communicated by:
Lehr- und Forschungsgebiet Informatik 9

Prof Dr. Ulrik Schroeder

The present work was submitted to Learning Technologies Research Group
Diese Arbeit wurde vorgelegt am Lehr- und Forschungsgebiet Informatik 9

Enabling Feedback for Block-Based Programming through the Development of an xAPI Logging Interface for Cubi

Ermöglichung von Feedback für blockbasierte Programmierung durch die Entwicklung einer xAPI-Logging-Schnittstelle für Cubi

Master-Thesis
Masterarbeit

presented by / von

Gušić, Lejla
365549

Prof. Dr. Ulrik Schroeder

Prof. Dr. Jürgen Giesl

Aachen, September 27, 2023

Contents

List of Figures iii

List of Tables v

List of Listings vii

1. Introduction 3

2. Background 5
2.1. Block-based programming languages . . . 5
2.1.1. Benefits and Challenges of BBPLs . . . 5
2.1.2. Comparative Analysis: Block-based vs. Text-based Programming . 5
2.1.3. Prominent Examples of BBPLs . . . 6
2.2. Cubi . . . 6
2.2.1. Cubi's Design and Structure . . . 7
2.2.2. Cubi's Limitations . . . 8
2.3. xAPI . . . 8
2.3.1. Effectiveness of Learning Analytics . . . 9
2.3.2. The xAPI Statements . . . 9
2.3.3. Learning Record Store . . . 10
2.4. Feedback . . . 11
2.4.1. Feedback types . . . 11
2.4.2. Gamification . . . 12

3. Related Work 13
3.1. Logging Interactions . . . 13
3.1.1. Logging in Block-based Programming Environments . . . 14
3.1.2. Logging xAPI statements . . . 15
3.2. Feedback in Programming Learning Environments . . . 16
3.2.1. Existing Feedback Strategies in Block-Based Programming Environments . . . 16
3.2.2. Challenges in Feedback Strategies . . . 16
3.2.3. Enhancing Feedback Mechanisms for Novice Programmers . . . 16
3.2.4. Evaluation of Feedback . . . 19

4. Research Design 21
4.1. Research Questions . . . 21
4.2. Logging Learning Experiences . . . 21
4.2.1. Learning experiences in block-based programming . . . 22
4.2.2. Theoretical logging capabilities . . . 22

4.2.3. Practical logging capabilities . . . 22
4.2.4. Meaningful Logging in Cubi . . . 23
4.2.5. Limitations . . . 25
4.3. Privacy . . . 26
4.4. Standardisation . . . 27
4.4.1. Block-based vocabulary . . . 27
4.5. Feedback Types in Block-Based Programming . . . 31
4.5.1. Existing Feedback Mechanisms in Cubi . . . 31
4.5.2. Enhancing and Expanding Feedback Mechanisms in Cubi . . . 33
4.6. Categorisation of Feedback for Block-Based Programming . . . 36
4.7. Summary . . . 37

5. Implementation **39**
5.1. Used Software . . . 39
5.2. The Logging Components . . . 40
5.3. The Logging Manager . . . 41
5.4. Vocabulary . . . 42
5.5. Summary . . . 42

6. Evaluation **45**
6.1. Setup . . . 45
6.2. Time to Interact . . . 45
6.3. Summary . . . 46

7. Discussion & Future Work **47**

Appendix **48**

A. Bibliography **49**

B. Glossary **55**

List of Figures

1.1. Illustration of a specific case from a Cubi programming course, where participants were challenged to correct their character's stationary behaviour. Left: Incorrect implementation using an incompatible code block resulting in an unchanging x-coordinate. Right: Correct use of the intended code block. 3

2.1. Cubi's user interface (version 2.9.1): an illustrative level with component explanations [24] 7
2.2. Exemplars of diverse block types with varied inputs and constraints; Note that start blocks cannot have parent blocks, and connections are established only when feasible. [24] 8

4.1. Different categories of blocks availabe in Cubi depending on the current level. Left side: One of the first levels with only basic blocks. Right side: A more advanced level with a wider range of blocks. 32
4.2. When the Execute button is pressed, the code is executed and the sprite behaves as programmed. The blocks currently being executed are highlighted in yellow. 32
4.3. Tutorials, located below the stage, explain the task and provide hints. The visually explanatory GIFs help users to understand the required actions. . 33

5.1. A structural overview of the CubiXLI integrated into Cubi. Within Cubi, each component is paired with a corresponding logging component. The central role of the Logging Manager is to facilitate the transfer of statements to the Learning Record Store. 39

6.1. TTI of Cubi with and without CubiXLI 46

List of Tables

2.1. Details of each property of an xAPI statement. Table provided by [58] . . . 10

3.1. Kiesler's analysis of applied feedback types in block-based programming environments [30]. 16

4.1. All possible user interactions in Cubi. 24
4.2. Cubi's xAPI vocabulary 28
4.3. Existing feedback mechanisms in Cubi. 33
4.4. Categorisation of feedback mechanisms in Cubi. 36

5.1. A comprehensive list of all statements logged by the logging components. 40

6.1. Specs of the localhost used for the evaluation 45

List of Listings

2.1. Basic xAPI statement. This statement represents *Shafi completed level 1* . . . 9

4.1. Context excerpt from the xAPI statement: Displaying session ID, platform version, and language settings. 28
4.2. Object segment of the xAPI statement: Specifically, this refers to the *controls_repeat* block within the *loop_block* category. 29
4.3. ContextActivities section of the xAPI statement: Shows the parent block (the *move* block), a child block (a "*math_number*" block) and the current sprite. 29

Abstract

Block-based programming languages (BBPLs) provide an ideal starting point for novice programmers and encourage computational thinking skills. They use visual blocks to represent code constructs, simplifying programming for learners by eliminating the need to memorise and type complex syntax. However, in BBPLs, syntactical errors are rare, shifting the focus to semantic issues, making automated feedback challenging. This thesis centers around Cubi, a BBPL, and introduces an Experience API (xAPI) logging interface to collect data for enhancing user feedback. By adopting the xAPI standard, this work aims to standardise data collection, enabling collaboration across BBPL platforms and paving the way for data-driven, personalised feedback improvements.

Chapter 1 Introduction

Block-based programming languages (BBPLs) provide a valuable first approach for young and novice programmers to acquire computational thinking skills and basic programming concepts. Within the realm of visual programming languages (VPLs), block-based languages have emerged as a distinct and user-friendly branch. One of the main factors contributing to the effectiveness of VPLs is the ease of use of the interface, reducing the burden of using complex syntax, and the promotion of an environment conducive to iterative experimentation [55].
In BBPLs, the reduced likelihood of syntactical errors shifts the focus to semantic issues, where most bugs affect the logic and functionality of a program, rather than its structural integrity. This unique environment presents challenges in providing automated feedback, particularly in scenarios involving open-ended programming tasks [54]. For instance, consider a situation in which an user accidentally selects a visually similar,but contextually different block. In such cases, the BBPL system does not intervene, as the syntax remains correct. However, the system lacks the understanding to accurately discern the user's intent, making it challenging to provide appropriate feedback (see Figure 1.1).
While the presence of an instructor could potentially provide valuable guidance, the feasibility of providing perfectly timed support to each individual student is limited. Studies underscore the critical role of feedback to novice programmers, as users who become stuck and lack assistance are more likely to abandon their programming endeavors [32]. Therefore, the integration of automatically generated elaborative feedback into the BBPL environment seems to be a promising solution. To gain an understanding of user actions, it is imperative to collect traces and logs that document user interactions with BBPLs.

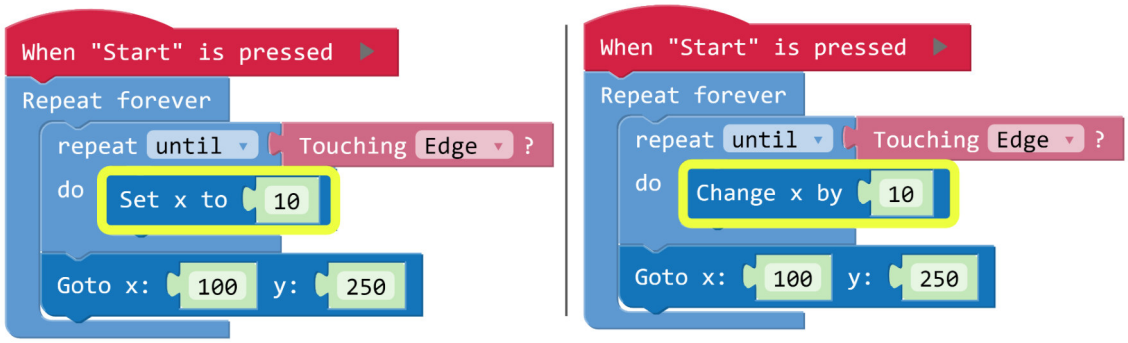


Figure 1.1.: Illustration of a specific case from a Cubi programming course, where participants were challenged to correct their character's stationary behaviour. Left: Incorrect implementation using an incompatible code block resulting in an unchanging x-coordinate. Right: Correct use of the intended code block.

This data-driven approach not only deepens the insight into the learning process, but also holds the potential to provide tailored, elaborate feedback to each user.
This thesis is centered around Cubi[1], a BBPL developed by IT4Kids. Cubi shares its roots with Scratch[2], which serves as a fundamental inspiration for its design and functionality. Similar to Scratch, Cubi uses the Blockly[3] framework.
To provide effective feedback to users, it is crucial to collect high-quality data [3]. In pursuit of this goal, this thesis introduces the implementation of *CubiXLI*, an Experience API (xAPI) logging interface within Cubi. CubiXLI serves the purpose of meticulously tracking students' behaviours as they engage in various tasks, offering valuable insights into the factors that may influence their performance. Such data-driven observations have the potential to identify at-risk students, enabling targeted support interventions. Moreover, the system can intelligently challenge advanced learners to maximise their learning experience.
The importance of using the xAPI standard in the context of this thesis is noteworthy. While there are related studies [13, 7] that address the recording of user progress and outcomes when working with BBPLs, the lack of a standardised framework is a critical challenge. By adopting the xAPI standard, this thesis aims to bridge this gap and establish a common ground for collaboration with other BBPL platforms. This standardisation will enable the comparison of results and findings across different BBPLs, transcending the limitations of individual platforms, and promoting a more comprehensive understanding of user interactions and learning outcomes.
Research has shown the benefits of automated, data-driven, and personalised feedback in improving student learning outcomes [31]. Building upon this foundation, this thesis aims to investigate the current feedback mechanisms within the Cubi programming environment. Furthermore, it explores the possibility of creating new feedback methods, which could improve learning outcomes for users.

[1] `https://it-for-kids.org/` *(Accessed: 15.03.2023)*
[2] `https://scratch.mit.edu/` *(Accessed: 21.08.2023)*
[3] `https://developers.google.com/blockly` *(Accessed: 15.03.2023)*

Chapter 2 Background

This chapter provides foundational context necessary for understanding the research presented in this thesis. It covers key topics such as block-based programming languages, the *Cubi* programming environment, the Experience API (xAPI), and the role of feedback mechanisms within educational settings.

2.1. Block-based programming languages

Visual programming languages definition

Visual programming languages (VPLs) provide an unique approach to programming education through the utilisation of visual representations for code constructs. This thesis will focus on block-based programming languages (BBPLs), a subtype of VPLs.

Utility for young learners

Research by Sun et al. [51] has shown the advantages of (BBPLs). Users of (BBPLs) make fewer syntax errors and produce more concise code in comparison to their text-based counterparts. In addition, BBPL tend to promote greater engagement in debugging activities. Their utility extends to young learners, who often face literacy challenges. BBPLs eliminate the need for manual coding, allowing users to use drag-and-drop techniques with colour-coded blocks for different coding elements. This has the potential to enhance programming efforts by alleviating concerns about grammatical accuracy.

2.1.1. Benefits and Challenges of BBPLs

Challenges commonly encountered when learning programming using text-based programming languages include confusion, frustration, and boredom [28]. Tasks, such as program architecture, function decomposition, and debugging frequently, present significant hurdles. BBPLs mitigate these challenges, providing learners with a more accessible and intuitive entry point into the field of programming.

Weintrop [55] conducted a study on the preferences of high school students, which revealed attractive aspects of block-based programming such as user-friendliness of the environment, natural language descriptions of code blocks, drag-and-drop interactions, trial-and-error support, and provision of visual outcomes. However, challenges were also identified, including a perceived lack of authenticity, limitations in expressive power and difficulties in writing complex programs.

2.1.2. Comparative Analysis: Block-based vs. Text-based Programming

Weintrop's [56] subsequent work focused on a comparative analysis of block-based and text-based programming in an educational context. The results showed that students exposed to block-based programming showed greater learning gains and a higher tendency to pursue future programming courses. Conversely, text-based programmers tended to view their solutions as more similar to those of professional programmers.

2.1.3. Prominent Examples of BBPLs

In educational contexts, several notable BBPLs have garnered recognition. These include platforms such as Scratch,[1] Alice,[2] snap!,[3] and Blockly[4]. However, this thesis will focus on Cubi,[5] which will be discussed in more detail in the following section 2.2.

2.2. Cubi

Introducing programming education at primary schools

Cubi, a block-based programming environment developed by *IT4Kids*,[6] serves as a platform for teaching programming concepts [25]. It has been successfully integrated into teacher-led primary school programming courses in Germany, benefiting 6,500 students across 54 educational institutions [23]. This strategic deployment aligns with Germany's demand for IT specialists, estimated at 137,000, emphasizing the significance of introducing programming education at the primary school level [5].
Research conducted by Stamatios et al. [49] confirms the effectiveness of block-based programming for primary school students. It demonstrates how this approach can nurture computational thinking and coding skills. Cubi emerges as a direct response to the need for coding education in primary schools, drawing inspiration from Scratch.

Cubi's pedagogical approach Cubi distinguishes itself by prioritizing user-friendliness for both educators and students. It offers a graded learning experience with distinct levels, tailored to various programming concepts. This incremental approach enables students to progressively master the available programming blocks. Furthermore, specific blocks are only accessible at particular levels, each complemented by suitable solutions. As of the present, Cubi encompasses a diverse range of 43 levels [24]. It is worth noting that the programming language within Cubi is also referred to as *Cubi*.

Stakeholders Cubi caters to the needs of four distinct user categories:

- **Novice learners**: Positioned as the central and primary users, novice learners actively engage with the platform to learn and apply programming concepts. They don't have little or no previous programming experience.
- **Intermediate learners**: Intermediate learners have a basic understanding of programming concepts and are capable of applying them to solve simple problems.
- **Teachers**: Teachers assume the crucial role of guiding students through their learning journey, providing instruction and support.
- **Administrators**: Administrators are responsible for maintaining the platform and creating new levels to enrich the learning experience.

[1] `https://scratch.mit.edu/` *(Accessed: 21.08.2023)*
[2] `http://www.alice.org/about/` *(Accessed: 23.08.2023)*
[3] `https://snap.berkeley.edu/` *(Accessed: 23.08.2023)*
[4] `https://developers.google.com/blockly` *(Accessed: 15.03.2023)*
[5] `https://editor.it-for-kids.org/` *(Accessed: 21.08.2023)*
[6] `https://it-for-kids.org/` *(Accessed: 15.03.2023)*

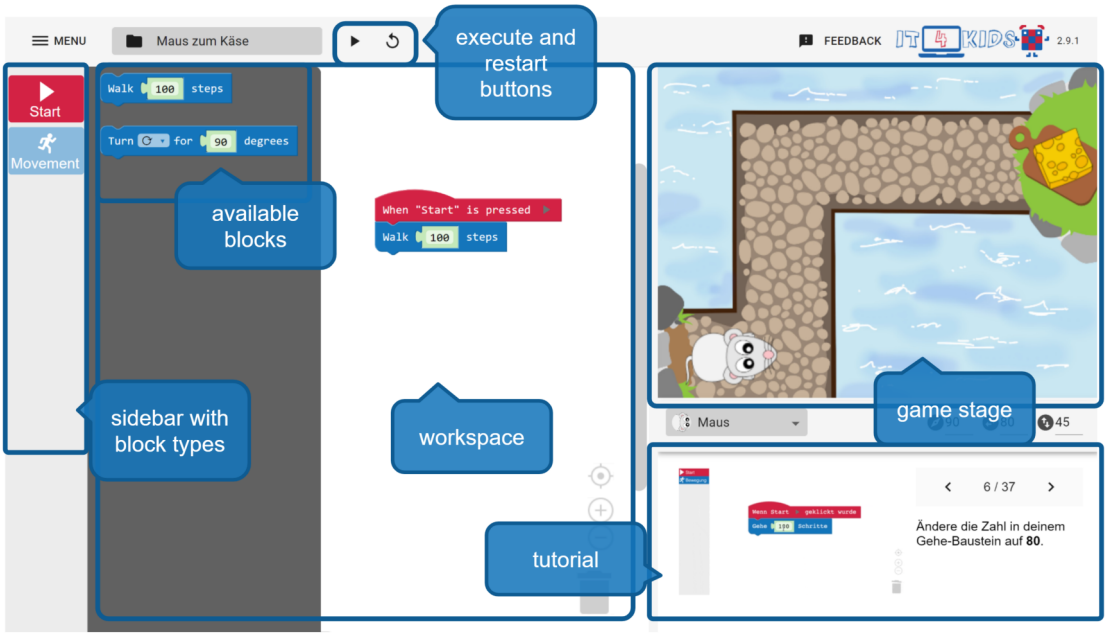


Figure 2.1.: Cubi's user interface (version 2.9.1): an illustrative level with component explanations [24]

2.2.1. Cubi's Design and Structure

Cubi is easily accessible through web browsers on PCs and tablets, eliminating the need for software installations. This design choice ensures convenient access to the platform. The user interface is designed with simplicity in mind, improving the experience for both teachers and students. Moreover, administrators retain the ability to customise the available blocks to suit diverse learning strategies. Cubi is versatile and supports use in multiple languages.

User Interaction and Programming Students use Cubi to create and play games, while interacting with their creations. The user interface is divided into two core sections: the coding area on the left and the stage on the right (see Figure 2.1). Users can drag blocks from the sidebar onto the stage and arrange them in a logical sequence. The block types are colour-coded to enhance their distinctiveness. At the top of the interface, two essential buttons are featured: "Execute Program" and "Restart Program". Upon execution, the programmed sprite behaves according to the defined logic. Furthermore, certain levels offer tutorials, presenting explanations through GIF pages.

Course Integration Users typically receive instructions from their instructors and access Cubi on their devices. Cubi primarily functions within structured courses and is less frequently employed as a standalone tool for independent learning, except when users wish to continue work initiated during a course.

Level Selection and Customisation Cubi provides predefined levels, created by IT4Kids, for teaching programming concepts. Users navigate the platform's menu to select their desired level. They can edit and save changes either locally or in cloud-based storage. Levels are saved as .cubi files, which can be downloaded locally. Additionally,

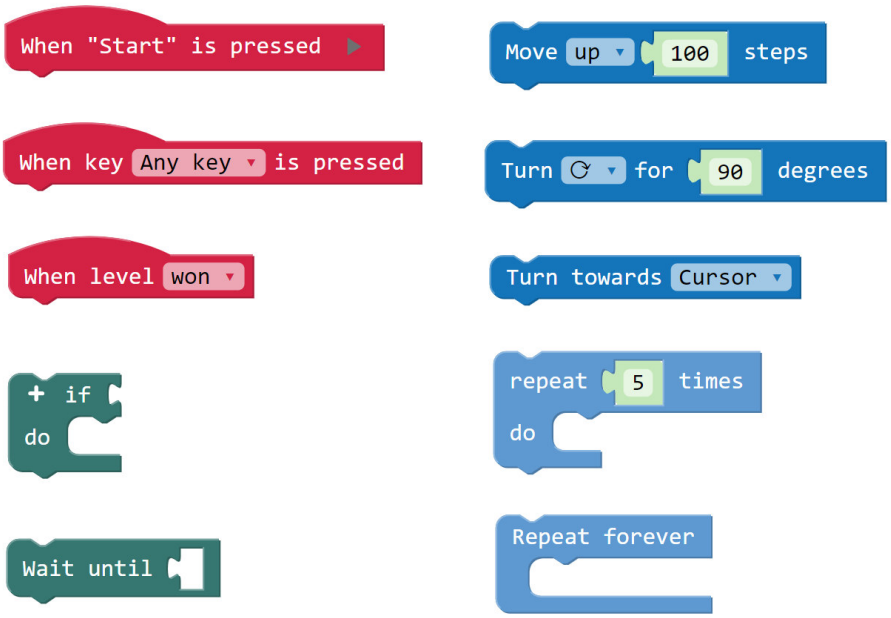


Figure 2.2.: Exemplars of diverse block types with varied inputs and constraints; Note that start blocks cannot have parent blocks, and connections are established only when feasible. [24]

users can open a new empty level, affording them greater creative freedom (sandbox-style). Access to restricted settings, which involve viewing solutions, restricting blocks/categories, and hiding blocks, is possible through a secret password known only to instructors or administrators.

Elements within Cubi Cubi contains two main elements: Sprites and Blocks. Sprites are programmable characters that give feedback on the code being executed. They can perform various actions, such as moving, changing costumes, being dragged, and delivering spoken or thought messages. Blocks have specific specifications and connectivity constraints, which determine how they can be linked to other blocks (see Figure 2.2). They have a variety of colours, which makes them easier to categorise and recognise.

2.2.2. Cubi's Limitations

Lack of user accounts

Despite its user-friendly design, Cubi has certain limitations. Cubi lacks user accounts, which facilitates quick initiation, but prevents long-term tracking of user activity. To continue working on the same level across multiple devices, users must either download and upload their progress locally or use cloud-based storage. Accessing the same level on another device via the cloud requires that the generated level code has been saved.
In addition, task requirements are communicated verbally by the teacher rather than being displayed on the screen. This reliance on verbal instructions from teachers can lead to misunderstandings and frustration among students.

2.3. xAPI

The Experience API (xAPI) is a technical specification, designed to streamline the collection and sharing of learning experiences [58]. Its main objectives are to facilitate the comprehensive understanding and comparison of learning experiences and outcomes across

contexts, platforms, and technologies. Moreover, the xAPI aims to provide a framework, that promotes seamless interoperability between services, involved in the creation, aggregation, storage, and processing of information about learning experiences.

2.3.1. Effectiveness of Learning Analytics

A notable study by Nouira et al. [37] highlights the xAPI's dynamic data model, which is adept at logging learners' experiences and performance. This alignment with the learning analytics (LA) domain underscores the xAPI's ability to track and store learning experiences and performances, and subsequently translate educational data into actionable insights, that improve the learning process - an overarching goal in the LA domain. LA have been shown to be useful [48] in improving teaching practice and identifying students at risk. The paper emphasises the importance of early assessment.

2.3.2. The xAPI Statements

xAPI statement components

An xAPI statement serves as a record of a learning experience, stored in a *Learning Record Store* (LRS). It comprises several key components, including the actor, verb, and object, with the optional inclusion of context and result. While these statements adhere to a machine-readable JSON format, they are equally adaptable to natural language descriptions. This can be extremely useful in the design process [58]. For instance, an illustrative xAPI statement might read as follows: *Shafi completed level 1* (see Listing 2.1).

Listing 2.1: Basic xAPI statement. This statement represents *Shafi completed level 1*

```
{
  "actor": {
    "name": "Shafrira Goldwasser",
    "mbox": "mailto:goldwasser@mit.com"
  },
  "verb": {
    "id": "http://adlnet.gov/expapi/verbs/completed"
  },
  "object": {
    "id": "http://example.com/activities/level-1"
  }
}
```

In addition to the essential properties demonstrated in Listing 2.1, it is important to recognise that xAPI statements possess a wide range of properties. As illustrated in Table 2.1, these properties provide contextual details, which can enrich the overall understanding and interpretation of learning experiences within the xAPI framework. The following paragraphs will provide more detailed explanations of the actor, verb, object, context, and result.

Actor The actor component defines the individual or group, responsible for performing the action specified in the statement. In the example provided in Listing 2.1, the actor is *Shafi*.

Verb The verb component describes the action or event, undertaken by the actor in conjunction with the object. In the example in Listing 2.1, the verb is *completed*.

Property	Description	Required
id	UUID assigned by LRS if not set by the learning record provider.	Recommended
actor	Whom the statement is about, as an agent or group object.	Required
verb	Action taken by the actor.	Required
object	Activity, agent, or another statement that is the object of the statement.	Required
result	Result object, further details representing a measured outcome.	Optional
context	Context that gives the statement more meaning. Examples: a team the actor is working with, altitude at which a scenario was attempted in a flight simulator.	Optional
timestamp	Timestamp of when the events described within this statement occurred. Set by the LRS if not provided.	Optional
stored	Timestamp of when this statement was recorded. Set by LRS.	Set by LRS
authority	Agent or group who is asserting this statement is true. Verified by the LRS based on authentication. Set by LRS if not provided or if a strong trust relationship between the learning record provider and LRS has not been established.	Optional
version	The statement's associated xAPI version, formatted according to semantic versioning 1.0.0.	Not Recommended
attachments	Headers for attachments to the statement	Optional

Table 2.1.: Details of each property of an xAPI statement. Table provided by [58]

Object The object component specifies the activity that the actor engaged in. The object can represent a broad array of activities, such as courses, quizzes, or games. In the example in Listing 2.1, the object is *level 1*.

Context The context component offers additional information surrounding the circumstances in which the activity occurred. This can encompass details, such as the instructor's name and the learning environment.

Result The result component provides insights into the outcome of the activity. This may include indicators of successful completion or the attained scores.

2.3.3. Learning Record Store

An essential component within the xAPI ecosystem, the Learning Record Store (LRS) serves as the central repository for xAPI statements, receiving, storing and facilitating their retrieval. The primary role of an LRS is to accumulate and provide access to learning records, as specified by the xAPI framework. Beyond its basic function of storing xAPI statements, an LRS opens up a realm of possibilities, enabling the tracking of a wide range of learning experiences.

2.4. Feedback

Programming education is recognised for its inherent challenges [4]. In this context, feedback plays a critical role in the learning process. When used skilfully, feedback becomes a powerful tool for promoting effective learning outcomes [20]. Typically, feedback tends to focus on past actions, often highlighting errors or omissions that require attention [26]. However, research also underscores the importance of positive feedback, as it bolsters learners' confidence [38].
Narciss [36] defines feedback as post-response information, aimed at informing learners about their current state of learning or performance. Similarly, Keuning [29] characterises feedback as a form of communication, directed at learners with the intent of reshaping their thinking or behaviour to enhance learning outcomes.
Feedback can be broadly categorised into two main types: formative and summative. Formative feedback seeks to guide learning by providing instructive insights, while summative feedback, often expressed as grades or percentages, delivers evaluative judgments on learners' work. It is worth noting that providing contextual information or explanations has been found to be more effective than offering solely summative feedback [20].

2.4.1. Feedback types

According to Narciss and Keuning, feedback in an interactive learning environment encompasses several types, as summarised below:
Simple feedback: This type of feedback doesn't aim to improve solutions and may even have a negative impact on student behaviour [29]. It includes:

- Knowledge of performance (KP): Offers summative feedback on achieved performance levels after completing multiple tasks.
- Knowledge of result/response (KR): Provides feedback on correctness.
- Knowledge of the correct result (KCR): Delivers descriptions of the correct solution.

Elaborated Feedback: This type provides learners with detailed insights and guides them towards a deeper understanding of concepts and solutions. It includes:

- Knowledge about task constraints (KTC): Offers insights to help learners navigate task-specific requirements and constraints. Subtypes include hints on task requirements (TR) and hints on task-processing rules (TPR).
- Knowledge about concepts (KC): Provides information to help learners understand the underlying concepts. Subtypes include explanations on subject matter (EXP) and examples illustrating concepts (EXA).
- Knowledge about mistakes (KM): Offers insights into different types of errors that learners might encounter. Subtypes include test failures (TF), compiler errors (CE), solution errors (SE), style issues (SI), and performance issues (PI).
- Knowledge about how to proceed (KH): Guides learners through refining their solutions and addressing errors. Subtypes include bug-related hints for error correction (EC), task-processing steps (TPS), and suggestions for improvements (IM).
- Knowledge about meta-cognition (KMC): Includes feedback related to the learning process itself.

2.4.2. Gamification

Gamification is the integration of game-like elements into non-game contexts to increase engagement and motivation. It capitalises on intrinsic human tendencies towards competition, achievement, and reward. The introduction of game-like elements can encourage learners to remain actively involved, thereby increasing their commitment to the learning process. To illustrate the effectiveness of gamification in boosting learner engagement and motivation, consider the following instances: Duolingo,[7] a language-learning app, uses gamification to transform the acquisition of a new language into an educational yet engaging and enjoyable experience. Users earn points, complete levels, and unlock achievements as they progress through language lessons. The app also includes a competitive element, allowing users to compete with friends and other learners [50]. Khan Academy,[8] an online learning platform, effectively harnesses gamification to inspire students to complete mathematical exercises. Within this context, learners can personalise avatars, gain points, and earn badges as they advance in their studies [17].

Examples of Gamification

[7] `https://de.duolingo.com/` *(Accessed: 13.09.2023)*
[8] `https://de.khanacademy.org/` *(Accessed: 13.09.2023)*

Chapter 3 Related Work

This chapter presents an analysis of previous investigations to establish the foundations for this thesis. The focus of this thesis includes the integration of an xAPI logging interface into Cubi, with a particular emphasis on exploring the potential of LA within Cubi and investigating diverse forms of feedback for block-based programming through Cubi. To guide this exploration, this chapter examines logging interfaces and investigates various feedback mechanisms for block-based programming.

3.1. Logging Interactions

In educational settings, the efficacy of feedback and hints has been shown to be most impactful when provided at the level of individual steps [26, 53]. This finding underlines the importance of logging students' steps within the learning proces.

Understanding the granularity of steps

Steps and Subgoals Jeuring's definition of a "step" [26] characterises it as either a code-edit, encompassing small keystrokes or the insertion of entire code blocks, or an user-action leading to environment actions like compilation, execution, hint retrieval, submission, or solution display. While code edits within BBPLs are primarily driven by mouse events, the essence of a step is determined by the code change itself.
The granularity of the steps can vary, with finer-grained collections, such as keystroke-level data, ensuring that the data is compatible with other code analysis efforts [26]. Section 3.1.1 will discuss existing logging interfaces with different granularities in BBPL.
"Subgoals" refer to groupings of steps that collectively contribute to achieving the ultimate goal. Breaking down a goal into manageable subgoals facilitates more effective learning [26]. The strategic use of subgoals enables learning environments to identify subsets of steps, thereby providing targeted feedback aligned with each subgoal.

Granularity of steps The granularity at which code edits are captured is important when analysing students' programming behaviour [26]. Different levels of granularity allow a more detailed understanding of the sequence of actions taken by learners. Ihantola et al. initially identified six such levels in their research [22], and Jeuring et al. have further enriched this categorisation by introducing an additional level involving tokens [26]. The following seven levels illustrate the range of granularity:

- keystroke: Capturing individual key presses, offering a fine-grained depiction of code composition.
- token: Identifying individual code elements; Often ending with a whitespace or a delimiter. In block-based programming, this could be a block or a block's argument.

- line: Recording entire lines of code altered during an editing process.
- file-save: Noting changes made to the codebase and saved as a file, encapsulating multiple edits.
- compile: Documenting compilation attempts and their outcomes, showcasing error correction efforts.
- execute: Monitoring the execution of code, unveiling learners' strategies in evaluating their solutions.
- submit: Marking the culmination of the programming task, signalling the student's readiness to share their work.

Fine-grained logging vs coarse-grained logging

The advantages of employing fine-grained logging, such as capturing keystrokes or individual code tokens, are notably significant within the context of this thesis. Fine-grained logging facilitates precise error tracking, empowering a more accurate pinpointing of the precise juncture at which an error materialised. Moreover, it enables an exhaustive analysis of coding behaviour, encompassing critical aspects such as coding speed, typing patterns, and the structural composition of the code. However, it is essential to acknowledge that fine-grained logging does possess certain challenges. Foremost among them is the substantial volume of data generated, often overwhelming the process of analysis and demanding substantial storage capacity. Furthermore, fine-grained logs, while rich in detailed actions, may sometimes lack the broader context necessary to comprehend a student's cognitive processes and their overarching problem-solving strategy.
Conversely, the utilisation of coarse-grained logging, exemplified by events like file-save or task submission, provides a more manageable data volume. Nevertheless, it falls short in providing the same level of granularity and insight into students' problem-solving processes that fine-grained logging affords. Logging at a coarser granularity, such as the line or file-save level, imparts valuable contextual information, aiding in the comprehension of the code's structural framework and its sequential flow. However, this approach faces challenges when attempting to precisely identify the specific point at which an error manifested. Furthermore, it may not capture every nuanced action, potentially resulting in the omission of vital details concerning the coding process.

3.1.1. Logging in Block-based Programming Environments

This subsection provides a comprehensive overview of related research regarding logging practices within block-based programming environments and offers insights into how novice programmers tackle problem-solving.

Tracking User Progress and Learning Trajectories Jiang et al.[27] employed an innovative methodology in their research by capturing code snapshots and constructing Abstract Syntax Trees (ASTs) every time an user clicked on the "execute" button in Snap!. This approach provided clear insights into the incremental progress of students' coding activities. By analysing the ASTs, the study identified four prominent trajectories within block-based programming: "quitters", "approachers", "solvers" and "knowers". Notably, this work represents an unique exploration of the nuanced ways in which students approach learning. . Identifying these personas not only offers a means to visualise student progress but also holds promise for tailoring personalised feedback strategies. This approach could encompass providing additional explanations to struggling students and assigning more complex tasks to proficient learners.

Execute-level logging

keystroke- and token-level logging

Leveraging Log Data for Teacher Interventions in Scratch Wu et al. [57] leveraged log data from Scratch interactions to analyse student behaviour and how the platform facilitates learning, with teacher interventions enhancing the learning process. The system logged the writing procedures and paths of students during programming assignments, aiding teachers in identifying students' difficulties and delivering timely feedback. The study assessed factors such as the utilisation of computational thinking skills, learning anxiety, motivation, and confidence. The findings underscore the significance of swift interventions and support, which not only alleviate anxiety but also boost learning motivation and self-confidence.

Analyzing Student Behaviour and Learning Patterns Filva et al. [13] conducted a detailed examination of student behaviour while coding with Scratch. They analysed user clicks within the interface, leading to the creation of predictive analytics that unveiled patterns in students' actions during problem-solving. The study introduced categories such as development-focused, organisation-focused, design-focused, and multimedia-focused students, providing educators with a tool to identify areas where students may be spending disproportionate amounts of time. These findings are invaluable for educators and align with a teacher-centered approach. In addition, Morshed et al. [35] have identified trajectory-based student modelling for novice programmers using block-based programming. Their research identified distinct progression trajectories, including early termination, high persistence, and efficient completion. Detailed analysis showed that these models can accurately predict student behaviour in future programming activities. This predictive capability offers the potential to adapt the learning environment by providing tailored guidance or feedback to students. Notably, the research also revealed that students are more likely to choose early termination as the difficulty of the tasks increases.
Despite the valuable insights gained from these studies, it is recognised that mapping these click events to specific levels of granularity can be challenging, particularly in the context of web-based applications such as Scratch. Nevertheless, the results of these research efforts contribute significantly to the understanding of student behaviour and learning patterns in block-based programming environments, providing actionable insights for educators and researchers in the field.

3.1.2. Logging xAPI statements

To date, no studies have been identified that utilise xAPI statements to log user interactions within block-based programming environments. However, research has demonstrated the successful application of xAPI statements for logging user interactions in alternative programming environments.

An xAPI Logging Interface for JupyterLab Within the environment of JupyterLab, Juxl has emerged as an xAPI logging interface that captures user-generated learning interactions during programming sessions within JupyterLab. Juxl [52, 6] serves as the foundational software that enables logging capabilities in JupyterLab, and several Juxl extensions are presented that demonstrate the potential for learning analytics and gamification within JupyterLab. In addition, comprehensive documentation is provided, including deployment and configuration guides for Juxl and Juxl-Hub-Auth, API documentation, a problem-focused example repository, and an xAPI vocabulary to support Juxl extension

BBPL	Feedback Options	KP	KR	KCR	EF												
					KTC		KC		KM					KH			
					TR	TPR	EXP	EXA	TF	CE	SE	SI	PI	EC	TPS	IM	KMC
Scratch	execute-button		✓														
	tutorials						✓										
Blockly	predefined connections of code blocks									✓		✓					
	pseudo code							✓									
	execute-button		✓			✓	✓		✓					✓			
	hint-button					✓	✓							✓	✓		✓
	debugger										✓						

Table 3.1.: Kiesler's analysis of applied feedback types in block-based programming environments [30].

development. The documentation of Juxl serves as a valuable resource for the present thesis, providing a solid foundation for the implementation of CubiXLI.

3.2. Feedback in Programming Learning Environments

Feedback plays a central role in programming learning environments, influencing students' understanding, motivation and problem-solving skills. Recent research efforts have explored various aspects of feedback provision in block-based programming contexts, providing valuable insights and methodologies.

3.2.1. Existing Feedback Strategies in Block-Based Programming Environments

Kiesler's research [30] provides a comprehensive analysis of different feedback types within BBPLs. This analysis sheds light on the diversity of feedback strategies used in these environments and provides valuable insights into their strengths and weaknesses. The table 3.1 provides an overview of the different types of feedback in BBPLs.

3.2.2. Challenges in Feedback Strategies

Keuning et al. [28] found a tendency in numerous feedback tools. Feedback primarily concentrates on final outcomes, often neglecting the entirety of the programming process, which is central to cultivating improved problem-solving skills. Furthermore, hints that prescribe the subsequent steps, focusing on the "what," frequently overlook the underlying "why" behind the proposed actions [34]. Keuning [28] highlights the obstacles that learners and educators encounter in programming. These challenges include program design, debugging, and dividing functionality into procedures. It is therefore imperative to develop strategies and guidance to help students overcome these challenges.

3.2.3. Enhancing Feedback Mechanisms for Novice Programmers

This subsection explores a range of feedback strategies within programming learning environments, each designed to improve the learning experience for novice programmers.

Interventions for novice programmers Dong et al. [10] conducted an investigation into the importance of interventions in block-based programming environments, identifying six different categories of intervention. Their research highlights the need for targeted support for novice programmers, who often exhibit inappropriate help-seeking behaviours. Their findings highlight the need for automated interventions to facilitate learner progression.

Personalised and real-time feedback Marwan et al. [33] have addressed the generation of personalised and adaptive feedback for novice programmers. Recognising the need for real-time support, their system employs immediate feedback mechanisms that promote engagement, motivation and, improved performance.

Feedback on Demand In another study by Marwan et al. [34], an "emergency" button was introduced to allow users to request feedback when needed. Efficacy assessments were conducted to determine responses to the feedback provided, distinguishing cases where hints were followed from those where they were ignored. A hint was considered to be followed if the student implemented the suggested block before receiving a new hint, while hint disregard was measured when users chose not to implement the suggested guidance. Reasons for hint disregard included confusing explanations, interpretation challenges, or users seeking feedback from alternative sources not detected by the automated system. It was also observed that while text prompts encouraged users to engage with the 'feedback on demand' feature, this did not always translate into improved learning outcomes or performance.

Guidelines for On-Demand Feedback An additional challenge in the area of on-demand feedback is that individuals who stand to benefit the most are often reluctant to seek guidance [2]. To address this issue, guidelines for effective feedback design have been proposed. These guidelines advocate the provision of on-demand hints based on underlying principles that facilitate content understanding. They also advocate providing bottom-out hints that help users overcome specific barriers. In terms of composition, effective feedback should succinctly encapsulate problem-solving knowledge by explaining both the required action and the contextual conditions that justify that action, along with a reference to the relevant domain principle. In addition, clues should briefly explain how the problem-solving principle is consistent with the current problem step. In particular, achieving conciseness while meeting these criteria is a nuanced challenge.

Reluctance in seeking feedback

Individualised next-step guidance through automated testing Automated testing is emerging as a powerful way to provide formative feedback [18]. The work of Deiner [9] and Obermüller [39, 12] serve as examples of this approach, using automated test creation to evaluate solutions and generate personalised next-step recommendations. Such methods ensure that feedback is tailored to students' unique solution paths, thereby increasing its relevance and effectiveness.

Data-driven feedback algorithmic approaches Price et al. [44] venture into algorithmic approaches, such as SourceCheck, which aim to identify the most appropriate next-step hints based on previous correct solutions. Their research highlights the delicate balance between supporting learners and avoiding counterproductive hints.

Mapping concepts to tasks A common pitfall in block-based programming is the prevalence of auto-grading feedback, which, while abundant, often lacks the depth required to promote meaningful learning [19]. At times, feedback is either absent or fails to provide learners with substantive support. Because block-based programming focuses primarily on teaching basic concepts, each block inherently represents a specific concept. This feature simplifies the mapping of tasks to these basic concepts. As explained in the paper by Haldeman et al. [19], the generation of hints should aim to guide users towards a

conceptual reassessment of their code, rather than simply suggesting changes that lead to convoluted solutions.

Visually appealing and understandable feedback While compiler error messages may not be directly applicable to block-based programming, it is worth noting that studies suggest that they are often unhelpful to novice programmers, and can even be a source of frustration and discouragement [19]. In the context of Cubi, where users are young children still struggling with literacy, feedback needs to be not only informative, but also visually appealing and easy to understand.

Perfect timing of feedback Timing is crucial when it comes to feedback. As Friemel et al. [16] argue, feedback should be delivered precisely when the user needs it, avoiding premature or intrusive interventions. Ideally, feedback should be triggered automatically after certain user actions, such as periods of inactivity, or when the system detects that the user needs assistance. This aspect can be particularly challenging due to its inherent individuality. However, it is important to recognise that both on-demand and automatically-triggered feedback have their own strengths and weaknesses.

Guiding Without Revealing Feedback should not simply reveal the correct answer, as Fisch et al. advocate [14]. Providing the answer immediately after a single incorrect attempt deprives learners of the opportunity to discover the correct solution for themselves. Instead, feedback should progressively support children as they strive to find the correct path, ensuring that each hint contributes to their understanding of the underlying concept.

Effective hint systems In their article, the authors of [43] outline the key attributes of a successful hint system. Hints should guide users to the correct solution without inadvertently exposing them to spoilers. The incremental nature of hints, where subsequent hints provide additional nudges in the right direction. It is important to discourage punishment for using hints, as users tend to use them only when they really need them.

Balancing level difficulty Balancing the difficulty of a level is a challenge in itself [21]. Hint systems can play a key role in achieving this balance. Finding a balance between hints that are too simple, potentially revealing too much information, and those that are too abstract is critical. Categorising hints and recognising their drawbacks, such as on-demand or delayed hints, is essential to optimising the learning experience.

Attention to hints Concati et al. [8] point out that several factors influence how learners engage with hints. In particular, studies show that users tend to ask for help infrequently, which is a weakness of on-demand feedback but a strength of automatic feedback. Providing too many hints can lead users to ignore future hints, especially if they are repetitive and do not improve in quality over time. The level of knowledge of the user also plays a role in the use of hints. An interesting aspect of hint effectiveness is its relation to the user's reaction after receiving the hint. Hints tend to get more attention if they are really needed or if the previous hint led to a successful action. The user's attitude towards seeking help has a significant impact on their engagement with hints. Users who readily accept help often appreciate confirmation that they are on the right track. Conversely,

people who typically avoid help may be reluctant to seek it when faced with an insurmountable challenge.
To optimise the delivery of hints, it is crucial to develop a strategy for identifying each user's personality and tailoring the presentation of hints to their expected attention span. Repetition of identical hints should be avoided, as it signals that hints need to evolve and improve with each interaction. Repetition can be counterproductive. In addition, users who read hints carefully are more likely to make correct moves.
However, it is important to be careful when providing hints. Some users may use hints to manipulate the system and cheat, while others may simply prefer to tackle challenges unaided. Timing is also critical. Hints need to be delivered at the right moments to capture the user's full attention.

Gamification Gamification in block-based programming can be a powerful motivator for learners. By incorporating elements such as badges, leaderboards and achievements, it encourages motivation, engagement and a sense of achievement. Learners are driven to complete tasks, compete and achieve clear goals in their learning journey. This extrinsic motivation, coupled with healthy competition, can encourage learners to invest more time and effort in mastering programming concepts.

Progress tracking Progress tracking provides learners with a visual representation of their coding journey. It provides a clear visualisation of their growth as programmers, allowing them to compare their initial attempts with their current skills. This self-assessment tool helps learners set realistic goals, make informed decisions about their learning path, and foster a growth mindset. Educators can also benefit from progress tracking data to provide targeted support and evaluate teaching strategies.

Code optimisation suggestions In the realm of block-based programming, code optimisation suggestions provide learners with valuable feedback on how to improve the efficiency and performance of programs created using blocks. While the process may differ from traditional text-based programming, the basic principles of optimisation remain relevant. Recent research by Obermuller et al. [38] has introduced the concept of "code perfumes" in block-based programming, analogous to "code smells" but with a positive connotation. Recognising and praising these code smells, as defined in their 25 patterns, becomes a crucial aspect of providing constructive feedback, as positive feedback has been shown to increase intrinsic motivation. In addition, the findings of Fradrich et al. [15] in identifying common programming errors in platforms such as Scratch can complement code optimisation by highlighting pitfalls and providing solutions. In the context of Cubi, the integration of code optimisation suggestions aligns with the goal of promoting good programming practice alongside the teaching of programming concepts, enabling learners to produce efficient, well-structured code that contributes to their growth as competent programmers.

3.2.4. Evaluation of Feedback

The evaluation of feedback mechanisms has received considerable attention in the field of programming education. A study investigating educators' feedback preferences sheds light on the design and implementation of next-step hints in programming exercises [40]. In particular, the research argues for the inclusion of higher-level hints, highlighting their effectiveness over syntax-based recommendations. These higher-level hints address

missing concepts, inadequately covered inputs, suggest simplified problem-solving approaches, and recommend the removal of extraneous code. The study also recommends the integration of explanatory text to improve learner understanding. In addition, Price et al. [45] emphasise the importance of comprehensive error explanations in feedback. Striking the right balance between specificity and generality in feedback design emerges as a critical consideration to address different user needs and skill levels.

Evaluation of data-driven Feedback Shabrina et al. [47] have conducted an investigation into the effectiveness of feedback generated within block-based programming environments. This investigation covers several dimensions, including the accuracy of the feedback, its role in guiding students through programming exercises, its contribution to task completion, its effectiveness in error situations, and users' responses to the feedback provided. The results underline the invaluable role of data-driven feedback in supporting student learning. Notably, the research also highlights potential challenges arising from students' over-reliance on feedback, which reduces their propensity to self-evaluate. To address this, the authors suggest strategies such as incentivising self-evaluation and implementing on-demand hints. Significantly, the research shows that feedback serves as a vital support mechanism for struggling students, positively influencing their learning trajectories.

Evaluation of proactive interventions Choi's study [7] provides empirical evidence of the effectiveness of proactive interventions in improving students' performance. The research highlights a positive correlation between the frequency of interventions and students' likelihood of passing exams. In addition, the study identifies at-risk students by analysing their click-activity patterns, providing a proactive approach to addressing their needs.
Choi also argues that the type of proactive intervention should be tailored to students' motivations. For example, high-achieving students may benefit from student-centred interventions such as e-tutorials that cater to their self-directed learning tendencies. Conversely, students facing academic challenges may require counselling and support to stimulate improvements in their attitudes, behaviour and cognitive skills. This personalised approach to proactive intervention holds great promise for promoting holistic student development.

Chapter 4 Research Design

This chapter outlines the research design adopted for this Master's thesis. The first section formulates two research questions, followed by successive sections that address methodologies aligned with these questions. The order of the sections reflects the order of the research questions.

4.1. Research Questions

This section outlines the fundamental research questions, that underpin the investigative trajectory of this thesis. The primary aim is to investigate feedback mechanisms in the context of block-based programming environments. By exploiting the inherent advantages of block-based programming languages BBPLs, known for their user-friendly interfaces and support for iterative exploration, this thesis aims to contribute the enhancement of the learning experience of novice programmers. Additionally, it addresses the significant challenge of delivering effective and timely feedback to users navigating the Cubi environment.

[RQ1] What are loggable learning experiences for automated feedback generation in Cubi? Recognising the significance of feedback in programming education and the potential limitations of human intervention, the thesis aims to explore the potential of automatically generated feedback as a means of enhancing the learning journey. To achieve this, the thesis implements *CubiXLI*, an xAPI logging interface, in order to collect relevant data. The aim of this thesis is to track learners' interactions and behaviours as they complete tasks, with the aim of collecting data for subsequent data mining. This data can then be used to identify patterns, pinpoint areas of difficulty and differentiate between learners who need additional support and those who can tackle more challenging tasks.

[RQ2] What are the existing and potential feedback types in block-based programming? In the realm of block-based programming, a crucial question arises: What feedback is really valuable for novice programmers of BBPLs? The second research question explores this question by identifying the feedback elements that could potentially effectively improve the skills and understanding of novice programmers.

4.2. Logging Learning Experiences

This section explores the first research question, namely *[RQ1] What are loggable learning experiences for automated feedback generation in Cubi?*. The discussion that follows addresses the learning experiences suitable for integration into Cubi's logging framework. Both theoretical and practical aspects of loggable experiences will be explored, examining what

facets of user interactions and activities can be captured to facilitate the generation of automated feedback. In addition, this section will consider the inherent limitations of the logging process, providing a comprehensive overview of the scope and constraints involved in capturing and using learning experiences to improve the feedback mechanism within the Cubi environment.

4.2.1. Learning experiences in block-based programming

Learning experiences

The term "learning experience" encompasses various interactions or engagements in which learning takes place [42]. In the context of BBPL, these learning experiences might be manifested as interactions with the online platform. For example, a user can drag and connect blocks of code in the workspace and then run the program to observe the behaviour of the sprite. In cases where the sprite doesn't perform as intended, the user can identify and correct errors in their code. This sequence illustrates how user engagement with the platform triggers the learning process.

Where does the learning take place

However, it is important to recognise that learning can also take place without direct manipulation of the editor. For example, thinking about the task description, examining the stage and strategising how to approach a challenge all contribute to the learning process even before interacting with the editor.

This thesis focuses on investigating the learning experiences that occur through interactions with Cubi's programming interface. It aims to provide valuable insights into how these interactions facilitate the acquisition of programming and problem solving skills.

4.2.2. Theoretical logging capabilities

In theory, within Cubi, it is possible to log a wide range of user interactions, much like user monitoring applications in web environments. Cubi's frontend, which manages user interactions, can be accessed because it's built using TypeScript. This accessibility allows for the potential logging of actions.

Imagine a scenario without any privacy constraints, where the goal is to capture the full extent of the user's engagement with the platform. This would require logging actions such as selecting, dragging, connecting and creating elements within the programming workspace, among other activities. Such extensive logging would provide a granular understanding of how learners interact with Cubi and potentially provide valuable insights into their learning journey.

It is important to note, however, that there are several feasibility challenges to implementing such extensive logging capabilities, primarily related to privacy concerns and the potential burden on system resources. Logging every single user interaction would generate an immense amount of data, requiring significant storage capacity and computing resources. In addition, the generation of vast amounts of data has the potential to pose a threat to user privacy and raise ethical concerns. Therefore, while the theoretical scope for logging may be vast, practical implementation must carefully balance the benefits of data collection with the challenges of resource limitations and privacy.

4.2.3. Practical logging capabilities

Capturing user interactions within Cubi requires a practical approach that navigates the complexities of event logging. While events such as clicks and keystrokes can be captured through browser events, it is important to recognise that these represent the elementary layer of user activity. The real challenge lies in deciphering these low-level

events to extract higher-level actions that provide deep insights into the learning process. Central to Cubi's logging capabilities is Blockly, the underlying programming framework. Blockly has a built-in event handler designed to extract information from user interactions. Importantly, Blockly takes this a step further by categorising events, simplifying the process of distinguishing between mundane UI interactions and substantive actions of pedagogical relevance. This event classification helps to ensure that the logged data remains focused on actions that are central to the learning experience.
For elements outside the immediate scope of Blockly, such as those within the stage or controls such as play and stop buttons, custom event listeners play a key role. These custom listeners extend Cubi's logging capabilities to a wider range of user interactions, ensuring that no valuable data goes unnoticed.

In essence, this subsection underscores Cubi's commitment to robust logging capabilities that enable the platform to delve deeper into user interactions. By intelligently capturing, categorising, and interpreting these interactions, Cubi could not only improve its understanding of user behaviour, but also use this knowledge to improve the learning experience, benefiting both learners and level creators.

4.2.4. Meaningful Logging in Cubi

In the Cubi programming environment, capturing user interactions for meaningful logs is essential for understanding the learning process and optimising the user experience. However, not all interactions are equally valuable for analysis. This section discusses the criteria for identifying and logging meaningful interactions in Cubi, taking into account the different user groups and types of interactions available in the platform.

User groups Cubi accommodates four distinct categories of users: novice learners, intermediate learners, teachers, and administrators, often referred to as "level coders". Each of these user categories has different behaviours and requirements within the platform, which requires a fine-tuned approach to logging interactions. When designing the logging system, it is imperative to make clear distinctions between these user categories.
Novice learners, typically aged between 6 and 10, often have limited or no previous experience of programming and may face literacy challenges.
Intermediate learners, who are in the same age group as the beginners above, may have previous experience of either another block-based programming environment or even text-based coding.
Teachers, who give lectures and oversee the learning process. They have privileged access to solutions and task descriptions, which enhances their role in guiding learners.
Administrators, often the architects behind level creation, play a key role in shaping the educational content within Cubi.
It is important to acknowledge that when users interact with a level, some blocks may already exist due to the efforts of the level coders. Therefore, the logging system must be capable of differentiating between user-created blocks and those that were present from the beginning.

Interaction	Description
Creating Blocks	Adding new code blocks to the workspace
Deleting Blocks	Removing code blocks from the workspace
Modifying Blocks	Changing the parameters or properties of blocks
Connecting Blocks	Linking code blocks together to create sequences
Running Code	Executing the program to see the result
Pausing Code	Temporarily stopping the program execution
Stopping Code	Ending the program execution
Creating Sprites	Adding new characters or objects to the stage
Switching Sprites	Changing the active sprite on the stage
Renaming Sprites	Giving a new name to a sprite
Deleting Sprites	Removing sprites from the stage
Creating Costumes	Designing new appearances for sprites
Switching Costumes	Changing a sprite's appearance during runtime
Deleting Costumes	Removing costumes associated with sprites

Table 4.1.: All possible user interactions in Cubi.

Identifying Interesting Interactions Cubi offers a range of interactions (see Table 4.1), from basic actions such as typing, resizing blocks and navigating the workspace, to more complex operations such as deleting a sprite, which automatically removes the associated blocks, or moving a block and its child elements simultaneously. While these interactions may seem straightforward, their pedagogical relevance requires careful evaluation. The key question is whether they are worthy of inclusion in the xAPI statements. Are they likely to provide enlightening insights, or do they risk contributing to a superfluous accumulation of data?

Focusing on Meaningful Interactions Meaningful interactions in Cubi are those that contribute to enhancing user feedback, identifying learning patterns, and optimizing the user experience. It is essential to justify the inclusion of specific interactions in the logs. For example, logging execution button presses is valuable in assessing the extent to which learners are following instructions and achieving learning objectives. This data helps educators measure learners' progress and ensure that the learning process is aligned with pre-defined educational goals.
It is important to exercise discretion in the logging process to avoid redundancy. Not every interaction warrants inclusion in the logs, as redundant logging can clutter the dataset. Exceptions to this guideline should only be made when such interactions offer unique insights or hold intrinsic importance within the broader user experience.

Beyond Interactions: Encompassing Critical Metadata The scope of logging goes beyond simple interactions to include key metadata. This includes a session ID, which, in the absence of user accounts, is crucial for tracking user progress across multiple levels. The recording of Cubi's version, the user's language preferences, the level name and the browser used by the user form a tapestry of data that enhances the contextual understanding of user interactions.

Exploring the logging strategy

Choosing Between Blocks and Code Logging One of the key decisions in Cubi's logging strategy is whether to log the individual blocks used in the programming process or the resulting code. While both options have their merits, there are several compelling reasons to log blocks:

- Granular understanding: logging blocks offers a more granular understanding of user interactions. It captures each discrete action taken within Cubi, including the placement, connection, and manipulation of blocks. This granularity is invaluable for educators and researchers seeking to comprehend how learners engage with the platform at a fine level of detail.
- Educational insights: blocks are the building blocks of learning within Cubi. They represent the fundamental concepts and actions that learners manipulate to create code. By logging blocks, the system gains insights into how users conceptualise and assemble their code, shedding light on their problem-solving strategies and learning progress.
- Flexibility with Blockly: Cubi's underlying framework, Blockly, is inherently block-based. Blockly seamlessly translates blocks into code and vice versa. This means that there is no need to log the translated code separately, as it can be derived from the logged blocks. This streamlined approach reduces redundancy in the logging process while retaining all the necessary information.

While logging code has its merits, focusing on logging blocks aligns closely with Cubi's block-based approach, allowing for a more detailed and insightful understanding of user interactions. Blockly's ability to seamlessly translate blocks into code further enhances the efficiency and effectiveness of this approach.

Summary In summary, Cubi's logging strategy aims to prioritise interactions that align with the learning objectives of the users. This strategic emphasis on meaningful interactions, underpinned by a robust rationale for inclusion, enables Cubi to collect a wealth of data. This data not only enhances the learning experience, but also provides Cubi with the insight needed to fully understand user behaviour and preferences.

4.2.5. Limitations

While Cubi's logging process provides valuable insights, it has inherent limitations.

Privacy concerns: Privacy concerns necessitate pseudonymisation of user data, particularly as Cubi is often used in shared educational environments with shared devices. As Cubi does not currently have user accounts, pseudonyms are used to protect user identities. However, this pseudonym-based approach limits the ability to comprehensively track an individual's learning progress over the course of a course, as device sharing can obscure the identity of individual learners. Privacy will be discussed in more detail in section 4.3.

Data volume and dark data: Collecting rich data may seem appealing, but practical considerations are critical. The concept of *dark data*, as explained in Al Kez's paper [1], highlights the potential downside of accumulating redundant or unnecessary data. Storing such dark data can contribute to increased carbon emissions, escalate a product's environmental footprint and drive up costs [1]. To address sustainability concerns, CubiXLI prioritises capturing specific and meaningful interactions that provide valuable insights, rather than logging every conceivable action.

Software version discrepancy: Cubi 3.0 is currently in development, while the previous version, Cubi 2, is actively used by the community. CubiXLI is designed for the newer version, which means that there is a lack of features that should ideally be available. While this thesis uses all of the functionality offered, it recognises the potential for more interactions in the newer version if it becomes the standard.

4.3. Privacy

Privacy is a fundamental consideration in any digital environment, including Cubi. It includes protecting users' personal information, controlling how their data is collected and used, and ensuring transparency and trust within the platform[46]. This thesis focuses on privacy in the context of learning analytics.

What is privacy Privacy in the digital realm refers to the right of individuals to keep their personal information, activities and communications confidential. It includes control over one's data, including who has access to it and how it is used. In the context of Cubi, privacy ensures that learners, teachers and administrators can engage with the platform without compromising their personal information or learning experience.

Why is privacy important Protecting the privacy of performance data is paramount and requires strict adherence to privacy standards [52]. Failure to address user concerns in this area may result in a suboptimal user experience and may even lead to the implementation of restrictive policies, potentially hindering the benefits that LA can bring. This thesis is dedicated to the task of not only acknowledging but also mitigating these privacy concerns, with the ultimate goal of fostering a safer and more productive learning environment to all users.

Anonymisation One of the most popular approaches of protecting personal data is anonymisation [41]. It involves the process of removing or encrypting personally identifiable information (PII) from user data. PII includes any information that can be used to directly or indirectly identify an individual, such as a name, address, telephone number or email address. In Cubi, this could include specific user account information if and when user accounts are implemented in the future.
Two key strategies for preserving privacy when working with user data are complete anonymisation and pseudonymisation. Complete anonymisation is a technique in which all data identifiers, including those of individual users, are completely anonymised, either by replacing them with the same identifier or by rendering them completely unidentifiable. This process removes all traces of individuality from the data. While complete data anonymisation can increase privacy to an extreme degree, it has significant consequences for the quality of the data. For example, it can make it impossible to track the progress of individual users over time, as their data is no longer identifiable.

Pseudonymisation Pseudonymisation is a related concept that involves replacing PII with pseudonyms or other non-identifiable data. Unlike anonymisation, which renders data completely unidentifiable, pseudonymisation retains a degree of identifiability through the use of pseudonyms or codes. It strikes a balance between protecting privacy and maintaining the ability to derive meaningful insights from the data.

Summary Although complete anonymisation of data provides better privacy, it may hinder the capability to obtain valuable insights from user data. Pseudonymisation, on the other hand, strikes a balance between keeping the data less identifiable yet retaining enough information for analysis. In educational platforms such as Cubi, where customised learning experiences and effective analytics are fundamental, achieving a balance between privacy and analytical utility is crucial.

4.4. Standardisation

In the field of LA, standardisation plays a key role in ensuring compatibility and interoperability of data across different systems and platforms [58]. xAPI, also known as the Tin Can API, is a fundamental standard for LA , providing a structured framework for tracking and reporting on learning experiences.
xAPI goes beyond data tracking; it establishes a common language and protocol for recording and sharing learning-related activities. One of its key strengths is ensuring interoperability of data across LRS, the repositories where learning experience data is stored.

Standardisation challenges While xAPI specifies data representation and communication interfaces, it does not impose a fixed vocabulary. This flexibility allows different systems to use their own terms, which can sometimes lead to challenges in data interpretation. For example, different systems may use different verbs to describe similar actions.

The role of registries To improve data interoperability and facilitate data analysis, it is advisable to adopt common verbs and objects from xAPI registries. These registries define standardised sets of terms that represent common actions and objects in the learning context. By adhering to widely accepted verbs and objects, data analysts can efficiently filter and analyse statements without the ambiguity of different definitions.

4.4.1. Block-based vocabulary

In the context of Cubi's programming environment, designing a comprehensive vocabulary for data representation is a particular challenge. Unlike traditional text-based programming, Cubi uses visual blocks to effectively teach coding concepts. The challenge is to accurately define and represent these block-based interactions in a standardised format.

	Name	Description
Activities	Cubi	A block-based programming environment.
	Block	A graphical programming element that represents a command or action.
	Sprite	A graphical object or character that can be programmed.
	Stage	A visual canvas where users can see the results of their programming efforts.
Verbs	Created	An actor pulled a coding block out of the toolbox.
	Deleted	An actor deleted a code block.
	Changed	An actor changed the code block's input.
	Moved	An actor moved a coding block inside the workspace to a different position.
	Loaded	A code block was loaded into the workspace.
	Renamed	An actor has renamed something.
	Switched	An actor has switched to another sprite.
	Executed	An actor has executed the program.
	Paused	An actor has paused the program.
Extensions	Browser	The browser used by the user.
	Level	The actor's current level within the block-based programming environment.
	NewParent	The block's parent after the block was moved.
	OldParent	The block's parent before the block was moved.
	NewValue	The block's value after the change.
	OldValue	The block's value before the change.

Table 4.2.: Cubi's xAPI vocabulary

Cubi's vocabulary for xAPI statements has been carefully crafted to capture the nuances of user interactions within the platform. The list of the new vocabulary is shown in Table 4.2.
It is important to emphasise that the table above provides a basic overview of Cubi's xAPI vocabulary, but it only scratches the surface of Cubi's extensive logging capabilities. In pursuit of a comprehensive insight for this thesis, Cubi's logging system includes additional elements that are not explicitly enumerated in the Table 4.2.

Distinct actor roles Building upon the user roles introduced in section 2.2, these roles will be grouped into two distinct actor categories. Novice and intermediate learners will collectively assume the role of "learners," while teachers and administrators will be categorised as "administrators." Learners represent users actively engaged in the learning process, while administrators, also known as "level creators", shoulder the responsibility of creating learning tasks. This distinction of roles is crucial in understanding the nuances of user interactions.
For example, when a learner creates a new block, they take on the role of actor, whereas in scenarios where pre-existing blocks are loaded onto the screen, the administrator becomes the actor. This distinction is of paramount importance as it provides crucial context for interpreting and analysing user interactions within Cubi.

Contextual depth The context component includes key elements such as session ID, the exact Cubi platform version being used, and the language preference selected (see Listing 4.1). The depth of contextual activity is dynamically adjusted in response to user interactions. For example, when users manipulate blocks, whether by moving or deleting them, contextual activity records intricate details such as the new parent, associated child blocks and the associated sprite. In addition, these logs maintain historical context by documenting the previous parent in such cases.

Listing 4.1: Context excerpt from the xAPI statement: Displaying session ID, platform version, and language settings.

```
{
  "context": {
    "registration": "4f068e34-3633-42e3-8d00-55ac33b18d90",
    "platform": "cubi/3.0.0-alpha-dev",
    "language": "de"
  }
}
```

Comprehensive block documentation CubiXLI captures detailed information about every interaction with the blocks. This information includes a unique block ID, the name of the block and its category. The unique block ID is made up of the block name and its Blockly ID, which is assigned by the Blockly framework. In addition, the exact category is stored in the extension field (see Listing 4.2). This thorough method of identification plays a crucial role in distinguishing between multiple blocks of the same type and gaining insight into their usage patterns.

Listing 4.2: Object segment of the xAPI statement: Specifically, this refers to the *controls_repeat* block within the *loop_block* category.

```
{
  "objectType": "Activity",
  "id": "https://xapi.elearn.rwth-aachen.de/definitions/
     blockBasedProgramming/activities/controls_repeat-4I03DxU}8hw;(
     $I864AO",
  "definition": {
    "name": {
      "en-US": "controls_repeat"
    },
    "type": "https://xapi.elearn.rwth-aachen.de/definitions/
       blockBasedProgramming/activities/block",
    "extensions": {
      "https://xapi.elearn.rwth-aachen.de/definitions/
         blockBasedProgramming/activities/blocktype": "https://xapi.
         elearn.rwth-aachen.de/definitions/blockBasedProgramming/
         activities/loop_block"
    }
  }
}
```

Related blocks Beyond logging individual blocks, CubiXLI captures information concerning associated blocks. This includes the parent block and the child blocks, that are nested within the current block. These details are recorded within the ContextActivities section of the xAPI statement, denoted as "parent" and "grouping" (see Listing 4.3 for an illustrative example). Furthermore, CubiXLI documents the current sprite details within the category extension. This data plays a key role in understanding the structure of the code and the relationships between different blocks.

Listing 4.3: ContextActivities section of the xAPI statement: Shows the parent block (the *move* block), a child block (a "*math_number*" block) and the current sprite.

```
{
  "contextActivities": {
    "parent": [
      {
```

```
        "objectType": "Activity",
        "id": "https://xapi.elearn.rwth-aachen.de/definitions/
           blockBasedProgramming/activities/move-bc|$,S]#9V`RJxyabyI4
           ",
        "definition": {
          "name": {
            "en-US": "move"
          },
          "type": "https://xapi.elearn.rwth-aachen.de/definitions/
             blockBasedProgramming/activities/block",
          "extensions": {
            "https://xapi.elearn.rwth-aachen.de/definitions/
               blockBasedProgramming/activities/blocktype": "https://
               xapi.elearn.rwth-aachen.de/definitions/
               blockBasedProgramming/activities/movement_block"
          }
        }
      }
    ],
    "grouping": [
      {
        "objectType": "Activity",
        "id": "https://xapi.elearn.rwth-aachen.de/definitions/
           blockBasedProgramming/activities/math_number-D-gbUPa,T(5Jyt
           [0^?5g",
        "definition": {
          "name": {
            "en-US": "math_number"
          },
          "type": "https://xapi.elearn.rwth-aachen.de/definitions/
             blockBasedProgramming/activities/block",
          "extensions": {
            "https://xapi.elearn.rwth-aachen.de/definitions/
               blockBasedProgramming/activities/blocktype": "https://
               xapi.elearn.rwth-aachen.de/definitions/
               blockBasedProgramming/activities/math_block"
          }
        }
      }
    ],
    "category": [
      {
        "objectType": "Activity",
        "id": "https://xapi.elearn.rwth-aachen.de/definitions/
           blockBasedProgramming/activities/sprite/430cf3ea-d80d-43a7-
           b88b-ac286ee691a5",
        "definition": {
          "name": {
            "en-US": "Cubi"
          },
          "type": "https://xapi.elearn.rwth-aachen.de/definitions/
             blockBasedProgramming/activities/sprite"
        }
      }
    ]
  }
```

```
}
```

In-depth sprite interaction analysis Sprite interaction reports provide a deeper understanding through contextual elements such as the sprite ID and its associated blocks. For example, when a user performs actions such as deleting a sprite, the report provides a comprehensive account of all associated block removals, providing a comprehensive perspective of the event.

Level interaction logging Cubi's logging doesn't stop at individual elements, but also includes user interaction with the entire level, including actions such as running the program. This monitoring of the number of executions helps to evaluate user compliance with instructional guidance and provides valuable insights into learning progress.

This comprehensive and multi-faceted approach to logging allows Cubi to effectively capture a wide range of user interactions. In the context of this thesis, it ensures that nuanced data analysis can be carried out, fostering a deeper understanding of user behaviour and its impact on the learning experience.

Summary This section addresses the first research question *[RQ1] What are loggable learning experiences for automated feedback generation in Cubi?*. In concludes that:

> Loggable learning experiences for automated feedback generation in Cubi include interactions such as block manipulation (e.g., moving or modifying blocks), code execution, and user activity with the platform, which are meticulously logged and categorised to enhance the learning process and provide data-driven feedback.

4.5. Feedback Types in Block-Based Programming

Within the domain of block-based programming, the search for effective feedback mechanisms is of profound importance in enhancing the learning process. This section is dedicated to addressing the second research question *[RQ2] What are the existing and potential feedback types in block-based programming?*, and performs an evaluation of the existing feedback mechanisms integrated into Cubi (see Table 4.3). It presents suggestions to enhance the learning experience. Additionally, this section classifies the feedback types for block-based programming.

4.5.1. Existing Feedback Mechanisms in Cubi

In the Cubi programming environment, a range of existing feedback mechanisms is designed to assist learners in their programming endeavors. This section examines the currently available feedback types within Cubi and evaluates their strengths and limitations.

Block categories and available blocks Cubi employs visual cues, such as block categories and available blocks, to guide learners. These cues inform users about the specific types of blocks required for the current level (see Figure 4.1). While this serves as a form of scaffolding, it should be noted that the presence of block categories may change as users progress through the course. (KTC-TPR)

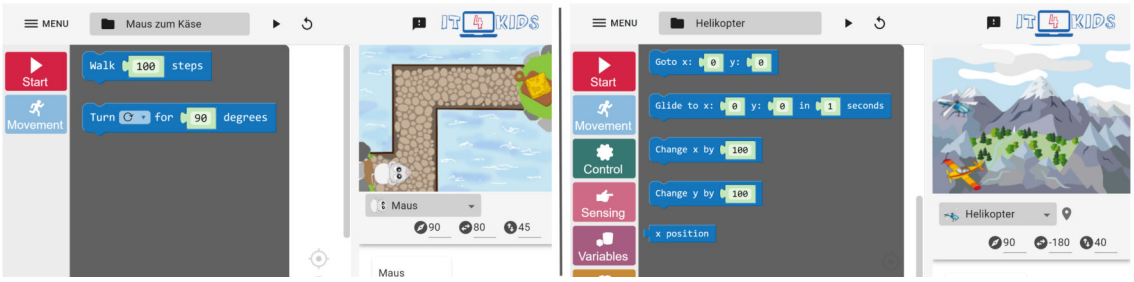


Figure 4.1.: Different categories of blocks availabe in Cubi depending on the current level. Left side: One of the first levels with only basic blocks. Right side: A more advanced level with a wider range of blocks.

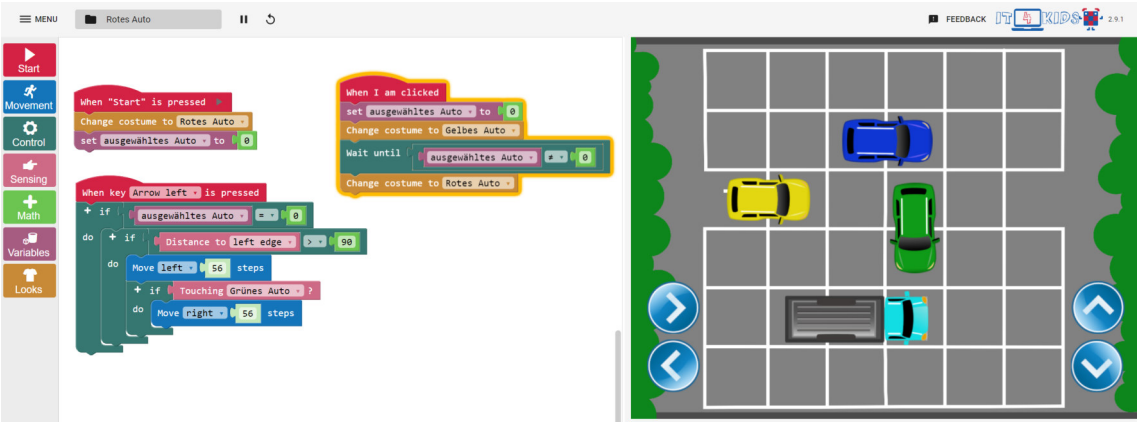


Figure 4.2.: When the Execute button is pressed, the code is executed and the sprite behaves as programmed. The blocks currently being executed are highlighted in yellow.

Play and reset buttons The "Play" and "Reset" buttons in Cubi provide an immediate real-time representation of the code's execution on the stage. This visual feedback allows learners to observe the behaviour of their code, aiding them in identifying errors and verifying correct functionality (see Figure 4.2). Additionally, level creators may have integrated dialogue explanations into the sprite, thus guaranteeing that when errors arise, learners receive accurate feedback. (KR, KH-EC, KTC-TR, KC-EXP)

Workspace arrangement In certain levels, Cubi pre-arranges blocks in the workspace, requiring learners to assemble them in the correct order. Importantly, students have the freedom to add new blocks to the workspace. This approach serves as a scaffold, helping users focus on the logic and sequencing of the blocks, rather than on syntactical correctness. These pre-arranged blocks primarily serve as guidance and hints, promoting a deeper understanding of the underlying programming concepts. (KTC-TPR)

Tutorials Cubi incorporates tutorials and textual subgoal descriptions for some levels. These tutorials provide step-by-step guidance, explaining the concepts underlying the task, as exemplified in Figure 4.3. Although not available for all levels, tutorials are designed to break down the solution process into manageable segments, enhancing learners' comprehension and problem-solving skills.(KTC-TR, KTC-TPR, KC-EXP, KC-EXA)

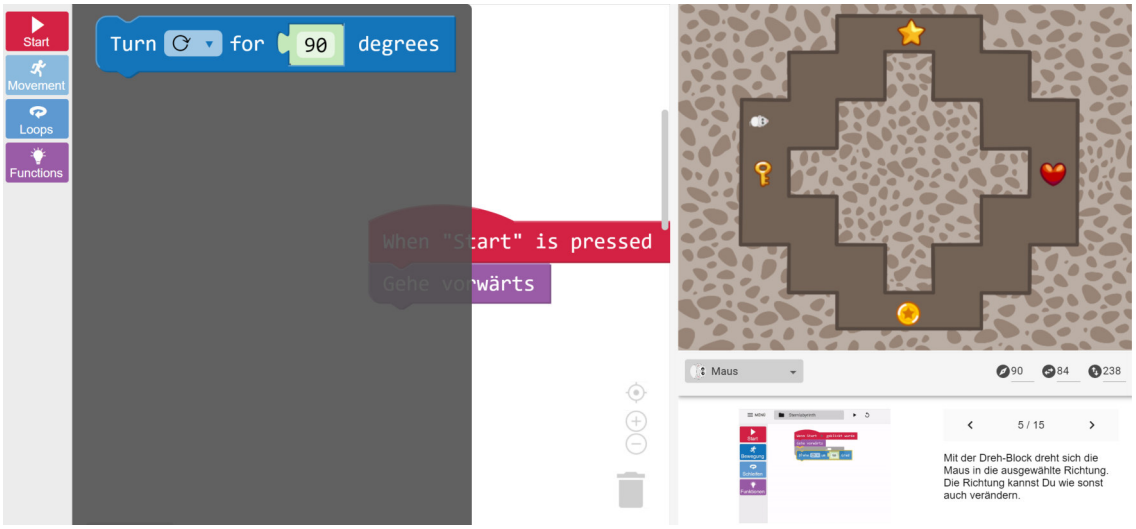


Figure 4.3.: Tutorials, located below the stage, explain the task and provide hints. The visually explanatory GIFs help users to understand the required actions.

				EF												
				KTC		KC		KM					KH			
Feedback Options	**KP**	**KR**	**KCR**	**TR**	**TPR**	**EXP**	**EXA**	**TF**	**CE**	**SE**	**SI**	**PI**	**EC**	**TPS**	**IM**	**KMC**
Available block categories					✓											
Available blocks					✓											
Block constraints									✓		✓					
execute-button		✓			✓	✓							✓			
Workspace arrangement					✓											
Tutorials				✓	✓	✓	✓									

Table 4.3.: Existing feedback mechanisms in Cubi.

Block constraints The block design within Cubi, and across all BBPLs, enforces syntactical correctness, ensuring that only compatible blocks can be connected (see Figure 2.2). This design constraint promotes error avoidance, as learners are guided towards constructing valid code structures. (KR, KTC-TPR)

Summary In summary, Cubi's existing feedback mechanisms provide a solid foundation for a supportive learning environment. However, there are numerous opportunities to enhance and expand these mechanisms to foster a more comprehensive and impactful learning experience. This thesis explores these opportunities in the next section.

4.5.2. Enhancing and Expanding Feedback Mechanisms in Cubi

The existing feedback mechanisms within Cubi form the basis of a supportive learning environment. However, the potential to further enhance and expand these mechanisms can foster a more comprehensive and impactful learning experience. This section explores potential ways to develop the feedback landscape in Cubi.

Task requirements In certain instances, learners may require access to the task description and objectives. Providing a description of the level task within Cubi can assist learners who have missed or need to revisit the task description and objectives, promoting better understanding and task completion. (KTC-TR)

Task completion feedback Cubi has a notable area for potential improvement, particularly in the area of providing task completion feedback. Currently, Cubi lacks feedback about correctness on submitted code, which represents a clear opportunity for improvement. The introduction of task completion feedback holds great promise for extending the platform's capabilities. Such feedback can act as a powerful motivator for users, encouraging them to strive for higher levels of performance. Cubi offers a sample solution for each task. Using this feature, one could extract the solution for a given task and compare it with the solution submitted by the user. This approach would not only provide learners with valuable insights, but would also enhance their learning experience. (KR)

Adaptive Feedback Integration It has been demonstrated that learning paths can be extracted from clickstream data [13]. By collecting data through xAPI statements, the platform can also identify these paths for its learners. This capability can be used to provide adaptive feedback. By leveraging user interactions and progress data, Cubi could dynamically customise feedback based on the unique needs and learning paths of individual learners. Such adaptive feedback has the potential to address the diversity of learners' skill levels and provide tailored instruction, thereby optimising learning outcomes [21].

Conceptual Feedback Mapping Mapping errors to underlying concepts and skills is a powerful approach to getting learners to think about their code conceptually, rather than just offering local, code-specific suggestions [19]. In Cubi, each block is a representation of a particular programming concept. If a user avoids using a specific block, this may indicate a gap in their understanding of that particular concept. To implement this, Cubi can provide users with detailed explanations or illustrative examples when they encounter errors related to specific blocks. For example, if a user misuses a particular block, the platform can provide contextual feedback that not only highlights the error, but also explains the underlying concept associated with that block. This approach enables learners not only to correct their errors, but also to gain a deeper understanding of the programming concepts they are working with. In essence, it turns errors into valuable learning opportunities, facilitating a more effective and conceptually grounded learning experience.

Error-Specific Guidance To enhance the learning experience in Cubi, the platform can adopt the strategy of providing error-specific guidance that is applicable to any error the user encounters. This approach goes beyond providing generic feedback and is particularly valuable when a user makes an error that leads to unexpected results or problems in the execution of their code. When a user gets stuck, Cubi's role is to accurately diagnose the nature of the error. Rather than leaving the user with a vague understanding that something is wrong, the platform identifies the specific type of error and breaks it down into its component parts. For example, it can identify problems related to incorrect code blocks, logical errors or syntax errors. Once the error has been identified, Cubi can provide tailored and actionable recommendations to fix it. This could include recommending the appropriate code block, providing guidance on how to correct logical errors, or suggesting syntax adjustments. The key is to provide the learner with the information and insight needed to effectively address the error. This approach prevents users from feeling stuck or frustrated when they encounter coding challenges, empowering them to learn from their mistakes and become more confident and skilled programmers [14].

Attention-Tailored Hints Based on the research conducted by Conati and Zhao [8], Cubi has the ability to implement attention-tailored hints to improve the learning experience for its users. This approach involves taking into account each user's disposition to receive hints, and tailoring the presentation of hints accordingly. This allows Cubi to create a more personalised and engaging learning environment. When a user encounters a challenge or needs assistance, Cubi can assess their preference for hints. Some users may prefer subtle hints, while others may benefit from more explicit guidance. By understanding these preferences, Cubi can dynamically adjust the delivery of hints to match each learner's unique learning style. For users who prefer subtle hints, Cubi can provide subtle cues that encourage independent problem solving and critical thinking. Conversely, for those who benefit from more explicit guidance, the platform can provide detailed and comprehensive hints to help them overcome obstacles.

Feedback analytics for teachers This concept is a powerful tool that could greatly benefit Cubi's educational ecosystem. It would provide instructors with valuable insights into learners' learning progress, going beyond traditional assessment and grading to provide a detailed analysis of common challenges and misconceptions [19]. This feature could include identifying patterns and trends in learners' errors, allowing teachers to tailor their teaching strategies to effectively address recurring issues. It could also prompt improvements to problematic levels or tasks and inform the creation of new educational content. Data-driven decision-making and a culture of continuous improvement would be key outcomes, ensuring that Cubi's learning environment evolves to meet learners' needs.

Task difficulty assessment Within the Cubi learning environment, task difficulty assessment is a key component of the educational process. This process involves an analysis of the complexity embedded in the programming tasks set for learners. By examining task complexity, educators can strike a delicate balance, ensuring that learners are presented with challenges that are both stimulating and manageable, thus preventing feelings of overwhelm. In addition, identifying task difficulty serves as a compass for selecting appropriate feedback mechanisms. In particular, as tasks vary in complexity, feedback strategies need to adapt accordingly. For example, more challenging tasks may require more complex and adaptive feedback mechanisms to guide learners effectively. By accurately measuring task complexity, educators can sequence assignments, ensuring a gradual increase in complexity as learners progress along their educational journey. In the future, Cubi could have the potential to use predictive algorithms to anticipate the difficulty of newly created levels, further enhancing the platform's adaptability and learning support.

Assessment of feedback effectiveness The question of whether the feedback provided within the Cubi platform was effective in helping learners is a critical one. It involves evaluating the impact of feedback mechanisms on learners learning outcomes. Evaluation in this area includes analysing whether the feedback helped learners to understand and correct their mistakes, improved their problem-solving skills and contributed to their overall programming skills. By collecting data on the effectiveness of feedback, Cubi can refine its feedback mechanisms to better meet learners' needs and preferences.

Feedback Category	Feedback Mechanism	Existing in Cubi
Immediate feedback	available blocks	✓
	available categories	✓
	play and reset button	✓
	tutorials	✓
	block constraints	✓
	task requirement	
	gamification	
On-demand feedback	task completion	
	personalised hints	
	positive feedback	
	progress tracking	
Delayed feedback	analytics for teachers	
	task difficulty	
	feedback effectiveness	
	learners engagement	

Table 4.4.: Categorisation of feedback mechanisms in Cubi.

Engagement and game interaction Monitoring whether the games created on the Cubi platform have been played by learners is an essential aspect of assessing engagement and learning. This involves tracking users' interactions with the games they have programmed. Understanding whether learners are actively engaging with their creations can provide insight into their level of interest and motivation. This data can be used to identify patterns in usage, such as which games are more appealing to learners or whether certain levels are frequently replayed. In addition, assessing game usage can help educators gauge the effectiveness of the platform in facilitating hands-on learning experiences and fostering creativity among learners.

4.6. Categorisation of Feedback for Block-Based Programming

This section deals with the categorisation of feedback mechanisms in Cubi. The various forms of feedback discussed in section 4.5 can be organised into three distinct categories within Cubi's pedagogical framework: *Immediate Feedback*, *On-Demand Feedback* and *Delayed Feedback*. This classification includes both existing feedback mechanisms in Cubi and potential additions. The categorisation is is shown in Table 4.4.

Immediate feedback Immediate feedback is delivered instantly, originating directly from Cubi and is independent of xAPI statements. It is primarily directed to the learner and is triggered by their actions within the platform, such as clicking a button or executing a program.

On-demand feedback On-demand feedback is generated by an Educational Data Mining (EDM) system (engine) and is based on users' xAPI statements. This type of feedback is addressed to each individual learner. It is triggered when the learner specifically requests it, for example when they ask for help.

Delayed feedback Delayed feedback is generated by the EDM system based on user statements collected over a 24-hour period. This feedback is intended for instructors and level coders. It serves several purposes, including helping tutors to refine their teaching methods or helping level coders to improve their designs. Essentially, it acts as a tool for continuous improvement within Cubi's educational ecosystem.

Summary This section addresses the second research question *[RQ2] What are the existing and potential feedback types in block-based programming?*. In concludes that:

> The utilisation of collected statements by Cubi holds the potential to enhance the learning experience by providing learners with more personalised feedback and teachers with valuable insights to improve their teaching methods. The feedback is categorised into immediate, on-demand and delayed feedback. This categorisation helps research efforts to evaluate the effectiveness of feedback and encourages continuous improvement of overall BBPLs' feedback mechanisms.

4.7. Summary

This chapter provides an overview of the research questions of the thesis: *[RQ1] What are loggable learning experiences for automated feedback generation in Cubi?* and *[RQ2] What are the existing and potential feedback types in block-based programming?*.

Section *Logging Learning Experiences* (4.2) focuses on the research question *[RQ1] What are loggable learning experiences for automated feedback generation in Cubi?*. This section provides a comprehensive overview of the logging process within Cubi, its theoretical and practical aspects, and the associated limitations. In addition, it discusses the criteria for identifying and logging meaningful interactions in Cubi, taking into account different user groups and the pedagogical relevance of interactions. Finally, a decision is made to prioritise the logging of individual blocks used in programming over the resulting code. This choice is based on several compelling reasons, including achieving granularity, gaining valuable pedagogical insights, compatibility with the Blockly framework, and support for different learning styles. *RQ1*

The next section *Privacy* addresses the challenge of striking a balance between privacy and analytic effectiveness in learning platforms such as Cubi. Achieving this balance is of paramount importance, as it protects users' personal information while enabling the skilled use of data to enrich personalised learning experiences and support comprehensive analytics.

The *Standardisation* section highlights the importance of adopting the xAPI standard and establishing a common vocabulary specific to BBPLs. This approach simplifies data analysis, facilitates data comparison, and promotes compatibility and interoperability across different platforms and systems.

The section *Feedback Types in Block-Based Programming* (4.5) explores the question *[RQ2] What are the existing and potential feedback types in block-based programming?*. It discusses the crucial role of effective feedback mechanisms in block-based programming. It evaluates the feedback mechanisms currently available and suggests improvements and extensions to feedback mechanisms. *RQ2*

Section 4.6 categorises the feedback mechanisms in Cubi into three distinct categories: *Immediate Feedback*, *On-Demand Feedback* and *Delayed Feedback*. This categorisation helps research efforts to evaluate the effectiveness of feedback and encourages continuous improvement of overall BBPLs' feedback mechanisms.

Chapter 5 Implementation

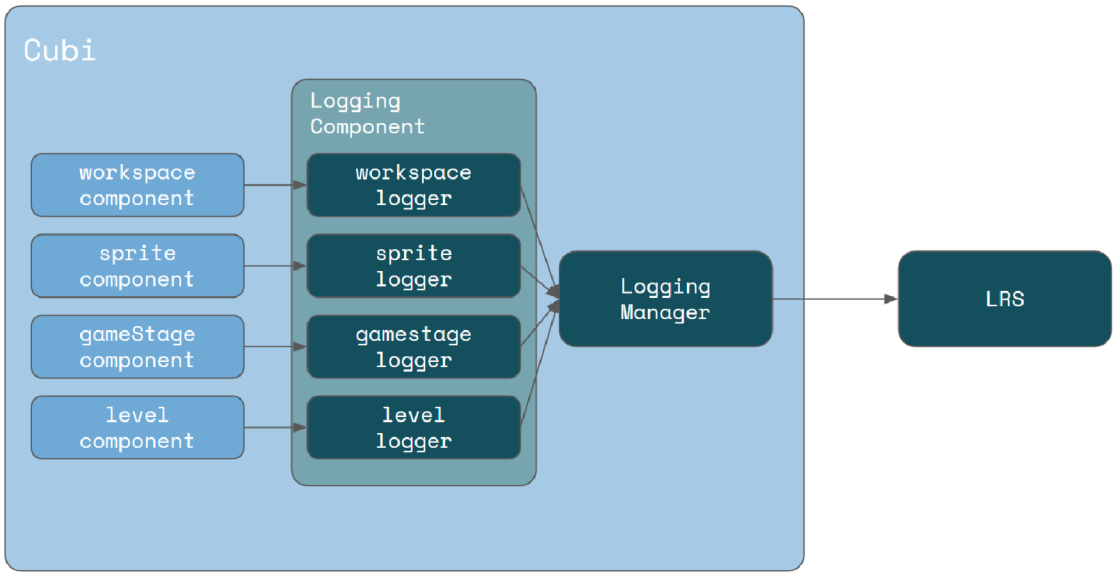


Figure 5.1.: A structural overview of the CubiXLI integrated into Cubi. Within Cubi, each component is paired with a corresponding logging component. The central role of the Logging Manager is to facilitate the transfer of statements to the Learning Record Store.

The structure of the CubiXLI is shown in the Figure 5.1. It consists of two main components: the `Logging Component` and the `Logging Manager`. Within this system, the `Logging Components` are responsible for capturing statements and, if necessary, grouping these statements before passing them to the `Logging Manager`. The `Logging Manager` is then responsible for adding contextual information to these statements before passing them on to the LRS.
The following sections provide an in-depth exploration of each of the component within this architectural framework and a presentation of the associated vocabulary.

5.1. Used Software

In developing CubiXLI, *TypeScript*[1] was chosen as the primary programming language. This choice is consistent with the underlying technology of Cubi, which is also implemented in TypeScript. TypeScript, a statically typed superset of *JavaScript*, compiles to standard JavaScript. The implementation also includes a suite of unit tests developed

[1] `https://www.typescriptlang.org/` *(Accessed: 12.09.2023)*

using the *Jest*[2] framework. In addition, library dependency management is facilitated by the *Yarn*[3] package manager.

5.2. The Logging Components

The logging components within Cubi provide basic logging capabilities, consisting of four different logging modules, each assigned to a specific aspect of Cubi. These logging capabilities are centred around the workspace, stage, level and sprite. A comprehensive list of all statements is detailed in Table 5.1.

workspace	
blockCreatedStatement	Logged when the user creates a block by dragging it out of the toolbar. Includes the accessory sprite.
blockLoadedStatement	Logged when existing blocks load in the workspace. Includes the accessory sprite.
blockMovedStatement	Logged when the user moved a block. Includes the accessory sprite, parent and child blocks.
blockChangedStatement	Logged when the user changes a block. Includes the accessory sprite, the old and new values.
blockDeletedStatement	Logged when the user deletes a block. Includes the accessory sprite and child blocks.
variableCreatedStatement	Logged when the user creates a variable. Includes the accessory sprite.
variableRenamedStatement	Logged when the user renames the variable. Includes the accessory sprite.
variableDeletedStatement	Logged when the user deletes the variable. Includes the accessory sprite.
level	
levelOpenedStatement	Logged when the user opens a level.
levelSavedStatement	Logged when the user saves a the level.
levelCreatedStatement	Logged when the user opens a blank level.
stage	
stageExecutedStatement	Logged when the user executes the code.
stagePausedStatement	Logged when the user pauses the code.
stageStoppedStatement	Logged when the user stops the code
sprite	
spriteCreatedStatement	Logged when the user creates a new sprite.
spriteSwitchedStatement	Logged when the user switches to another sprite.
spriteRenamedStatement	Logged when the user renames the sprite.
spriteDeletedStatement	Logged when the user deletes the sprite.
costumeCreatedStatement	Logged when the user creates a new costume for a sprite.
costumeSwitchedStatement	Logged when the user switches the costume of a sprite.
costumeDeletedStatement	Logged when the user deletes the costume of a sprite.

Table 5.1.: A comprehensive list of all statements logged by the logging components.

[2] `https://jestjs.io/` *(Accessed: 12.09.2023)*
[3] `https://yarnpkg.com/` *(Accessed: 12.09.2023)*

Workspace Logging This component plays a role in translating data generated within the Cubi workspace into structured xAPI statements, effectively documenting the user's journey through the programming concepts taught in Cubi.
The code's dexterity in handling events is demonstrated by its real-time interaction with Blockly events. It captures events such as 'CREATE', 'DELETE', 'CHANGE' and 'MOVE' and translates them into xAPI statements.
One notable aspect is the merging of statements, which improves the readability of logs. When several related interactions occur simultaneously, such as the modification of several blocks, the code merges them into one statement. This approach minimises redundancy and provides a concise overview of the user's actions.
In addition, the `StatementLogger` class faces the challenge of distinguishing between user-created blocks and loaded blocks, a nuance that Blockly does not inherently detect. To address this, the code examines whether a block has been moved; if so, it indicates a creation event, otherwise it indicates a loading event. In the case of a load event, the statement attributes the action to the admin actor.
The code implementation also incorporates error handling mechanisms to ensure the resilience of the logging process. Error management prevents disruption and maintains the integrity of the educational experience.

Stage Logging This component is responsible for documenting interactions during code execution. It captures the user's actions when running, pausing and stopping code events, providing a view of the user's engagement with the code execution process. This functionality enables a thorough understanding of how users interact with the code and how they manage its execution.

Level Logging This component is responsible for recording interactions associated with different levels. It logs user actions when they open, save or create new levels, providing insight into how users engage with different levels. This capability enables an understanding of how users interact with various levels and their approach to managing their progress on a session basis.

Sprite Logging This component is responsible for logging user interactions with the sprites in Cubi. It records user actions when creating, switching, renaming, and deleting sprites. It also records the creation, switching, and deletion of costumes. This functionality allows for a thorough understanding of how users interact with the sprites and their costumes during the programming process and while playing the games they create.

5.3. The Logging Manager

The `Logging Manager` plays a pivotal role in overseeing the logging process, facilitating the transmission of xAPI statements to a designated endpoint, thereby enabling the capture of significant interactions and activities within the Cubi platform. The `Logging Manager` consists of three integral classes: the `StatementLogger`, `ContextLogger` and `XAPIManager`.
The `ContextLogger` takes responsibility for constructing contextual information. It assembles various contextual elements such as session identifiers, platform-related details, relationships between blocks, and accessory sprites. These context objects enrich the xAPI statements, providing valuable insight for subsequent analysis.

The `XAPIManager` connects to the specified LRS and sends the fully-formed xAPI statement to the endpoint.

StatementLogger The `StatementLogger` is responsible for creating complete xAPI statements. It takes partial statements generated by the `Logging Components` and enriches them with actor information to make them complete and coherent. To make these records more informative, the `StatementLogger` can differentiate between different types of users, such as regular users and administrators, based on certain distinctions. After gathering all the relevant information, the `StatementLogger` compiles it into a complete statement of a user's interaction on Cubi.

ContextLogger The `ContextLogger` module is responsible for generating rich contextual information to be appended to xAPI statements during user interactions within Cubi. This includes creating unique session identifiers, capturing platform-related details, representing various elements within the Cubi environment, and facilitating relationships between blocks and sprite-related activities. It also records data about deleted blocks and their relationships, increasing the depth of analysis when examining user actions related to block removal. These contextual objects enrich the xAPI statements with invaluable insights for subsequent analysis.

XAPIManager The `XAPIManager` acts as the bridge connecting the Cubi platform and the LRS. It takes the complete xAPI statement and securely transmits it to the designated endpoint within the LRS. This process ensures that valuable data about user interactions in Cubi is safely and accurately stored for later analysis and tracking.

5.4. Vocabulary

In the context of the Cubi programming environment, a dedicated vocabulary plays a key role in facilitating the generation of xAPI statements and ensuring consistency in the representation of user interactions. This vocabulary module, integrated into Cubi, allows the conversion of Cubi's internal TypeScript objects into structured xAPI objects. This conversion process simplifies the transformation of various Cubi elements, including but not limited to programming projects and activities, into coherent xAPI objects.
This separation of logging capabilities and vocabulary within the Cubi architecture offers distinct advantages. It allows the consistent use of the same vocabulary and helper functions across different components and extensions within Cubi. This promotes consistency in the representation of user actions and improves the overall comprehensibility of the logged data. As a result, the vocabulary module can evolve separately, ensuring its adaptability to future enhancements and changes within the Cubi programming environment.

5.5. Summary

This implementation chapter describes the architecture of CubiXLI. This interface consists of the logging component and manager, which work together to capture, structure and submit xAPI statements to the LRS.
The XAPIManager class is central to the logging management. It manages the submission of xAPI statements to the LRS, recording user interactions within Cubi.
Logging components in Cubi focus on the workspace, stage, level and sprite, providing

detailed tracking of user interactions.
The Vocabulary module translates internal TypeScript objects into xAPI-compliant statements, ensuring consistency in the representation of user actions. The separation of logging and vocabulary supports modularity.

Chapter 6 Evaluation

The evaluation in this chapter focuses on assessing the performance of CubiXLI. The primary goal is to ensure that CubiXLI works seamlessly without disrupting the user's workflow. It does work invisibly in the background, requiring no user interaction. However, it is important to consider the potential impact on the user experience, such as loading and response times. Specifically, the additional time CubiXLI may add to the website loading process before users can interact with Cubi, and any delays in user interactions. Longer loading times can have a negative impact on the overall user experience, as reported in a study [11]. In particular, perceived waiting time is subjective and depends on user expectations, with a logarithmic relationship between time and quality of user experience.
A similar consideration applies to execution time. Statement logging is asynchronous with other functionality, so the time for events such as program execution is not affected by the logging interface. However, it is important to evaluate the potential impact on the time it takes for users to interact with Cubi. This evaluation is done using the Time To Interact TTI metric, which is discussed in more detail in the section 6.2.

6.1. Setup

The evaluations are based on the specific setups described in table 6.1. The computer running the browser and accessing Cubi, will be referred as *localhost*.

CPU	Intel(R) Core(TM) i5-9300H CPU @ 2.40GHz
RAM	8,00 GB
Storage	1TB NVMe Micron_2200_MTFD
Operating System	Windows 11 Home 22H2
Browser	Google ChromeVersion 116.0.5845.188 (Official Build) (64-bit)
Cubi	3.0.0-alpha-dev

Table 6.1.: Specs of the localhost used for the evaluation

6.2. Time to Interact

Time To Interact (TTI) is a critical metric that measures the time elapsed between a user accessing a website and the point at which they can actively interact with the website. Importantly, at TTI all basic functionality should be available to the user. The TTI is particularly important in this context as it has a direct impact on the user experience, which can be worsened by long loading times [11].

Calculating TTI TTI is calculated by measuring the time between when the user accesses Cubi and when Cubi resolves the "sync workspace with level" operation.

Differences TTI is measured both with and without the presence of CubiXLI The difference between these two measurements reflects the impact of CubiXLI on TTI.

Results The results of the TTI evaluation (see Figure 6.1) show that the logging interface has a minimal effect on TTI. Furthermore, the comparison with the TTI of the baseline (no CubiXLI) is also negligible, indicating that the presence of CubiXLI should not significantly affect the user experience.

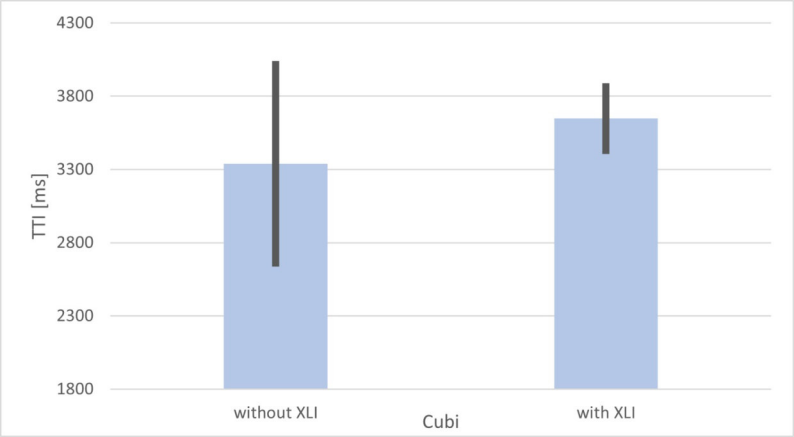


Figure 6.1.: TTI of Cubi with and without CubiXLI

6.3. Summary

The evaluation considered factors such as user experience, loading times and interaction delays. It was conducted using the (TTI) metric, which measures the time from user access to full website interaction. The results showed that the login interface had only a small impact on TTI, suggesting that it should not have a significant impact on the overall user experience. This evaluation provides valuable insights into the effectiveness of CubiXLI and its compatibility with a seamless user experience.

Chapter 7 Discussion & Future Work

This thesis presents two research questions:

RQ1 What are loggable learning experiences for automated feedback generation in Cubi?

RQ2 What are the existing and potential feedback types in block-based programming?

In order to effectively generate valuable feedback for users, the foundation lies in the collection of high quality data. This essential data collection process formed a central aspect of the research methodology of this thesis. The first research question involved a careful examination of the learning experiences that could be captured within Cubi. The purpose of this investigation was twofold: firstly, to determine the extent of data that could be captured by CubiXLI, and secondly, to determine the meaningfulness of this data in the context of feedback generation. In addition, it addressed the question: whether the data captured by CubiXLI possessed the necessary qualities to underpin effective feedback mechanisms. In order to identify useful feedback strategies, the research conducted a comprehensive review of related work and carefully considered the possibilities inherent in the data that could be logged. In particular, the lack of a standardised approach to data handling across different BBPLs led to the adoption of the xAPI standard as the basic framework for Cubi. A key decision was whether to log blocks or as generated JavaScript code. The rationale for choosing blocks stemmed from their clarity in illustrating the sequential steps learners take while solving a problem. Furthermore, Blockly has the ability to convert these blocks into JavaScript code when necessary, ensuring flexibility for future use. However, it is important to acknowledge certain limitations that were encountered during this research endeavour. The new version of Cubi, the chosen platform, was still in development at the time of writing. As a result, aspects such as game interactions were not recorded due to their absence in the current state of the software. Furthermore, the lack of user accounts within Cubi posed a challenge, limiting the ability to track user progress over time.
The second research question explored the types of feedback that are potentially valuable in the context of block-based programming environments. Through a thorough review of the existing literature and an analysis of the possibilities offered by loggable data, new forms of useful feedback were proposed. These forms of feedback offer a departure from the dominant error-centric approach and aim to provide more constructive and encouraging feedback to users. It should be noted, however, that these proposed feedback types have not been empirically tested with users and their applicability may vary between different block-based programming languages. In addition, certain feedback types are specifically tailored to Cubi, taking advantage of its unique features. Furthermore, it provides a categorisation of the feedback types that can serve as a valuable resource for

researchers and educators in the field.

Future user studies represent a promising avenue for future work in the context of Cubi and its logging interface. Conducting in-depth research involving learners, teachers and administrators can provide invaluable insights into the effectiveness of the platform's feedback mechanisms. Future studies could explore the preferences and learning styles of different user groups and how personalised feedback can improve the learning experience. In addition, longitudinal studies that track student progress over time can provide a comprehensive understanding of skill development in block-based programming. These user-centred studies have the potential not only to refine the recording process, but also to contribute to the wider discourse on effective computing education in primary schools.

Appendix A Bibliography

[1] Dlzar Al Kez, Aoife M Foley, David Laverty, Dylan Furszyfer Del Rio, and Benjamin Sovacool. Exploring the sustainability challenges facing digitalization and internet data centers. *Journal of Cleaner Production*, 371:133633, 2022.

[2] Vincent Aleven, Ido Roll, Bruce M McLaren, and Kenneth R Koedinger. Help helps, but only so much: Research on help seeking with intelligent tutoring systems. *International Journal of Artificial Intelligence in Education*, 26:205–223, 2016.

[3] Elaf Abu Amrieh, Thair Hamtini, and Ibrahim Aljarah. Preprocessing and analyzing educational data set using x-api for improving student's performance. In *2015 IEEE Jordan conference on applied electrical engineering and computing technologies (AEECT)*, pages 1–5. IEEE, 2015.

[4] Brett A Becker, Graham Glanville, Ricardo Iwashima, Claire McDonnell, Kyle Goslin, and Catherine Mooney. Effective compiler error message enhancement for novice programming students. *Computer Science Education*, 26(2-3):148–175, 2016.

[5] Bitkom. Trotz Krieg und Krisen: In Deutschland fehlen 137.000 IT-Fachkräfte. `https://www.bitkom.org/Presse/Presseinformation/Deutschland-fehlen-137000-IT-Fachkraefte`. Accessed: 2023-08-23.

[6] Annabell Brocker, Sven Judel, Ulrik Schroeder, and Yanik Söltzer. Juxl: Jupyterlab xapi logging interface. In *2022 International Conference on Advanced Learning Technologies (ICALT)*, pages 158–160. IEEE, 2022.

[7] Samuel PM Choi, Sze Sing Lam, Kam Cheong Li, and Billy TM Wong. Learning analytics at low cost: At-risk student prediction with clicker data and systematic proactive interventions. *Journal of Educational Technology & Society*, 21(2):273–290, 2018.

[8] Cristina Conati, Natasha Jaques, and Mary Muir. Understanding attention to adaptive hints in educational games: an eye-tracking study. *International Journal of Artificial Intelligence in Education*, 23:136–161, 2013.

[9] Adina Deiner, Patric Feldmeier, Gordon Fraser, Sebastian Schweikl, and Wengran Wang. Automated test generation for scratch programs. *Empirical Software Engineering*, 28(3):79, 2023.

[10] Yihuan Dong, Preya Shabrina, Samiha Marwan, and Tiffany Barnes. You really need help: Exploring expert reasons for intervention during block-based programming assignments. In *Proceedings of the 17th ACM Conference on International Computing Education Research*, ICER 2021, pages 334–346, New York, NY, USA, 2021. Association for Computing Machinery.

[11] Sebastian Egger, Tobias Hossfeld, Raimund Schatz, and Markus Fiedler. Waiting times in quality of experience for web based services. In *2012 Fourth International Workshop on Quality of Multimedia Experience*, pages 86–96. IEEE, 2012.

[12] Benedikt Fein, Florian Obermüller, and Gordon Fraser. Catnip: An automated hint generation tool for scratch. In *Proceedings of the 27th ACM Conference on on Innovation and Technology in Computer Science Education Vol. 1*, pages 124–130, 2022.

[13] Daniel Amo Filvà, Marc Alier Forment, Francisco José García-Peñalvo, David Fonseca Escudero, and María José Casañ. Clickstream for learning analytics to assess students' behavior with scratch. *Future Generation Computer Systems*, 93:673–686, 2019.

[14] Shalom M Fisch. Making educational computer games" educational". In *Proceedings of the 2005 conference on Interaction design and children*, pages 56–61, 2005.

[15] Christoph Frädrich, Florian Obermüller, Nina Körber, Ute Heuer, and Gordon Fraser. Common bugs in scratch programs. In *Proceedings of the 2020 ACM conference on innovation and technology in computer science education*, pages 89–95, 2020.

[16] Celina Friemel, Stefan Morana, Jella Pfeiffer, and Alexander Maedche. On the role of users' cognitive-affective states for user assistance invocation. In *Information Systems and Neuroscience: Gmunden Retreat on NeuroIS 2017*, pages 37–46. Springer, 2018.

[17] The Octalysis Group. KHAN ACADEMY: ELEARNING GAMIFICATION THROUGH AN OCTALYSIS LENS. `https://octalysisgroup.com/2015/09/khan-academy-elearning-gamification-through-an-octalysis-lens/`. Accessed: 2023-09-13.

[18] Marcelo Guerra Hahn, Silvia Margarita Baldiris Navarro, Luis De La Fuente Valentín, and Daniel Burgos. A systematic review of the effects of automatic scoring and automatic feedback in educational settings. *IEEE Access*, 9:108190–108198, 2021.

[19] Georgiana Haldeman, Andrew Tjang, Monica Babeş-Vroman, Stephen Bartos, Jay Shah, Danielle Yucht, and Thu D Nguyen. Providing meaningful feedback for autograding of programming assignments. In *Proceedings of the 49th ACM Technical Symposium on Computer Science Education*, pages 278–283, 2018.

[20] Qiang Hao, David H Smith IV, Lu Ding, Amy Ko, Camille Ottaway, Jack Wilson, Kai H Arakawa, Alistair Turcan, Timothy Poehlman, and Tyler Greer. Towards understanding the effective design of automated formative feedback for programming assignments. *Computer Science Education*, 32(1):105–127, 2022.

[21] Hao He, Yulin Zhu, and Wei Cai. An adaptive hint system for puzzle games: A multimodal-based approach. In *2022 IEEE Games, Entertainment, Media Conference (GEM)*, pages 1–6. IEEE, 2022.

[22] Petri Ihantola, Arto Vihavainen, Alireza Ahadi, Matthew Butler, Jürgen Börstler, Stephen H Edwards, Essi Isohanni, Ari Korhonen, Andrew Petersen, Kelly Rivers, et al. Educational data mining and learning analytics in programming: Literature review and case studies. *Proceedings of the 2015 ITiCSE on Working Group Reports*, pages 41–63, 2015.

[23] IT4Kids. Wirkung von IT4Kids. https://it-for-kids.org/wirkung/. Accessed: 2023-08-23.

[24] IT4Kids. Cubi editor . https://editor.it-for-kids.org/. Accessed: 2023-08-21.

[25] IT4Kids. IT4Kids It-kurse für grundschulen. https://it-for-kids.org/. Accessed: 2023-03-15.

[26] Johan Jeuring, Hieke Keuning, Samiha Marwan, Dennis Bouvier, Cruz Izu, Natalie Kiesler, Teemu Lehtinen, Dominic Lohr, Andrew Peterson, and Sami Sarsa. Towards giving timely formative feedback and hints to novice programmers. In *Proceedings of the 2022 Working Group Reports on Innovation and Technology in Computer Science Education*, pages 95–115. Association for Computing Machinery, 2022.

[27] Bo Jiang, Wei Zhao, Nuan Zhang, and Feiyue Qiu. Programming trajectories analytics in block-based programming language learning. *Interactive Learning Environments*, 30(1):113–126, 2022.

[28] Hieke Keuning, Bastiaan Heeren, and Johan Jeuring. Strategy-based feedback in a programming tutor. In *Proceedings of the Computer Science Education Research Conference*, pages 43–54, 2014.

[29] Hieke Keuning, Johan Jeuring, and Bastiaan Heeren. A systematic literature review of automated feedback generation for programming exercises. *ACM Transactions on Computing Education (TOCE)*, 19(1):1–43, 2018.

[30] Natalie Kiesler. An exploratory analysis of feedback types used in online coding exercises, 2022.

[31] Ekaterina Kochmar, Dung Do Vu, Robert Belfer, Varun Gupta, Iulian Vlad Serban, and Joelle Pineau. Automated personalized feedback improves learning gains in an intelligent tutoring system. In *Artificial Intelligence in Education: 21st International Conference, AIED 2020, Ifrane, Morocco, July 6–10, 2020, Proceedings, Part II 21*, pages 140–146. Springer, 2020.

[32] Michael J Lee and Joseph Chiou. Animated hints help novices complete more levels in an educational programming game. *Journal of computing sciences in colleges*, 35(8), 2020.

[33] Samiha Marwan, Ge Gao, Susan Fisk, Thomas W Price, and Tiffany Barnes. Adaptive immediate feedback can improve novice programming engagement and intention to persist in computer science. In *Proceedings of the 2020 ACM conference on international computing education research*, pages 194–203, 2020.

[34] Samiha Marwan, Nicholas Lytle, Joseph Jay Williams, and Thomas Price. The impact of adding textual explanations to next-step hints in a novice programming environment. In *Proceedings of the 2019 ACM conference on innovation and technology in computer science education*, pages 520–526, 2019.

[35] Fahmid Morshed Fahid, Xiaoyi Tian, Andrew Emerson, Joseph B. Wiggins, Dolly Bounajim, Andy Smith, Eric Wiebe, Bradford Mott, Kristy Elizabeth Boyer, and James Lester. Progression trajectory-based student modeling for novice block-based programming. In *Proceedings of the 29th ACM Conference on User Modeling, Adaptation and Personalization*, pages 189–200, 2021.

[36] Susanne Narciss. Feedback strategies for interactive learning tasks. In *Handbook of research on educational communications and technology*, pages 125–143. Routledge, 2008.

[37] Azer Nouira, Lilia Cheniti-Belcadhi, and Rafik Braham. An enhanced xapi data model supporting assessment analytics. *Procedia Computer Science*, 126:566–575, 2018.

[38] Florian Obermüller, Lena Bloch, Luisa Greifenstein, Ute Heuer, and Gordon Fraser. Code perfumes: reporting good code to encourage learners. In *The 16th workshop in primary and secondary computing education*, pages 1–10, 2021.

[39] Florian Obermüller, Ute Heuer, and Gordon Fraser. Guiding next-step hint generation using automated tests. In *Proceedings of the 26th ACM Conference on Innovation and Technology in Computer Science Education V. 1*, pages 220–226, 2021.

[40] Benjamin Paaßen, Jessica McBroom, Bryn Jeffries, Irena Koprinska, and Kalina Yacef. Next steps for next-step hints: Lessons learned from teacher evaluations of automatic programming hints. In *Joint Proceedings of the Workshops of the 14th International Conference on Educational Data Mining (EDM)*, 2021.

[41] Abelardo Pardo and George Siemens. Ethical and privacy principles for learning analytics. *British journal of educational technology*, 45(3):438–450, 2014.

[42] Great Schools Partnership. The glossary of education reform. `https://www.edglossary.org/learning-experience/`. Accessed: 2023-08-12.

[43] PostCurious. What makes a good hint system? `https://www.getpostcurious.com/post/what-makes-a-good-hint-system`. Accessed: 2023-08-11.

[44] Thomas Price, Rui Zhi, and Tiffany Barnes. Evaluation of a data-driven feedback algorithm for open-ended programming. *International Educational Data Mining Society*, 2017.

[45] Thomas W Price, Samiha Marwan, Michael Winters, and Joseph Jay Williams. An evaluation of data-driven programming hints in a classroom setting. In *Artificial Intelligence in Education: 21st International Conference, AIED 2020, Ifrane, Morocco, July 6–10, 2020, Proceedings, Part II 21*, pages 246–251. Springer, 2020.

[46] Nadezhda Purtova. Do property rights in personal data make sense after the big data turn?: Individual control and transparency. *Individual Control and Transparency (November 13, 2017). N Purtova'Do property rights in personal data make sense after the Big Data turn*, 2017.

[47] Preya Shabrina, Samiha Marwan, Andrew Bennison, Min Chi, Thomas Price, and Tiffany Barnes. A multicriteria evaluation for data-driven programming feedback systems: Accuracy, effectiveness, fallibility, and students' response. *arXiv preprint arXiv:2208.05326*, 2022.

[48] Dalia Abdulkareem Shafiq, Mohsen Marjani, Riyaz Ahamed Ariyaluran Habeeb, and David Asirvatham. Student retention using educational data mining and predictive analytics: a systematic literature review. *IEEE Access*, 2022.

[49] Papadakis Stamatios. Can preschoolers learn computational thinking and coding skills with scratchjr? a systematic literature review. *International Journal of Educational Reform*, page 10567879221076077, 2022.

[50] Raw Studio. Case study: How Duolingo Utilises Gamification to Increase User Interest. https://raw.studio/blog/how-duolingo-utilises-gamification/. Accessed: 2023-09-13.

[51] Dan Sun, Fan Xu, and Yan Li. Using learning analytics in understanding students' behavior in block-based and text-based programming modality. In *2021 Tenth International Conference of Educational Innovation through Technology (EITT)*, pages 271–275. IEEE, 2021.

[52] Yanik Söltzer. An xAPI logging interface for JupyterLab with a focus on gamification. Masterarbeit, RWTH Aachen University, Aachen, 2022. Veröffentlicht auf dem Publikationsserver der RWTH Aachen University; Masterarbeit, RWTH Aachen University, 2021.

[53] Kurt VanLehn. The relative effectiveness of human tutoring, intelligent tutoring systems, and other tutoring systems. *Educational psychologist*, 46(4):197–221, 2011.

[54] Christiane Gresse von Wangenheim, Jean Carlo Rossa Hauck, Matheus Faustino Demetrio, Rafael Pelle, Nathalia da Cruz Alves, Heliziane Barbosa, and Luiz Azevedo. Codemaster - automatic assessment and grading of app inventor and snap! programs. *Informatics Educ.*, 17:117–150, 2018.

[55] David Weintrop and Uri Wilensky. To block or not to block, that is the question: students' perceptions of blocks-based programming. In *Proceedings of the 14th international conference on interaction design and children*, pages 199–208, 2015.

[56] David Weintrop and Uri Wilensky. Comparing block-based and text-based programming in high school computer science classrooms. *ACM Transactions on Computing Education (TOCE)*, 18(1):1–25, 2017.

[57] Ting-Ting Wu, Chia-Ju Lin, Shih-Cheng Wang, and Yueh-Min Huang. Tracking visual programming language-based learning progress for computational thinking education. *Sustainability*, 15(3):1983, 2023.

[58] xAPI Spec Working Group. xapi-spec. https://github.com/adlnet/xAPI-Spec/blob/master/xAPI-About.md. Accessed: 2023-08-07.

Appendix B Glossary

AST	Abstract Syntax Tree.
BBPL	Block-Based Programming language.
CE	Compiler errors.
EC	Bug-related hints for error correction.
EDM	Educational Data Mining.
EXA	Examples illustrating concepts.
EXP	Explanations on subject matter.
IM	Improvements.
KC	Knowledge about Concepts.
KCR	Knowledge of the Correct Result.
KH	Knowledge about how to proceed.
KM	Knowledge about mistakes.
KMC	Knowledge about meta-cognition.
KP	Knowledge of Performance.
KR	Knowledge of Result.
KTC	Knowledge about Task Constraints.
LA	Learning Anayltics.
LRS	Learning Record Store.
PI	Performance issues.
PII	Personally Identifiable Information.
SE	Solution errors.
SI	Style issues.
TF	Test failures.
TPR	Hints on task-processing rules.
TPS	Task-processing steps.
TR	Hints on task requirements.
TTI	Time To Interact.

VPL A programming paradigm where code is constructed using visual blocks or graphical elements instead of traditional text-based code.

xAPI experience Application Programming Interface.

Eidesstattliche Versicherung

Gušić, Lejla	365549
Name, Vorname	Matrikelnummer (freiwillige Angabe)

Ich versichere hiermit an Eides Statt, dass ich die vorliegende Arbeit/Bachelorarbeit/ Masterarbeit* mit dem Titel

Enabling Feedback for Block-Based Programming through the Development of an xAPI Logging Interface for Cubi

selbstständig und ohne unzulässige fremde Hilfe (insbes. akademisches Ghostwriting) erbracht habe. Ich habe keine anderen als die angegebenen Quellen und Hilfsmittel benutzt. Für den Fall, dass die Arbeit zusätzlich auf einem Datenträger eingereicht wird, erkläre ich, dass die schriftliche und die elektronische Form vollständig übereinstimmen. Die Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.

Aachen , September 27, 2023	
Ort, Datum	Unterschrift

*Nichtzutreffendes bitte streichen

Belehrung:

§ 156 StGB: Falsche Versicherung an Eides Statt
Wer vor einer zur Abnahme einer Versicherung an Eides Statt zuständigen Behörde eine solche Versicherung falsch abgibt oder unter Berufung auf eine solche Versicherung falsch aussagt, wird mit Freiheitsstrafe bis zu drei Jahren oder mit Geldstrafe bestraft.

§ 161 StGB: Fahrlässiger Falscheid; fahrlässige falsche Versicherung an Eides Statt
(1) Wenn eine der in den §§ 154 bis 156 bezeichneten Handlungen aus Fahrlässigkeit begangen worden ist, so tritt Freiheitsstrafe bis zu einem Jahr oder Geldstrafe ein.
(2) Straflosigkeit tritt ein, wenn der Täter die falsche Angabe rechtzeitig berichtigt. Die Vorschriften des § 158 Abs. 2 und 3 gelten entsprechend.

Die vorstehende Belehrung habe ich zur Kenntnis genommen:

Aachen , September 27, 2023	
Ort, Datum	Unterschrift

Acknowledgement

Firstly, I would like to express my sincere thanks to Prof. Dr. Ulrik Schroeder and Prof. Dr. Jürgen Giesl for for examining this thesis. Your feedback and evaluation are greatly appreciated. I'm very grateful to my supervisor, Annabell Brocker, for her excellent guidance and constant support during my academic journey. I want to express my sincere thanks to the whole IT4Kids team for their valuable discussions and feedback. Special thanks to Johanna, Chris, and Fynn for their support and guidance throughout this thesis. Finally, I want to thank my family and friends who supported me throughout this thesis. A special shoutout goes to David, Maged, Ronja, and Nina for their valuable advice and unwavering encouragement.