibility checks. Depending on the configuration and student actions, validation may produce one of the following results:

- Confirmation (“OK” in figure below)
- Blocking error (“FAIL”), meaning that action that student attempts to do is invalid and must be prevented. For example, this could be any action that would lead to catastrophic consequences immediately, or it could be an action that results in a trivial mistake that lab professors are not interested in.
- Non-blocking error (“FAIL_BUT_GO”), meaning that action that student attempts to do is invalid, but should be allowed for pedagogical reasons (for example, to give student a chance to correct it later).
Figure 2 - Sample scenario of a biotechnology virtual laboratory experiment

Since being created for first edition of biotechnology virtual laboratory based on of OpenSim virtual world platform, Reactive Tutor has been rewritten in C# to work with Unity applications. It is an important foundational block for this master work as it implements all of the ITS core components (domain, student and tutoring modules) on top of which this master work is based.
3 SIEMA: Virtual Reality Training of Car Seat Assembly Workers

SIEMA stands for “Sistemas Inteligentes para ayuda al Entrenamiento en Montaje manual de Asientos de automóviles”, literally meaning “intelligent training systems for manual assembly of car seats”. Request for proposal for this project has originated in a company that, throughout this report will be called SeatInc, specializing in car seat assembly for a number of car manufacturers. The problem that SeatInc is currently facing is manifold:

- High employee turnover with new employees having to go through a training period, before target productivity and quality can be reached.
- Upcoming change in production catalogue requiring retraining of existing staff.
- The existing training process has a bottleneck when it comes to practical exercises due to limited availability of equipment that can be used for training and the need for a skilled trainer to be present.
- The time that is wasted disassembling the seats that are assembled during practical exercises.

Considering the above points, SeatInc is looking to improve efficiency of their current training program in order to be able to train new and existing workers quicker and to a higher standard as well as to potentially modify their hiring process to identify early on whether a candidate is suitable for a role or not.

A virtual reality-based training tool was proposed in order to address the issues currently faced by SeatInc.
3.1 Objectives

Due to the very specific context of the project and lack of domain knowledge in the team, the definition of the project objectives has not been finalised until after the first phase of the project (Analysis of the context of use) has been concluded. After visiting the factory and carrying out several workshops with the client, the following objectives for the desired system have been identified:

- Improve student knowledge of the domain vocabulary prior to commencing supervised practice in the assembly line.
- Improve student knowledge of the assembly processes and sequences of steps prior to commencing supervised practice in the assembly line.
- Increase the number of students that can go through procedural training simultaneously, avoiding the situation where one student does really practice, and the rest just observe.
- Increase the effective time that each student can spend practicing the assembly process.
- Reduce the number of candidates that are rejected after attempting practical exercises.
- Reduce the time and effort required to re-train existing employees using new processes.
3.2 Description of the Generic Process for Manufacturing a Car Seat

Work process at SeatInc is arranged in assembly lines. Each assembly line is equipped for producing predetermined seat types. Assembly line consists of one or two slowly moving conveyor belts with clearly marked work stations throughout the length of the belt.

![Figure 3 - One of SeatInc assembly lines](image)

Each of the work stations is manned by a worker responsible for performing a particular manufacturing step. The car seat assembly happens incrementally. At first, the seat base and back rest are assembled in parallel by layering elements on top of frames. Later on, two fragments are merged onto single conveyor belt and assembly continues in linear fashion.

As most of the components are delivered to the factory as finished products, SeatInc workers mostly have to perform actions to pick up/drop objects or connect components together by clipping, screwing or stapling them together. These are the kind of actions that will have to be modelled in the virtual environment by means of VR devices such as headsets, data gloves, motion-tracked controllers, etc.
3.3 System Architecture

Considering that objectives for this project include improvements to the procedural training of the staff, a system that includes Reactive Tutor was proposed since Reactive Tutor has demonstrated its capabilities in the procedural training context (see section 2.3).

In order to improve system throughput (increase potential number of students practicing simultaneously) and reduce overhead of operating test assembly line a VR learning environment was chosen in favour of augmented reality alternatives. This allows for potentially unlimited number of VR stations to be setup for training both recruits and current employees that are being re-trained.

Technological realisation of proposed system allows for a number of variations. System can be designed to offer an immersive VR experience using VR goggles or a more conservative implementation using classical computer screen. Interaction with VR environment can also be implemented using various techniques depending on the desired range of supported actions and their accuracy. Some of the prominent options are:

- Optical tracking of user hands and body,
- Controller-based interaction with actions being bound to controller movement and button clicks and
- Hand and finger tracking via data gloves.

Finally, a number of graphical engines exist that support rapid creation of VR applications. These engines can be categories by the target platform that they support:

- Desktop
- Mobile
- Web
- Cross-platform
Unity3D is a popular cross-platform graphical engine that allows same source code to be compiled to run on a number of target platforms. It is also widely supported by VR hardware manufacturers with a lot of demos and tutorials available online.

Based on the above objectives, and considering available technology, the following VR training system architectures have been proposed.

### 3.3.1 Component and connector view

#### 3.3.1.1 Simulator instance per student

![Figure 4 - System architecture - Simulator instance per student](image)

The current system architecture assumes an individual isolated work environment for each of the students. The work environment shall comprise a Simulator instance, a set of VR goggles and controllers to interact with the VR environment and a supervisor interface through which student progress and performance inside VR can be monitored. After a lot of research and comparison across various VR hardware options available on the market performed by the SIEMA team members, HTC Vive VR goggles and Manus VR tracking gloves have been chosen for this project.
3.3.1.2 Shared simulator instance

The current approach attempts to optimize the resources needed to run a Simulator instance and proposes that all students are connected to a single Simulator instance but are working in separate but identical spaces inside the VR (“virtual rooms”).

3.3.1.3 Comparison

The shared Simulator instance approach appears to be preferable from the economical point of view and it is known to be used in many virtual worlds applications successfully. However, in the case of the most virtual worlds applications, students are using PC screens and browser clients to interface with the virtual environment. It is not immediately clear what limitations using VR goggles and controllers to interact with the virtual environment will imply. At the time of writing this document, the system has not been developed to the point where differences between the two variants can be evaluated more closely yet, so unfortunately this remains an open question for now.

3.3.2 Top modular decomposition of the simulator

The following diagram in Figure 6 looks closer at the inner structure of the Simulator component using the “Display-Logic-State” three layered approach. Workstation scripts refer to a collection of scripts implementing settings and behaviour of various workstations that the process of assembling car seat comprises from. The settings screen is seen as a point of entry to the system where current simulation parameters (simulated workstation and participating student) are selected. Tutor is the main logic module that analyses student actions and provides tutoring feedback when necessary.
Tutor Configuration is a collection of domain and utility configuration files necessary to run Tutor module. Student Ontology Model is a file-based storage for all student-related information that is gathered by the system.
Figure 6 - Top modular decomposition of the Simulator
4 Software Engineering Practices for a Virtual Reality project

This section will go over some of the key software development aspects, describe them briefly and document how they were applied in the context of the SIEMA project.

4.1 Software Lifecycle

4.1.1 Software lifecycle state of the art

Software life cycle, otherwise known as Systems Development Life Cycle (SDLC), is a term used to describe a process for planning, building, testing and releasing a software system. Three major types of lifecycles can be identified:

- Waterfall model – the oldest and most straightforward SDLC model. The project plan contains a number of phases, each with its own detailed sub-plan. Each of the phases is executed sequentially, with outputs from the previous phase becoming a key input for the following phase. Its simplicity and intuitiveness come at the cost of high risk of failure since a fault made at an earlier phase becomes progressively more expensive to fix during later phases.

- Iterative model – the type of models that focus on repetitive actions. One of the main goals is to produce working versions of software quickly, thus repeating core project phases on the smaller scale each time. It has many advantages, among which are potential for a better feedback flow from the client and ability to “adjust the course” more easily due to small, rapid, self-contained versions. Subtle differences exist between iterative and incremental models, but for the purpose of this document they will be considered as parts of a common category of life cycles.

- Big Bang Model – a high-risk model that focuses most of the efforts on the development phase. Due to an almost total absence of requirements and design phases, this model works best for small projects.
In practice, software development teams follow one of the existing software development methodologies that, among other things, imply the use of a certain SDLC, an extension or variation of one.

4.1.2 Application in SIEMA project

First of all, it is worth highlighting that the scope of SIEMA project can effectively be split in 2 high-level parts – virtual reality and Intelligent Tutoring System components. Due to the very little overlap between the two in terms of the software interfaces and human interaction (two dedicated teams have worked on those components independently), it is appropriate to consider separate software development lifecycles for each of the components.

4.1.2.1 Virtual reality component software development lifecycle

Analysis and design of virtual reality component has proven to be difficult since the beginning. Inability to carry out requirements gathering workshop with client in person has been the bottleneck of this project during early stage. In addition, once the requirement gathering workshop took place, the news broke that the client wishes to develop the VR training system for new work process that has not been implemented yet. That meant that only requirements gathered via interviews were pertinent to new work process while all observational activities carried out where potentially irrelevant as they were done in the context of currently existing work processes. Finally, uncertainty with the choice of hardware for the VR domain prevented system architecture from being completed with satisfactory degree of confidence.

Due to all of the issues mentioned above, it was decided to follow an iterative software development lifecycle model. Its ability to produce meaningful results after each of the iterations and relative ease with which changes can be introduced were considered the best way to counter project risks related with uncertainty. Iterations were defined using requirements for the new work process where separate process phases acted as milestones. Domain knowledge, gained by observing existing work
process and capturing relevant details using photo and video, was used to design virtual environment and define and implement a set of interactions relevant to each of the phases of the project.

Since no particular software development methodology was used, iterations did not have a fixed duration or complexity. At the end of each of the iterations, an internal demo was performed to share the knowledge among the team members, brainstorm problems that were identified and gather peer feedback.

Due to delays related to VR hardware (comparison, choice and procurement), early iterations were developed using traditional screen as an output and mouse and keyboard pair as input devices. For this reason, client demos were postponed until VR hardware platform is chosen and integrated with already implemented process phases. Decision was taken despite slower client feedback loop and increasing risk of important errors/omissions going unnoticed in order to ensure that client demos are as close as possible to final product (most importantly in terms of interface and usability).

4.1.2.2 Intelligent tutoring system software development lifecycle

Intelligent tutoring system component for SIEMA project consisted of adaptation and extension of existing tutoring system (also known as Generic Tutor or Reactive Tutor). All of the work for this component was expected to be performed by a single person in isolation from other project components. A major extension of reactive tutor to make tutoring system more intelligent was based on a doctoral dissertation by another UPM student Diego Riofrío [1] that proposed a design and logic for a tutoring strategy based on a collective student model, as well as a way to combine two or more tutoring strategies in a single Intelligent Tutoring System, thus allowing for future extension of the system.

Considering the well-presented use case, requirements and system design of Intelligent Tutoring System, as well as existing implementation of the collective student model, there was little need for a sophisticated software development
methodology. Kanban-like methodology was followed that best fits into iterative (in this case incremental) SDLC.

As part of the chosen development methodology, increments of functionality were identified and managed as “To-Do” list. This list was frequently revised and updated with new items as the development continued. Closer to the implementation deadline dictated by the scope of this master thesis the need for prioritization was identified which was satisfied using simple ordering of the remaining work items.

Other software development methodologies were considered (such as SCRUM and TDD) but were ultimately rejected mainly due to the high process maintenance overhead required to run those methodologies by a single person. Other reason they were not used is their innate focus on teams and team work – a lot of those benefits are lost when working alone.

4.2 Continuous Integration

4.2.1 Continuous integration state of the art

Term “Continuous Integration” in the context of software engineering is used to refer to the practice of frequent (at minimum daily) merge of work done by all developers. Concept of continuous integration was initially adopted and advocated by Extreme Programming software development methodology [4], resulting in it being adopted by other software development methodologies as well. Main advantage of continuous integration is reduction in time between introducing a fault and it being revealed. As a side effect, integrating continuously saves development teams the trouble of merging large amounts of changes in one go which can be a tedious, time-consuming and error-prone activity on its own.

Following list captures some of the best practices that should be followed in order to achieve continuous integration:

- Maintain code repository
• Automate the build
• Keep features small
• Commit often
• Every commit should be built
• Keep the build fast
• Make build results public

A number of tools exist to assist developers with adopting continuous integration. Some of the most notable features offered by these tools are:

• Ability to configure origin of the source code to be built and tested
• Ability to integrate with popular version control systems in order to automate continuous integration for each of the commits
• Ability to automate the build of the source code
• Ability to automate execution of various tests
• Ability to parallelize test execution to reduce overall build time

Available tools can be split in two categories: on-premises and hosted(cloud), with most notable being:

• Jenkins (on-premises)
• TeamCity (on-premises & hosted)
• Bamboo (on-premises)
• CircleCI (hosted)
• Travis CI (on-premises & hosted)

### 4.2.2 Continuous Integration in SIEMA project

Key requirements for continuous integration tool for SIEMA project were ease of use and maintenance since it had to be set up and maintained by a single person. It was considered to be a bonus if CI tool supported Docker to enable its use for environment configuration purposes and enforce standardisation.

Considering above mentioned requirements for the tool and previous personal experience, it was decided to use CircleCI hosted solution in order to run CI for two
parts of this project: Unity3D code base and standalone tutor package. On-premise-only options were not considered completely due to the additional infrastructure demands that were identified as potential bottlenecks at the early stages of the project. Out of the remaining hosted CI tools, CircleCI has most inclusive free plan which allowed to use it for 2 parts of the project, both being hosted as private repositories.

For Unity3D code base, CI was configured to perform code style checks and run unit tests. In case of tutor package, CI was used to build and run automated tests only. Code style checks were not performed on tutor package code base due to high volume of legacy code that did not comply with any particular code style and required substantial amount of effort to be refactored and standartised.

### 4.3 Configuration Management

#### 4.3.1 Configuration management state of the art

Configuration management in its purest, most generic form is a system engineering process for ensuring that all of product attributes are consistent with the requirements and design throughout its life[5]. In the context of IT service management, configuration management is the process that tracks all of individual configuration items in an IT system, ranging from something as simple as a server up to complex structures of numerous departments and divisions[6].

Fundamental component of configuration management is configuration management database (CMDB) that contains information about individual configuration items. Information in CMDB is used for CM planning, control, monitoring and verification. In addition, CM identifies a number of procedures that must be defined for each software project to ensure soundness of CM process:

- Configuration identification
- Configuration control
- Configuration status accounting
- Configuration audits
One particular configuration item that was of most interest for SIEMA project was Operating System and Application. There are a number of tools out there aimed at helping developers and system administrators manage configuration of the operating system and 3rd party applications involved. Some of the most popular of them are: Ansible, Chef and Puppet. They use Infrastructure as Code (IaC) to automate and standardize configuration management.

A new class of tools that use IaC has been growing rapidly in the last 5-10 years. These are the tools that combine IaC with virtualization platform – Vagrant, Docker, Kubernetes. Although, configuration management is just a little part of what they are capable of, consistency and reproducibility of working environment is still one of the key selling points of these tools.

### 4.3.2 Configuration management in SIEMA project

Due to relatively small project size and lack of dedicated and experienced people capable of setting up and maintaining Configuration Management process in accordance with best practices, no global configuration management that could cover all areas of project lifecycle (from requirements to testing) was used. However, since potential difficulties due to different operating systems used by team members were identified at the early stages of the project, a virtualization platform capable of IaC (Docker) was used to automate and standardize the process of building the project and running style and unit tests for all developers, regardless of the operating system they were running on.

Once necessary tools have been identified, a Dockerfile was created that defined a container image capable of building project and running necessary tests. Dockerfile was then uploaded to Docker Hub, ensuring that built and cached image of this container is always readily available over the network. Resulting image was also used during CI configuration to ensure that CI server running all automated checks was consistent with the development environment.
4.4 Testing

4.4.1 Testing state of the art

In software engineering, testing is one of the most common ways to ensure the correctness of the produced software with respect to the specification. There exist a number of testing techniques with key difference being the abstraction level at which testing is performed: unit testing, integration testing, system testing, etc.

Of them all, unit testing is probably being talked about the most, it being the first type of tests written for any project. Unit testing is a well established practice in software engineering these days with well-defined guidelines on how to approach writing unit tests and how to produce software design that is unit-testable. Unit testing has also inspired software experts to come up with entire software development lifecycles centers around unit testing (and other higher-level testing techniques) – Test Driven Development (TDD).

4.4.2 Testing in SIEMA project

As expected, testing in SIEMA project was mainly focused at the unit testing level. Although tools used for testing two parts of the SIEMA project were different, approach taken was mostly similar.

4.4.2.1 Testing tutor component

Testing tutor component of the system was performed in 3 stages:

- Regression testing of tutor domain configuration
- Regression testing of Generic Tutor
- Unit testing of new extensions to Collective Student Model

First 2 stages were performed half-way through the project, once it became clear that some significant changes will have to be made to existing code base for performance reasons and in order to follow software design best practices. As such, unit tests were created to cover some of the key parts of the existing Generic Tutor (domain configuration & simple student actions). Later on, once necessary changes and
enhancements have been completed, unit tests were created to cover new parts of the code.

NUnit framework for creating unit tests for .Net Framework was used in this project. Unit tests were executed from JetBrains Rider IDE during development and from the CI environment upon every new commit afterwards.

4.4.2.2 Line coverage

Tools for gathering line coverage data were explored in order to obtain some metrics about effort made and effort remaining, but it was discovered that no straightforward way exists to gather line coverage data for .Net Framework applications outside of Windows platform. Microsoft Visual Studio comes with built-in support for code coverage analysis, however Microsoft Unit Testing Framework must be used which is only available for MS Visual Studio for Windows and is not capable of running tests using Mono runtime.

A number of official (MonoCov) and 3rd party line coverage analysis tools exist that claim support of Mono runtime, but are all either target Mono on Windows, or not actively developed. Not to mention lack of integration with modern C# IDEs like JetBrains Rider.

4.5 Code Style

4.5.1 Code style state of the art

Code style (style guide, programming style) is a set of rules or guidelines used when writing the source code of a computer program. Areas of source code covered by style guides include indentation, alignment, spacing, naming and many more. Although, there are claims that following a common style guide in the team improves code maintainability and readability, there is no doubt style guides improve team productivity in one particular way – no more arguments about how the code should look (tabs vs. spaces, camel case vs. Pascal case vs. snake case and other “holy wars” of software engineering).
Some programming languages and frameworks come with their own style guides, others leave the decision up to the developers. Two types of tools exist to assist developers with following style guide:

- Checkers – tools that verify that source code complies with given style guide and report detected violations
- Formatters – tools that in addition to checking for style guide violations are able to auto-format the code to match chosen style guide

### 4.5.2 Code style in SIEMA project

C# programming language, used to modify and extend ITS as well as script Unity scenes, has a well-defined style guide. In addition, some of the most popular IDEs for C# come with a built-in support for either code style checking or formatting.

Challenge for SIEMA project was to find a code style checker that can be executed as a standalone application (being able to plug it into IDE was consider a bonus) and that was cross-platform. In other words, SIEMA project needed a code style checker that supported Mono runtime and could be at least executed from a Linux command line in order to be incorporated into Continuous Integration.

A number of style checkers that support C# have been considered:

- Gendarme – code inspection tool for .Net applications and libraries that uses rule-based configuration to perform style, security and performance audit of the code.
- Resharper – JetBrains plugin for Visual Studio capable of code analysis, refactoring, code formatting, generation and cleanup.
- StyleCop – open source static code analysis tool from Microsoft that check C# code for conformance with a combination of StyleCop and .Net Framework Design Guidelines recommended styles.

Gendarme was rejected due to its dependency on compiled DLL or EXE in order to perform analysis. Resharper was the preferred tool but could not be used due to lack of support for Mono runtime. In the end, StyleCop in a combination with 3rd party utility StyleCopCLI[7] was used to execute style checks as part of the CI workflow.
It is important to notice that style checks were only performed for the Unity scripts due to large pre-existing code base of Reactive Tutor and Collective Student Model that require significant amount of effort in order to be refactored to match any single style guidelines.

### 4.6 Team Management

Team management is very broad discipline and depends on a lot of factors: team size, project size, team location, software lifecycle followed, etc. This section will only describe team management techniques used in SIEMA project without claiming their superiority over other approaches.

Although, most of the time all team members were physically in the same location, an instant messaging app Slack was used to facilitate communication. In addition, Slack was configured to notify development team members about new commits and build status updates for Slack to act as a main notification hub.

Although provisions were made, due to small team size with each developer working independently without code overlap, features like pull requests and peer reviews could not be leveraged.
5 An Intelligent Tutoring System for SIEMA

5.1 ITS Architecture and Current State of the Project

The Intelligent Tutoring System (ITS) implemented as part of this master work was defined and documented in Diego Riofrío’s PhD. thesis “Propuesta de un Modelo de Comportamiento Colectivo de Estudiantes para un Sistema Inteligente de Tutoría dirigido al Entrenamiento Procedimental”. The main focus of his thesis was the design of a collective student model and a tutoring strategy based on this model. This strategy exploited similarities between actions performed by groups of students in order to dynamically identify to which of the groups do students belong and provide tailored tutoring based on the behaviour previously observed among that group.

In addition to the collective student model and the tutoring strategy based on it, this PhD. thesis proposes a tutoring approach that combines multiple tutoring strategies into a single system capable of switching between various tutoring strategies based on which one is deemed more beneficial at any given point in time.

![ITS architecture diagram]

Figure 7 - ITS architecture

Figure 7 illustrates the high-level ITS architecture that consists of two main types of components – tutoring components implementing particular tutoring strategy and coordinator implementing strategy for switching between tutoring components. Such
architecture offers maximum flexibility in terms of adding and removing new tutoring strategies and modifying the logic for switching between them (meta-tutoring).

Next, we will describe briefly the Reactive Tutor, the Student Collective Model and the Tutoring Coordinator to introduce the theoretical foundations of the software that has been modified or extended in this master thesis:

5.1.1 Reactive tutor

Reactive Tutor is responsible of providing the generic tutoring. It is implemented in its second edition and used by UPM’s GATE department to offer virtual laboratory training to students of a number of degrees. The first edition (as mentioned in section 2.3) was used in conjunction with OpenSim virtual world platform and implemented using one of scripting languages supported by OpenSim. The second (current [8]) edition of the Reactive Tutor is implemented as a collection of C# DLLs that allow it to be used with modern platforms for virtual environment such as Unity3D. Other major change included in 2nd edition of Reactive Tutor is the use of an ontology for defining the student model (background, logs, etc.). This addition has paved the road towards implementation of Individual Student Model [3] in the future.

Existing Reactive Tutor implementation structure follows closely the classical ITS architecture extended with a World Module (see Figure 8):
Here, Student Module is responsible for maintaining information about students that are in the system, creating new ones if necessary and also managing student log input/output activities. Student Module is also the place where student ontology model currently resides. Expert Module represents a domain-independent expert system, capable of loading domain and domain action configuration from an Excel spreadsheet and validating student actions in accordance with this configuration. World module is tasked with keeping track of objects referenced by a domain configuration in order to be able to determine situations when students attempt to use the objects that they can’t (either because it’s not ready or in use by someone else). Communication Module was envisaged as a bridge between C# based 2nd generation Reactive Tutor and legacy OpenSim project used for visualisation. As the visualisation was re-implemented in Unity, it became possible for it to interact directly with Reactive Tutor module via exposed API.

5.1.2 Collective student model

Collective Student Model has been partially implemented by Diego Riofrío in order to be able to analyse performance of various clustering algorithms used to assign students to groups. As such, the implementation focused on some key aspects of the model such as processing historical student logs, student clustering and creation of action automaton to represent sequence of actions taken by students of each of the clusters.
In terms of structure, Collective Student Model has two main components: student clustering and action automaton.

![Collective Student Model structure](image)

Figure 9 - Collective Student Model structure

Clustering component applies clustering algorithm implementations supplied by Weka to some of the key attributes of student action logs, such as total practice time, number of correct actions, number of errors and split of error states across relevant and irrelevant zones. As a result, all available students are assigned to one of the clusters along with other students with similar performance.

Initially, Collective Student Model is built at tutor launch time, when all available student logs are aggregated, clustered and combined into cluster automatons. Afterwards, model is kept up to date throughout the training by adding new student actions to the cluster automaton in real time.

Overall state of completeness of Collective Student Model prior to start of this master work can be expressed as 80% with remaining 20% requiring work on adapting interfaces to work with real-time student information as well as algorithms and data structures necessary to provide tailored tutoring messages.
5.1.2.1 Student action automaton

Once students are grouped in clusters using one of the available clustering methods, the aggregate student action automaton is produced by combining action logs of all students within the same cluster.

Automaton nodes represent practice states after performing some domain action and arches represent different types of events provoked by the students such as attempts of wrong actions, performed actions or errors associated to wrong actions. Since automaton can be considered a directed graph, it was convenient to use it to store and retrieve information about various event and state frequencies within the context of particular cluster.

It is also worth noting that all graph nodes were divided in three zones: correct flow zone, irrelevant errors zone and relevant errors zone (see Figure 10):

- Correct flow zone corresponds to states that are in accordance with domain action configuration and hence represent states after valid actions were performed.
- Relevant errors zone corresponds to states of practice after certain types of invalid actions have been performed. Namely, relevant are considered errors that have high academic value – dependency and incompatibility check failures, timeouts, etc. In order for a state to be assigned to relevant error zone, not only it has to be of one of the above-mentioned types, but it has to also be non-blocking, i.e. the error that was detected by student was allowed to continue with the action in academic purposes.
• Irrelevant errors zone mostly contains states that corresponds to students performing insignificant mistakes, such as attempting to execute same action twice or attempting to execute action that is not part of the domain configuration at all (may be a sign of technical problem). Other error types can also result in irrelevant error zone state if they are blocking errors, i.e. tutor is configured to detect and block actions, thus preventing student from diverging from correct execution path.

5.1.3 Tutoring coordinator

At the core of Tutoring Coordinator defined in Diego Riofrío’s thesis was the idea of layered approach: stacking available tutoring strategies starting from the most granular and precise (Individual Student Model based on student knowledge model) all the way to generic automatic tutor (Reactive tutor). Whenever tutoring feedback is required, first the strategy considered to be the most effective is attempted. If the strategy is deemed not to have sufficient support or otherwise not applicable at the moment, tutoring coordinator moved down a layer and attempts to apply next tutoring strategy.

Tutoring coordinator was only defined as a high-level architectural component with algorithms for identifying most suitable tutoring strategy defined using pseudocode. As such, this software component had to be designed and implemented from scratch in this master thesis, and in this process was incorporated the other tutoring components.
5.2 Problems Found

5.2.1 Problems in the existing reactive tutor design and implementation

5.2.1.1 Software practices

Although VCS was used during implementation of Collective Student Model, some of the VCS best practices were not followed, resulting in performance issues. Biggest issue was lack of use of .gitignore file to exclude undesired files such as compiled binaries and logs produced by tutor from Git repository. As a result, repository size was significantly inflated, slowing down the checkout time, which became crucial with the introduction of Continuous Integration.

Another aspect of software engineering that was somewhat abused was code comments. This topic is part of the software style guidelines and as such, there are many works out there advocating various approaches to code commenting. However, general consensus on the answer to the question “How many comments should there be in the code?” is expressed well by the phrase “as little as possible, as much as necessary”, with focus on explaining not what but why certain things are done in the code. Unfortunately, in some cases these best practices were not followed, which has inhibited code reading and understanding as well as left some crucial questions about design choices and logic unanswered.

Finally, considering fairly large source code base of Reactive Tutor and Collective Student Model, none of the components used any kind of automated testing. This fact in addition to lack of documentation often created uncertainty about certain aspect of the code being “a bug or a feature”, since it wasn’t immediately obvious whether particular behaviour was intended or not. Also, since automated testing often acts as regression testing during the maintenance phase, not having unit tests in the first place has slowed down significantly making changes to the existing code due to extra effort required to ensure that external system behaviour remained unchanged.
5.2.1.2 Dependency on single configuration file library

The way Reactive Tutor was implemented, it had dependency on a built-in system library for working with configuration files. The library defines conventions for configuration file naming and location. When executed as part of a standalone .Net console application, it is possible to satisfy those conventions. However, since Reactive Tutor will be used inside Unity3D runtime environment, there are limitations imposed by Unity3D framework on the locations where files can reside. As such, using this particular library has proven to be impossible inside Unity3D project.

5.2.1.3 .Net framework version incompatibility

Initially, Unity3D project was being run using stable Mono version which at the time of writing corresponded to .Net v3.5. However, all of Reactive Tutor modules were implemented using .Net v4.6. As such, attempting to load Reactive Tutor DLLs into Unity3D project resulted in version incompatibility errors.

5.2.1.4 Singleton controller classes

Second edition of Reactive Tutor codebase consisted of 2 major sub-parts [8] – Reactive Tutor logic for returning tutoring feedback and generic tutoring framework that provides access to functionality such as:

- Domain configuration (including domain actions)
- Student module
- Logging
- Validation

Code was organised in separate modules (sub-libraries) with varying degree of decoupling. Majority of modules had a top-level controller class responsible for providing interface to module functionality. Unfortunately, these controller classes were implemented using singleton pattern. This wasn’t a problem when only Generic Tutor existed and neither it was a problem for Tutoring Coordinator since all instances of various tutoring strategy implementations (generic, collective, etc.) share tutoring framework classes. However, Singleton classes became an issue when multiple domains were attempted to be loaded sequentially. Since different domains come with different configurations, file location and logs, use of singleton pattern resulted in corrupted state. This was not an issue with 1st edition of Reactive Tutor
since it had fully implemented support for domain stages – a way to group actions within a domain configuration. This approach could have been used in SIEMA project to model actions pertinent to various workstations, but during the implementation of 2nd edition of Reactive Tutor support for domain stages has not been implemented correctly and as such is not currently available. Coincidently, domain configurations used with 2nd edition of Reactive Tutor before SIEMA project did not depend on domain stages features, which contributed to the problem not being discovered earlier.

**5.2.1.5 Tutor message classification**

The interface of the Generic Tutor had provisions for two main outputs:

- validation result, stating whether performed action is valid or not
- tutor messages

Tutor messages were returned as a list of strings, potentially containing a mixture of message types, such as confirmation messages, messages providing hints for next actions and error messages. It was expected that with addition of new tutoring strategies, new types of messages will emerge (such as error prevention messages of Collective Model Tutor).

One of the requirements of SIEMA project was to differentiate between experienced students and complete beginners and provide different degree of tutoring to each. While beginners may find it useful to hear confirmation messages once correct actions have been performed, it may be not necessary (and even disturbing) for experienced students. Unfortunately, with the initial design of the Generic Tutor interface there was no way to further filter tutoring feedback based on the message type due to the use of plain strings for passing the messages.

**5.2.1.6 Action validation**

Action validation refers to the part of generic tutoring framework that is responsible for computing return code for each of the student actions. As the input, this part of framework receives details of the domain to determine the context, student to determine what previous actions have been performed, action that is being performed currently and object that action is applied to. As an output, it provides an integer in the set {0, 1, -1}, signalling whether action is valid("1"); invalid and should be prevented("0"); or invalid but student can continue("-1"). In addition, error messages
can be potentially added to the list of tutor messages to explain the nature of issues discovered during action validation.

After working closely with 2\textsuperscript{nd} edition of Generic Tutor and studying the internals and usage of its validation module, it became clear that there are a number of design problems and bugs present in it. The three problems that had highest impact on the project progress are:

- Duplicate code
- Absence of error messages in case of non-blocking dependency errors
- Incorrect sequence of Action and Error logs with respect to each other

\textit{Duplicate code}

Main top-level validation function \texttt{ExpertControl.ActionValidation(\ldots)} responsible for initiating a number of checks, processing and aggregating validation results initially consisted of 457 lines of code and comments. Out of those 457 lines, 190 lines (almost half) were repeated twice. To illustrate the problem better, here is the outline of \texttt{ActionValidation(\ldots)} method:

```
1. if (X)
2. {
3.     if (Y)
4.     {
5.         //Do ABC
6.     }
7. } else
8. {
9.     //Do ABC
10. }
```

Only in case of \texttt{ExpertControl.ActionValidation(\ldots)}, “ABC” was 190 lines long. Considering that average screen can only display around 70 lines vertically, this has been a major obstacle for understanding the validation code, analysing its design and spotting any potential issues.

\textit{Absence of error messages in case of non-blocking dependency errors}

This problem was identified while covering new extensions of Collective Model Tutor with unit tests.
Incorrect sequence of Action and Error logs with respect to each other

This problem lies on the verge between validation and logging. It is best described using small example:

Assume domain consisting of 3 actions: A, B, C that can only be performed in that particular order. If action C is performed before action B, then a non-blocking validation error or blocking validation error is raised depending on how the dependency between B and C is configured. In terms of logging, sequence of student actions “A -> C” should produce the following sequence of log entries in the two possible cases:

<table>
<thead>
<tr>
<th>#</th>
<th>Non-blocking error</th>
<th>Blocking error</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Action log: A</td>
<td>Action log: A</td>
</tr>
<tr>
<td>2</td>
<td>Action log: C</td>
<td>Error log: “C before B”</td>
</tr>
<tr>
<td>3</td>
<td>Error log: “C before B”</td>
<td></td>
</tr>
</tbody>
</table>

*Figure 11 - Expected log entry sequence for blocking and non-blocking dependency errors*

This logic was determined in the first edition of Reactive Tutor written for OpenSim and it was expected for second edition of Reactive Tutor to mimic this behaviour. The order of logs is crucial for Collective Student Model. It is worth noting that in one case, “Action log: C” appears before error log, but in the other case it is not present at all.

However, second edition of Reactive Tutor produced following log entry sequence:

<table>
<thead>
<tr>
<th>#</th>
<th>Non-blocking error</th>
<th>Blocking error</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Action log: A</td>
<td>Action log: A</td>
</tr>
<tr>
<td>2</td>
<td>Error log: “C before B”</td>
<td>Error log: “C before B”</td>
</tr>
<tr>
<td>3</td>
<td>Action log: C</td>
<td></td>
</tr>
</tbody>
</table>

*Figure 12 - Actual log entry sequence for blocking and non-blocking dependency errors*

Reason for that was that action logs were created at the very last moment, after all validations were performed. As the result, validation error logs were created and stored before any action log could be created.
Although this bug doesn’t seem like much either, matters were complicated by the fact that creation of log entry resulted in instantaneous write to the log file. So, in order to be able to control sequence and timing on log entries, writing logs had to be decoupled from creating in-memory log entries.

### 5.2.1.7 Unity performance issues & LINQ

After Reactive Tutor was plugged into test Unity scene, it was noticed that after some time, the tutor response time has been increasing significantly. A test domain with 1MB of student log entries was prepared in order to reproduce the issue. While the issue was reliably reproducible in Unity test scene, same behaviour was not observed when Reactive Tutor was executed via .Net console application. This has cast suspicions on C# code performance inside Unity.

After a bit of searching, Microsoft documentation was found containing recommendations for improving Unity performance [10]. The top reason responsible for CPU performance issues was listed as garbage collection. In particular, this document recommends minimising unnecessary memory allocations inside the code. The list of unintuitive reasons why memory allocations may take place was topped by use of LINQ queries.

LINQ (Language-Integrated Query) is .Net framework API for writing SQL-like expressions for querying collections of data. While they are capable of replacing clunky and hard-to-read loops that use auxiliary variables with several lines of clear statements, they come at the cost. Jackson Dunstan [11] wrote a blog post where he performed a micro-benchmark of LINQ queries in Unity and compared their performance to that of traditional for-loops. In short, using LINQ queries in Unity is 30 to 60 times slower than using native looping statements.

After this discovery, Reactive Tutor code was inspected and a number of LINQ queries used to search and select entries from the student logs collection were found. Refactoring those LINQ queries using for-loops has resulted in a Reactive Tutor performance inside Unity comparable to that of a standalone .Net console application.
5.2.1.8 Logging

In addition to logging-related issues mentioned in the previous section, other problems related purely to logging module were identified.

Generic Tutor uses an ontology model for persisting student logs. A 3\textsuperscript{rd} party tool called Apache Jena [12] is used for working with ontology objects and it comes with out-of-the-box mechanism for marshalling and unmarshalling ontology objects using XML files. Since Jena is written in Java, it is interfaced with Generic Tutor C\# code via IKVM.NET [13], which is an implementation of Java for Mono and Microsoft .Net Framework. It contains a Java VM implemented in .Net, collection of Java system class libraries implemented in .Net and a set of tools that enable Java and .Net interoperability.

After analysing the implementation of logging module and performing several simple load tests, following facts were established:

- each call to create new log entry resulted in log entry being written to the disk.
- in order to persist new log entry, the entire student log ontology model has to be marshalled and written to XML file.

Although logging module has been saving ontology to disk after each new log entry is created, the operation of saving ontology to disk is getting progressively slower as the number of log entries in student log increase. In addition, the ontology can be re-saved multiple times per student action (with minor changes), if multiple validation error logs are generated. This way of logging implementation has impacted performance of providing tutor feedback significantly and affected tutor’s ability to provide feedback in real time.

5.2.1.9 Future performance requirements

The reason why it is important to maintain real-time performance is to ensure that the virtual environment that tutoring system is integrated in is:

- Able to respond immediately to student actions considering validation results of the action performed. For example, if action results in an error, VR component may choose to “undo” a change or reset position of some object in
virtual environment. To maintain realism and minimise student confusion, it is better to respond to student actions as quickly as possible.

- Able to immediately provide meaningful explanation or advise based on the action validation results. For example, if action results in an error, then, in addition to visual feedback from the environment, the student should also receive tutoring feedback. Like with visual feedback of VR environment, long delays between action and tutoring feedback are undesirable.

Considering above performance constraints, it was not clear how fast Collective Model Tutor will perform with the load of real student logs accumulated after several years of practices.

5.2.2 Problems in the collective model tutor design and implementation

5.2.2.1 Non-reusable code for updating cluster automaton

As mentioned earlier, Collective Model Tutor was implemented partially by Diego Riofrío. The part that was already available is the actual Collective Student Model, capable of processing collection of student logs and producing clusters of students and graphs depicting action sequences made by all students in the cluster. What it lacked was the actual tutoring component – something capable of processing student input in real time and using Collective Student Model to provide tutoring.

Due to these limitations of initial implementation, not only certain modules were missing, but also the already existing ones were not built to support real-time operations with Collective Student Model. One of the key areas that was not designed with end-use in mind was the part responsible for creating and updating student action automaton. The available code was written on assumption that all student actions necessary to build automaton are accessible via logs and can be loaded in bulk. As such, there was no easy way to add logs entries one by one and refactoring was necessary.

5.2.2.2 Automaton implementation and issues with path finding

In addition to the previous issue, a more fundamental design issue was identified in the Collective Student Model. The cluster automaton is a graph combining sequences
of actions of all students in that cluster. Internally, automaton is represented by a collection of states and transitions. Each of the states and transitions has some related information stored about it, namely number of students that had visited particular state or followed given transition. Although, thanks to clustering, students with some similarities are grouped together, there may be some differences in the sequence of steps that students perform, thus producing different graphs. Two key problems can arise when individual student logs are combined to create a common cluster automaton:

- Paths that none of the students followed are created
- As a particular case of the above, graph cycles can be produced

In this case, it may appear from the combined graph that there were students who performed 2-3-4 and 1-3-5 sequences of steps, although that is not the case.

This case is similar to the previous with the complication of producing graph cycle. Having cycles in the graphs has implications on the algorithms used to work with the graph. For example, special care needs to be taken when implementing search algorithm in the graph to avoid getting stuck in infinite loops due to graph cycles. In
addition, graph cycles add uncertainty to the data model since there is no way of
telling whether a cycle is a direct result of student repeating several actions or if it is
an artefact from merging multiple student actions. Finally, even if the resulting cycle
is a direct result of one student’s actions, current data model doesn’t support capturing
information about how many times cycle has been traversed, making it impossible to
calculate precise graph metrics, such as how many times a node or arch have been
visited.

Above mentioned issues came to light when working on the tutoring aspect of
Collective Model Tutor and implementing algorithms for preventing likely errors at
an early stage. Determining likely errors involved calculations of path confidence and
frequency from student current state to all potential error states. Since combined
cluster graph contained paths that none of the students took, in some situations
support and confidence of the path leading to an error were inflated.

5.2.2.3 Pseudocode errors

In several instances, during the implementation of algorithms used to provide tutoring
based on Collective Student Model, minor errors in the pseudocode documenting
those algorithms have been identified and fixed on the go.
5.3 Design of the Extended Intelligent Tutoring System

5.3.1 Package diagram

The diagram in Figure 15 outlines final package structure of this particular ITS. A lot of similarities can be identified between this diagram and the diagram describing state of the project after 2nd generation Reactive Tutor has been completed (Figure 8 - Package diagram of Reactive Tutor [8]). The main difference is that the Reactive Tutor implemented to be a standalone top-level package is now just a part of a bigger Tutoring Module. Tutoring Module contains Reactive Tutor, tutor based on Collective Student Model as well as Tutoring Coordinator.
5.3.2 Class diagrams

5.3.2.1 Tutoring module

Figure 16 - Tutoring Module Class Diagram
The new Tutoring Module, in addition to absorbing previously existing Reactive Tutor Module, now also hosts Collective Model Tutor, Tutoring Coordinator and Common modules.

Common module contains unified `ITutor` interface and `AbstractTutor` class that took on the responsibility of performing common tutoring framework activities, such as validating student actions, creating students, etc. All specific tutoring strategy classes (`ReactiveTutor`, `CollectiveModelTutor`, `TutorCoordinator`) extend `AbstractTutor` and thus inherit all core tutoring framework functionality. New `ITutor` interface methods (`Validate(…)` and `GetTutorMessages(…)`) can be seen in the class diagram above for the reasons mentioned in sections 5.2.1.5, 5.2.1.6 and 5.2.1.8. Detailed explanation of the reasons behind new interface methods and their responsibilities can be found later in section 5.4.2.

Collective Model Tutor module (CMTutor) hosts a collection of sub-modules responsible for delivering Collective Student Model and tutoring strategy based on this model. Contents of this module will be examined closer in the next section.

*TutorCoordinator* module supplies a same name class that implements tutoring strategy layering approach as mentioned in section 5.1.3.
5.3.2.2 Collective model tutor – closer look

Figure 17 - Collective Model Tutor module class diagram