Early Identification of Students at Risk using Affective Surveys in CS1 Labs

Gareth Sears

School of Computing Science
Sir Alwyn Williams Building
University of Glasgow
G12 8QQ

A dissertation presented in part fulfilment of the requirements of the Degree of Master of Science at The University of Glasgow

12 October 2020

Abstract

Delayed interventions and a lack of confidence in faculty support are major influences on CS1 student attrition and disengagement. Consequently, considerable research has been conducted into developing software early warning systems which can rapidly identify students at risk and assist instructors with providing student support. However, these systems have received criticism for their opacity and diminishing of student agency. Researchers have also highlighted apophenia concerning the data sources driving such systems. This report documents the development and evaluation of a dual dashboard early warning system - *Check In* - which aims to address such criticisms. It uses students' affective state as a measure of risk. Affective state has been shown to influence CS1 performance and specific states such as frustration have been linked to attrition. The software administers affective surveys to students using a series of 2-dimensional grids, building on previous research. Students then complete a qualitative journal entry. Risk weights are assigned to students based on sentiment analysis of the journal response and whether student's 2-dimensional grid responses fall within configurable 'danger zone' bounds. The early warning system automatically flags students as needing support if the responses' combined risk weights exceed configurable thresholds. Additionally, flags can be assigned manually, either by instructors or by students themselves. An initial evaluation indicates the software shows potential. It was positively received by students and instructors recognised its utility, though some were discouraged by its complexity. The report concludes with suggestions to rectify the software's shortcomings and directions for further work.

Keywords: CS1, student support, educational technology, sentiment analysis, early warning system, retention, higher education, survey instrument, student emotions

Education Use Consent

I hereby give my permission for this project to be shown to other University of Glasgow students and to be distributed in an electronic format. **Please note that you are under no obligation to sign this declaration, but doing so would help future students.**

Name: Gareth Sears Signature: Gareth Sears

Acknowledgements

I would like to thank Dr Mireilla Bikanga Ada for her truly invaluable supervision, guidance, and emotional support throughout the project.

Many thanks to Glynn and Mitchell at Backbeat Technologies for helping me grow as a programmer and for encouraging me to break out of my comfort zone by using a completely new technology stack.

Also huge thanks to my friends and family for their patience and support. Especially to Lizzie, whose strength and love have spurred me throughout, and Cookie the cat, for grounding me when things got tough and diligently keeping my lap warm while I wrote the report.

Contents

Chapter 1

Introduction

1.1 Project Motivation

The current demand for computer science graduates cannot be overstated. According to the UK Commission for Employment and Skills, information technology is expected to make the largest contribution toward the UK's economic growth in the near future [64]. Software developers are the most desired of all STEM professions in the UK labour market [65] and there is also a desperate need for CS graduates to meet staffing deficits in teaching and postgraduate research [58]. However, there is a bottleneck in meeting this demand. According to the Higher Education Statistics Agency, computer science has the highest first degree non-continuation rate in the UK when compared to other fields of study [20]. Indeed, the dropout rate of CS students and engineering is well documented in the literature, particularly with respect to the first 2 years of study [21, 40, 7, 41]. Considerable research has been undertaken to identify which factors influence a student's decision to remain in CS1 [44]. Though the reasons for non-retention are complex and multifaceted, common themes have emerged when interviewing departing students. These include a perceived lack of student-to-faculty interaction [18, 44], delays in identifying students at risk, and failures to intervene at critical junctures during the course [46, 44]. CS educators and researchers generally agree on the importance of early intervention to support students at risk [43] and accordingly there has been an interest in developing software early warning systems (EWS) to support instructors to this end.

EWS are a subset of the broader field of learning analytics (LA), which concerns "the measurement, collection, analysis and reporting of data about learners and their contexts, for purposes of understanding and optimizing learning and the environments in which it occurs" [60]. EWS use input data to anticipate negative outcomes at an early enough stage so measures can be taken to avoid or mitigate them. They are also required to present their results to stakeholders in an easily understood manner, for example, via a dashboard interface [30]. This rapid identification of students at risk is paramount in CS1. Students often only realise they have fallen behind when they receive a substantial assignment or a failing mark, and may only seek help until at a stage where interventions are unable to make significant difference [44] .

The data sources used to drive such systems varies. Learning analytics research has revealed that "virtual learning environment (VLE) data, assessment scores, institutional variables, differences in technology use, students' online engagement and course design and usage can predict, to various degrees, student retention" [2]. However, the vast majority of research has tended toward using logs of computer-mediated interactions [53]. Some studies have called into doubt the correlation between logs and performance in face-to-face taught computer science courses [3], suggesting there is merit in exploring other avenues. Student affect and its influence on attrition and performance have recently gained traction in CS education research. Therefore, affective data may offer such an avenue, either as an alternative or a supplement to existing EWS data sources.

1.2 Project Aims and Objectives

This project aims to develop a software tool, *Check In*, which administers affective surveys to CS1 students following course labs. It will use this data to drive an EWS which automatically identifies students needing support in a sufficiently short time frame. Finally, it will report these findings to stakeholders via a student-facing and instructor-facing dual dashboard. Ideally, the application will act as a general support platform, facilitating interactions between its stakeholders. Its overall goal is to help improve student retention on first year computer science courses.

1.3 Report Structure

The following report details the research, design, implementation and evaluation stages of the project and its software product. Chapter 2 describes the software's requirements gathering process. It includes a literature review and an analysis of existing software. Chapter 3 outlines the high level design decisions which informed software's development. Chapter 4 describes the development methodology used and the implementation of the software's most complex features. Chapter 5 describes the project's testing and evaluation stages and discusses their results. Finally, chapter 6 summarises the current state of the project and suggests directions for future work and research.

Chapter 2

Analysis

This chapter details the research which informed the project's scope and software requirements. Firstly, a literature review is presented to frame the project within contemporary research. Secondly, existing software tools pertinent to the project's objectives are analysed. Following this, the requirements gathering process is described. The chapter concludes by outlining the software specification which resulted from this process.

2.1 Literature Review

2.1.1 Surveying emotional response to CS1 components

There is a rising interest in monitoring students' emotions in CS education research. A subset of this research has focused on tracking students' affective response to specific components of CS1 courses, such as labs and assignments. These studies aim to identify learning barriers and assist with interventions. The vast majority have focused on programming labs and assignments [33, 26, 29, 19, 42], although there has been some expansion into other contexts, such as algorithms labs [63].

Several of these research papers have indicated a correlation between a student's affective state and their achievement. McKinney and Denton conducted pretest and posttest surveys following programming tasks and found that affective states such as interest/enjoyment, perceived pressure, and perceived competence (among others) significantly correlated with students' CS1 programming grades [33]. They also discovered that students' positive affective states tended to decline over the duration of the course, but this could be tempered with timely affective interventions [33]. Lishinki et al. found students' emotional reactions to programming projects had significant effects on future project outcomes. Feeling frustrated or inadequate had a particularly lasting negative impact [29]. Finally, Haden et al. investigated student affect in programming labs and found that students' satisfaction, confidence with planning, and sense of improvement significantly correlated with their final marks. However, unlike McKinney and Denton, they found attainment did not correlate with interest, perceived difficulty, or familiarity with the material [19].

Research has also shown affective surveys may also provide insights beyond the measurement context. Kinnuen and Simon's investigated students' emotional experience during programming assignments and identified that a student's "auxiliary emotional load", that is to say, course issues and life events, influenced how students reflected on their assignments. Students openly acknowledged this influence, despite knowing it was outside the scope of the questions they were asked [26]. Such influence may assist with detecting students at risk for reasons extraneous to the course artefact (labs, assignments, etc.) under measurement.

The above studies suggest there is merit to using affective feedback data to direct interventions. However, only McKinney and Denton's study reported doing this. The studies also highlight that

no standardised affective measurement instrument currently exists. The majority used unvalidated Likert scales and only Lishinki et al.'s was based on a previously validated survey instrument - the intrinsic motivation inventory [29]. Only two of the papers examined deviated from using Likert scales. Kinnuen and Simon used semi-structured interviews, which offered greater insight into students' affective state and auxiliary emotional load at the cost of being significantly more labour intensive to analyse [26]. Haden et al. used 2-dimensional grids which bound affective criteria together in pairs. There is precedent for this novel approach, as affective state co-occurrence has been documented in previous programming studies [10]. However, it remains unvalidated and the pairs chosen were based only on instructor intuition [19]. Further analysis of their *Student Affect Tool* is given in section 2.2.

2.1.2 Learning Analytics and Learning Dashboards

As mentioned in chapter 1, learning analytics forms the basis of a large number of early warning systems. Learning analytics tools can benefit stakeholders across all levels of education. This project primarily targets the meso-level (curriculum, instructors), where benefits of LA systems include [22]:

- Real-time: Monitoring learning progression, Creating meaningful interventions, Increasing interaction
- Predictive: Identifying learners at risk, Planning interventions

It should be noted that the majority of evidence for LA tools' effectiveness lies at this meso-level (learning support and teaching), whereas their effectiveness at the micro-level (improving learning outcomes) is under greater contention [2].

LA tools use a variety of methods for achieving their goals. Primary techniques include distillation of data for human judgement, prediction, and outlier detection [28]. A shift toward the former in place of automated alternatives has been reported in recent years, as it allows stakeholders to "better leverage human judgement" [23]. Learning dashboards (LDs) are perhaps the most commonly used implementation of this approach. These are displays that aggregate "different indicators about learner(s), learning process(es) and/or learning context(s) into one or multiple visualizations" [53]. However, care must be taken in their design. If dashboards are not aligned with an instructors' pedagogical philosophy, the data presented may not make sense to the user, limiting the tool's utility [11]. Therefore they to be "sufficiently configurable that they can be enfolded into existing instructional practices" [11].

Dashboards which display identical, or near-identical, information to both students and instructors are known as dual-dashboards. One motivation behind such developments is that meso-level data can 'nudge' students to seek support when they otherwise may not [17]. Another is that dual dashboards' transparency addresses power imbalances between stakeholders [28], alleviating student concerns over invasive observations and reduced autonomy [51]. Nevertheless, care should be taken when implementing dual dashboards, as a lack of alignment with learning theory can result in negative reception. For example, displaying a students performance with respect to their peers may motivate performance-approach orientated students but demotivate others by impairing their academic self-concept [23, 52]. An example of a dual dashboard software tool is analysed in section 2.2.

2.1.3 Student Feedback as a Data Source and Educational Sentiment Analysis

To date, comparatively few LA systems use direct student feedback (such as questionnaires and interviews) as a primary data source. For instance, Schwendimann et al.'s literature review of LA dashboards reveals only that 13% (7 out of 55) used such feedback as a data source (and of these, only one used it exclusively) [53]. This is perhaps surprising given that universities have traditionally used feedback surveys extensively and that their explicit connection to the student experience means they is less likely to suffer from the same *apophenia* as other LA data sources [24]. However, exceptions are found in the area of educational sentiment analysis. Sentiment analysis is "the process of identifying and classifying users' opinions from a piece of text into different sentiments - for example, positive, negative, or neutral - or emotions such as happy, sad, angry, or disgusted to determine the user's attitude toward a particular subject or entity" and it is achieved using "machine learning and natural language processing techniques" [47]. Rani and Kumar's literature review of educational sentiment analysis tools cites 3 studies which use course feedback surveys as a data source [4, 14, 35], 2 which use feedback from a student response system (including their own) [47, 6], and 1 which uses student diary entries [38]. While two of the papers explicitly refer to sentiment analysis's potential for classroom interventions [47, 38], none investigate the effectiveness of any intervention based on the systems' output, instead choosing to focus on the efficacy of the algorithms. This indicates the potential for further research into using direct student feedback and sentiment analysis of qualitative responses as the basis for an EWS.

2.2 Analysis of Existing Software

The following section presents the analysis of existing software products which was used to inform *Check In*'s features and requirements. The first two products were specifically created for CS1 courses and aim to help instructors identify students at risk. The third is a commercial survey platform.

Student Affect Tool

The Student Affect Tool is a series of 2-dimensional grids which ask students to evaluate their programming experience in labs against pairs of affective criteria (see figure 2.1). The tool provides instructors with several data visualisations. Scatter diagrams show answer distributions for each lab's grid (see figure 2.2) and line charts illustrate how students' affective responses vary over the course of a semester. The software's goal is to provide instructors with actionable data to support early interventions with students experiencing difficulties [19, 42].

Using an affective XY grid to collect responses offers several benefits. Firstly, it provides an alternative to qualitative data collection instruments which are "very expensive in terms of resources (physical and human) and time" [42]. Secondly, observers can use the grid to identify students in "sweet spots" and "danger zones". These are quadrants on the grid representing dual affective states which can influence learning in positive or negative ways. The authors note that a single dimension is often not enough to fully determine "danger zones". They give the example that a "hard" lab can affect learning positively if the student is engaged and working at a level of "engaged concentration" or negatively if they are bored [5, 42].

The current iteration of the tool would benefit from several modifications. Firstly, a dual dashboard which offers students support status notifications would address power imbalances and also limit survey fatigue, as disengagement is primarily induced when students feel little action is taken in response to their input, rather than by the number of surveys taken [56]. Secondly, judicious inclusion of qualitative questions may support instructors' interpretation of the data, as the tool's authors state that further student interviews are currently needed to confirm the accuracy of their judgements [42]. Finally, the software only offers "danger zone" insights via class level scatter plots. These can assist instructors with course diagnostics but cannot identify individuals within the zones. Implementing such a feature would be necessary for rapidly identifying individuals at risk in large cohorts.

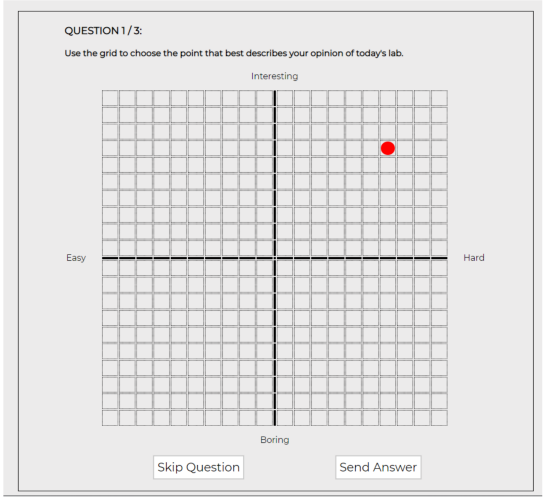


Figure 2.1: Screenshot of the Student Affect Tool showing an affective evaluation question (*Easy-Hard vs Boring-Interesting*) [42]

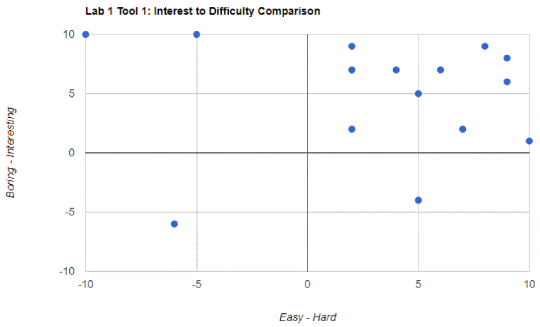


Figure 2.2: Example teacher visualisation in Student Affect Tool showing all students responses for the *difficulty* and *interest* dimensions for a single programming lab. [42]

Students At Risk

StudentsAtRisk is an early detection system which flags students who have not engaged with course materials for two weeks as "at risk". It features a dual-facing dashboard as well as the ability for instructors to manually flag students and students to manually flag themselves [1].

This implementation has many positives. Firstly, it empowers students. Its dual dashboard offers transparent representations of data to both instructors and students and its its novel "self-flag" feature promotes student agency (see figure 2.3) [1] . Another positive is that it assigns student risk on a per course basis rather than across the full CS1 programme [1]. This notifies instructors best suited to intervene and also may assist students with adjusting their workload across courses, which is important as declining performance in one course is often the result of refocusing efforts in another [25]. Finally, its simple dashboard interface focuses on a few select metrics and gives feedback on the "how and why" of flagging. This benefits instructors, who often struggle to make sense of dashboards utilising various data sources [48] and find opaque signals unhelpful with their teaching [11].

There are two main criticisms of the application. Firstly, it utilises a visualisation which compares

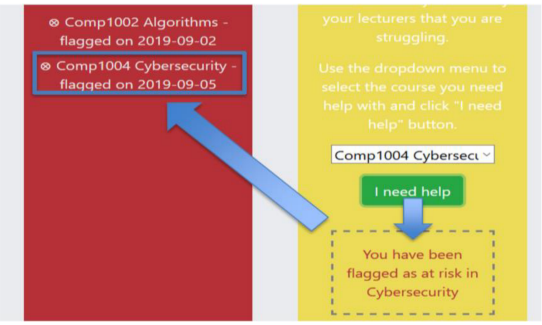


Figure 2.3: StudentsAtRisk's self flag feature [1]

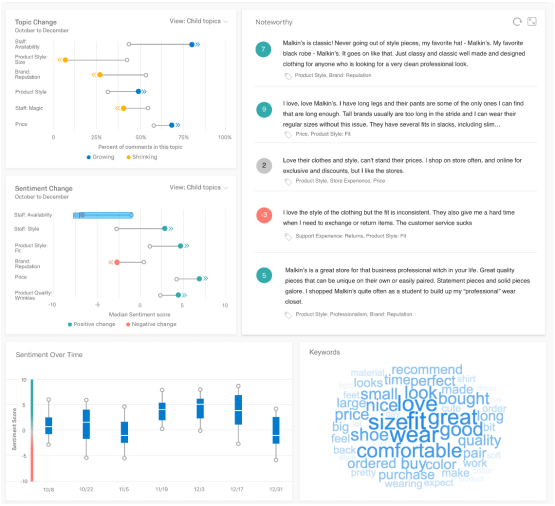


Figure 2.4: Qualtrics's sentiment panels, as advertised on its website [45]

students with their peers, which is problematic for the reasons described previously (see section 2.1.2). The second is that its automatic detection algorithm, while intuitive, has yet to be evaluated in practice [1] and its simplicity may not represent the complex and multidimensional factors that determine student success [28]. That said, such simplicity may not necessarily be detrimental to short term risk detection. A recent literature review of more sophisticated predictive models indicates that relatively few (8 out of 47) were found to work with high accuracy on a time frame of less than a quarter of a module [46]. Additionally, lack of algorithm transparency in such models can cause instructor frustration [11]. Thus, simple approaches should not be ruled out, though in such cases instructors should have control of the configuration, both to align with their pedagogy and to adapt thresholds to varying course content and cohorts.

Qualtrics XM

Qualtrics XM is a commercial survey platform used by "over 2000 colleges and universities". It claims to provide stakeholders with higher education insights, including into student engagement. Its *iQ* platform uses statistical analysis and machine learning to generate insights by matching response data (which it terms *experience data*) to a business's operational data [45]. *iQ* can perform sentiment analysis on survey text responses as well as classify comments by topic. The results are displayed using dashboard visualisations, which can show trends over time (see figure 2.4). In the context of this project, such a feature could automatically identify and prioritise known red flags from CS retention literature.

A drawback of Qualtrics's sentiment analysis is that the corpora used to train its machine learning model is unknown. Rani and Kumar state that mismatched domains can result in erroneous sentiment analyses. They give the example of the word "early" as having negative connotations in education ("The lecture is too early!") but positive ones in retail ("The courier arrived early.") [47]. Qualtrics's presumably generalised model may be susceptible to such errors. It does offer a feature to manually correct any misinterpreted sentiment, but this could be laborious if there are many false positives.

2.3 Requirements Gathering

The project's requirements gathering process followed Sommerville's requirements elicitation and analysis model shown in figure 2.5 [61]. The literature review and analysis of existing applications above informed the initial *requirements discovery and understanding* phase, along with open interviews with the project's client. The client was Dr Bikanga Ada. Her background in CS education research made her representative of a target user and lent pedagogical authority to the process.

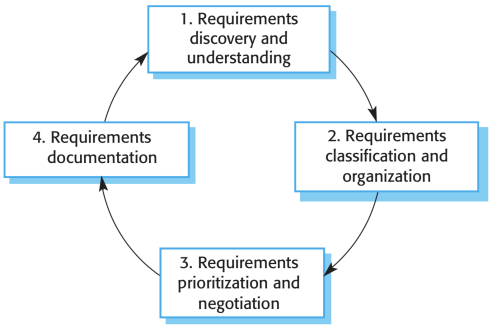


Figure 2.5: The requirements elicitation and analysis process (from Sommerville [61])

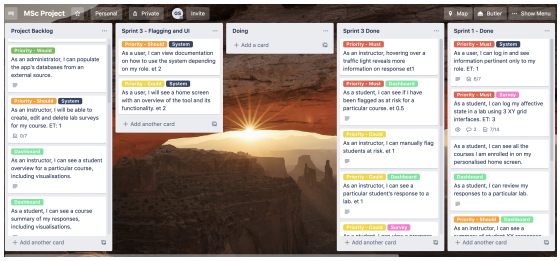


Figure 2.6: The project's kanban board with user stories and tasks

For the *classification and organization* phase, an information architecture document was prepared to synthesise the first phase's findings. This document included a product overview, user personas, a provisional list of functional and non-functional requirements, a proposed system architecture, an entity relation (ER) diagram, initial design wireframes and a sitemap. This was presented to the client for the *requirements prioritisation and negotiation* phase, resulting in amendments to the initial specification and the design prototype. The specification was delineated into the three main platform features: the survey instrument, the early warning system, and the dual dashboard. Requirement priorities were then negotiated with the client using MoSCoW (must, should, could, would) classifications. The resulting information architecture document can be viewed in compressed folder's `/docs/requirements/information_architecture_presentation.pdf`.

Additional input was required to prioritise student-orientated features. A survey was prepared and disseminated among students on a taught CS master's programme at the University of Glasgow. Questions queried the extent to which students would be happy to disclose information (to inform qualitative survey items), the devices they would use (to determine responsive design directions and cross platform compatibility), and preferences for dashboard functionality (to prioritise the student-facing dashboard features). Though the master's demographic was not fully representative of undergraduate CS1 users, it was a foundation course for students without a computer science background, thus was sufficiently comparable for the purposes of the survey. The survey and its results can be viewed in compressed folder's `/docs/requirements/student_survey.pdf` and `/docs/requirements/student_survey_results.pdf`. An ethics checklist for all surveys and evaluations is in `/docs/ethics_form.pdf`.

The final phase was *requirements documentation*. A product specification was finalised (see section 2.4) and a project backlog of user stories and tasks was created to support an agile development methodology (see figure 2.6 and chapter 4). This allowed requirements refinement at the sprint

planning stages and further negotiation during sprint reviews. The full list of the user story cards can be viewed in the compressed folder's `/docs/requirements/kanban_boards.pdf`.

2.4 Specification

This section outlines the functional and non-functional requirements of *Check In* and their MoSCoW priority as determined by the requirements gathering process. Additional item descriptions and justifications are given in appendix A.

2.4.1 Functional Requirements

Survey Instrument

- Provides an XY interface for querying students affective responses *(must-have)*.
- Students can enter qualitative feedback on their course. *(must-have)*.
- Qualitative feedback is analysed by sentiment analysis tools *(must-have)*.
- Instructors can create their own survey formats *(could-have)*.
- Qualitative responses can be automatically classified by topic *(would-have)*.

Early Warning System

- Calculate a risk factor for students based on their lab survey responses *(must-have)*.
- Automatically flag students at risk at regular time intervalss on a course by course basis *(must-have)*.
- Instructors can define and configure danger zones for question responses *(should-have)*.
- Course thresholds for risk are configurable *(should-have)*.
- Instructors can manually flag students at risk *(could-have)*.
- Students can flag themselves as at risk *(could-have)*.

Dashboard

- Instructors can see a summary of all students at risk on a course. *(must-have)*.
- Students can see if they have been flagged as at risk for a course. *(must-have)*.
- Instructors can review an individual student's survey responses. Students can review their own. Multi-level representations indicate response risk *(should-have)*.
- Users are given a reason for why a student has been flagged.*(should-have)*.
- Students can see graph summaries of their own lab responses over the course. Instructors can see all graph summaries. *(would-have)*.

General

- The system can be populated by course data from an external source *(should-have)*
- The system is synchronised with course data from an external source *(could-have)*.

2.4.2 Non-functional Requirements

Availability

- Interaction with the tool should not be bound to a specific time, location, or device. *(must-have)*.

Usability

- Students should be able to rapidly access their dashboard and complete each survey quickly *(must-have)*.
- Instructors should be able to determine students at risk rapidly with little cognitive load *(must-have)*.
- Clear documentation and descriptions are provided to ensure users can understand the tool and its operation *(should-have)*.

Privacy and Compliance

- Users have access only to information which pertains to their role *(must-have)*.
- The application is GDPR compliant *(could-have)*.
- Different types of instructor have different security privileges *(would-have)*.

Extensibility

- The system should be extendable to other courses and schools *(would-have)*.

Chapter 3

Design

The following chapter describes the high level design of *Check In*. The information architecture presentation created during requirements gathering formed the basis of the initial design. Its key components are discussed below. As the project followed an agile methodology (see chapter 4), its design and implementation were interleaved. Therefore, the subsequent sections should be viewed as the result of an iterative process, rather than a sequence of distinct development phases.

3.1 System Architecture

A key functional requirement was availability. It was therefore decided to realise the project as a web application. User installation would not be required and the ubiquity of browsers would allow students to access the software on their own devices, a clear preference in the student requirements survey's results. The Symfony 5 web framework was chosen as the project's primary technology stack. Symfony supports rapid development by combining generic modules with a standardised structure [55].

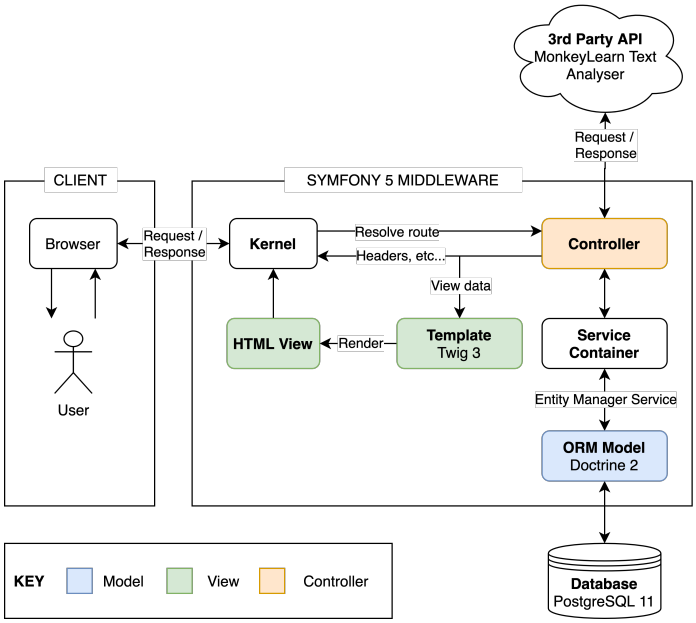


Figure 3.1: System architecture with high level Symfony request / response workflow with MVC components highlighted

Symfony broadly follows a model-view-controller (MVC) architecture. HTTP requests are converted into responses by the `AppKernel` class, which calls `Controller` class methods based on the URL route requested. When data needs to be persisted, `Controller`s access and manipulate `Entity` classes. These are object representations of database entities, and form the *model*

layer of the application architecture. They are mapped to a relational database using Doctrine 2, a mature object relational mapper (ORM) [15]. The PostgreSQL relational database was chosen over alternatives in anticipation of the complex reporting required by the learning dashboard. For example, PostgreSQL can aggregate survey data over time frames using window functions which are not available in noSQL or lightweight relational alternatives (e.g. MySQL). To decouple database queries from other classes, `Repository` classes are used. This design pattern is well suited to encapsulating heavy and complex querying [16]. `Repository` classes are managed by the Symfony service container, making them available for dependency injection into `Controller`s and other classes. `Controller`s also use external services. MonkeyLearn's Text Analyser API was chosen for sentiment analysis [36], as training and validating a context-sensitive model would require time and resources beyond the project's scope. For the *view* layer, the application uses Symfony's Twig 3 template engine, which can interpolate dictionaries of PHP objects into HTML responses.

3.2 Risk Calculation

The system needed to assign risk to students based on their survey responses. The proposed solution was to assign *risk weights* to survey item responses which fell within their corresponding question's *danger zone* value bounds. XY questions' danger zones would correspond to quadrants on the response grid and sentiment questions' danger zones would correspond to confidence bounds for negative classifications. The students' combined risk weights for a lab was termed their *risk factor*. Risk factor calculation is outlined in figure 3.2).

$$riskfactor(\%) = \frac{\sum_{i=1}^{n} r_i}{\sum_{i=1}^{n} rmax_i}$$

where n = survey item index

r = risk weight assigned to student's ith item response

$rmax$ = the maximum risk weight possible for the ith survey item

Figure 3.2: Risk calculation for lab survey response

Students would be automatically flagged as needing support their lab risk factor exceeded $X\%$ for Y consecutive weeks. The system's initial values of $X = 70\%$ and $Y <= 2$ were determined by client interviews, but these would be configurable.

3.3 Database Design

The project used a top-down database design methodology. An ER diagram was created to identify major entities and associations for client validation (see the compressed folder's `/docs/requirements/information_architecture_presentation.pdf`). This was then used as a basis for constructing a relational schema (see appendix figure A.1).

Key application requirements were incorporated into the schema's design. The 1:M relationship

between Course and Course_Instance allows courses to be repeated and run parallel to each other. Students are enrolled on Course_Instances via the Enrolment table. Enrolments contain risk flag attributes so students can be flagged on a course by course basis. Another aspect of the design is the normalization of questions and lab surveys to meet the application's extensibility requirements. 1:M relationships between Lab and Lab_*_Questions permit changing survey items between labs. Additionally, 1:M relationships between *_Question tables and Lab_*_Questions reduce redundancy when reusing survey questions. Finally, a 1:M relationship between Lab_*_Question and Lab_*_Question_Danger_Zones permits multiple danger zones per question with varying risk weights. See figure 3.3 for a concrete example.

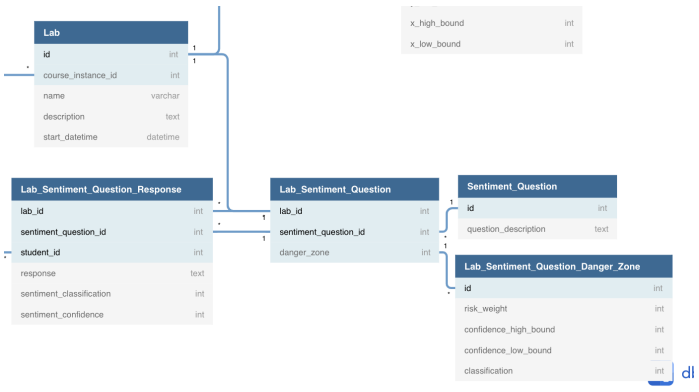


Figure 3.3: Normalization of sentiment question entities for flexibility and extensibility

It should also be noted that the schema reflects some idiosyncrasies of using the Doctrine ORM. For example, composite keys are not used because they can have a considerable performance impact [15].

3.4 Sitemap / Security Considerations

Sitemaps are an important aspect of web application as they influence how a user navigates the site and experiences its data. The application's sitemap is shown in figure 3.4. Two major decisions were made during its design.

The first was whether to use a multi-page or single-page survey format. A multi-page design was chosen as it allows partial data capture, even on non-completion, as well as flexible question order and skip logic [34]. However, this decision may affect mobile users, as studies have indicated that multi-page surveys take longer to complete on mobile devices [32].

The second regarded page privileges. To ensure transparency between stakeholders, several dashboard pages would be visible to both students and instructors. Exceptions were the survey question pages, as these would be solely for student use, and the course and lab summaries, which were restricted to instructors to reduce negative reactions to peer comparisons and for data protection reasons.

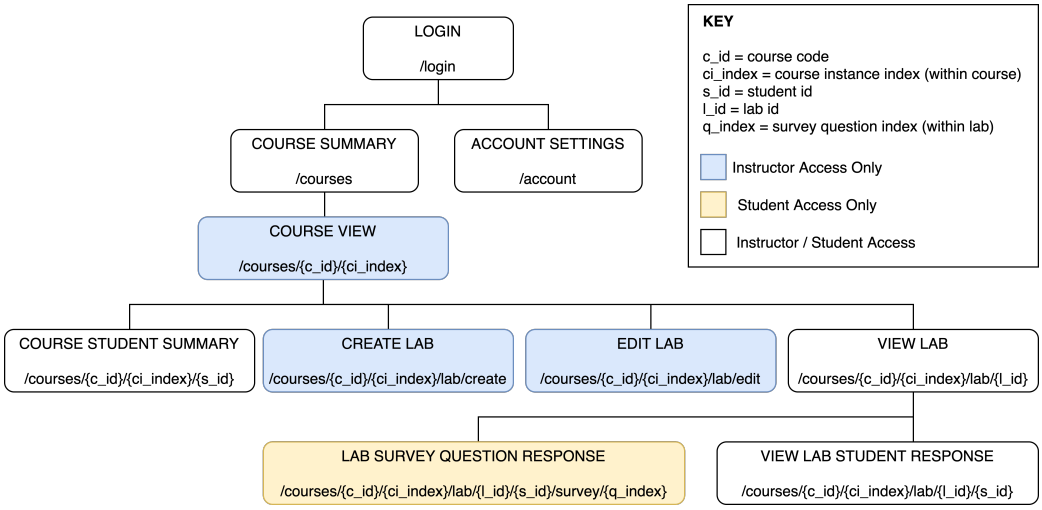


Figure 3.4: Sitemap design, including access restrictions

3.5 Wireframe Prototype and Interface Design

An interactive wireframe prototype of the application was created to explore user interface solutions with the client (see figure 3.5). This resulted in several key interface decisions.

Firstly, a 'breadcrumb' interface was proposed as the client was frustrated with early prototypes lacking clear navigation and signposting. Full paths would be displayed to all users to indicate the data hierarchy and hyperlinks would be contextual depending on the user's security privileges (see figure 3.9).

Secondly, the prototype quickly demonstrated the XY grid would be used in a variety of contexts: as a survey form component (see figure 3.6), a lab summary visualisation, and a danger zone form component (see figure 3.7). Class visualisations would feature coordinate response counts to address Haden et al.'s scatter plot shortcomings [42]. Class responses would also be displayed when configuring danger zones, as they can help an instructor identify problematic regions.

Finally, two student 'nudge' features were proposed. A progress wheel would feature alongside qualitative questions to encourage users to produce an appropriate response length (see figure 3.8). This is because sentiment analysis models are often trained on particular data sets, and whether the text is long form or short form can affect classification effectiveness [8]. A nudge prompt for students to self-flag following a particularly negative qualitative response was also designed. This would temper any limitations of the automatic flagging algorithm, particularly with respect to capturing students' auxiliary emotional load. The latter interface, along with the full prototype wireframes, can be viewed in the compressed folder's `/docs/design/wireframe_prototype.pdf`.

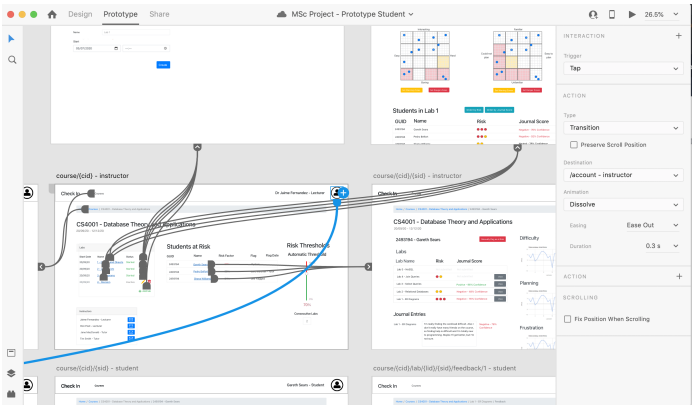


Figure 3.5: Creating the prototype interactions between the mock pages

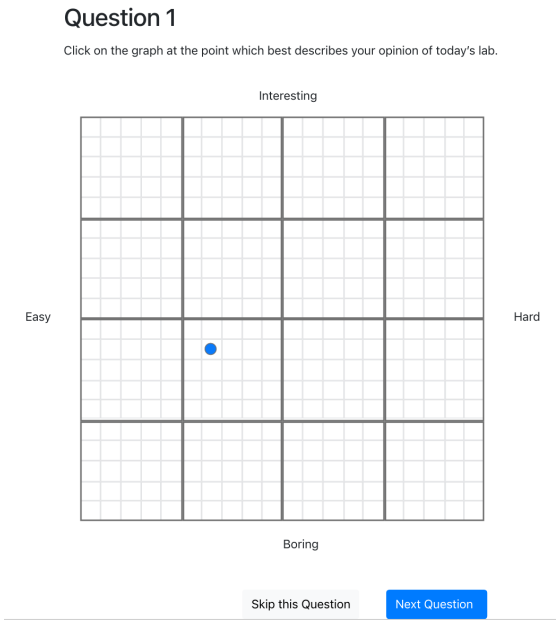


Figure 3.6: XY response form component

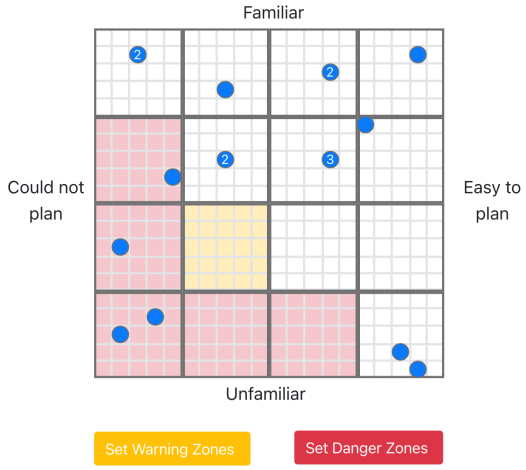


Figure 3.7: XY question responses and the ability to set danger zones on a single XY visualisation component

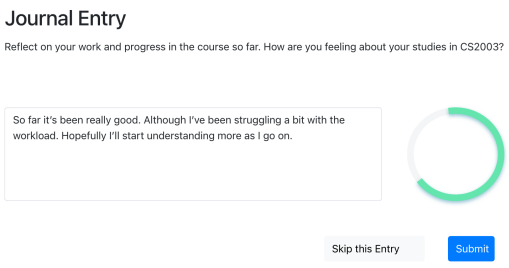


Figure 3.8: A nudge interface to promote optimal sentiment question response lengths

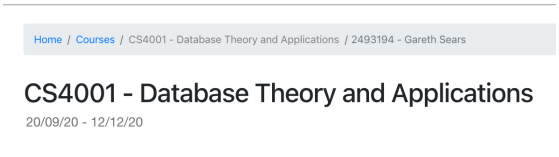


Figure 3.9: Student-facing breadcrumbs. Note the inactive hyperlink for the course page, which requires instructor privileges.

Chapter 4

Implementation

This chapter describes the project's development methodology and the most complex aspects of its implementation. For other features, the project's source code should be consulted. In addition to the Symfony framework, the software uses a number of external libraries. These are listed with their attributions and additional descriptions in appendix A.6. For an in-depth description of the compressed folder's contents and development setup instructions, please consult the folder's `README.md`.

4.1 Product Demonstrations

A video demonstration of the current implementation is provided at `https://youtu.be/SYm_O8dxNI0`.

An evaluation version of the application is hosted at `https://qlitmnms3a-vioxgpwe4okw6.eu.s5y.io/`. This will be available until November 30th 2020. Login using the following credentials:

- **Student email:** *test@student.gla.ac.uk* **Password:** *password*
- **Instructor email:** *test@glasgow.ac.uk* **Password:** *password*

4.2 Development Methodology

The project followed an agile development methodology. Three development sprints were time-boxed to implement the three major functional-requirement components (see section 2.4). Each sprint included a planning stage, a testing stage, a sprint review with the client, and a fix period to act on client feedback. These are outlined in the project's Gantt chart, which was prepared for client validation at the project's inception (see the compressed folder's `/docs/project_gant_chart.pdf`).

4.3 Application Security

The application's security builds on Symfony's *Security Bundle* modules. Broad access restrictions are applied to routes using regular expressions in the `/config/packages/security.yaml` configuration file. More nuanced privilege checking in `Controller`s is delegated to `Voter` classes (found in the compressed folder's `src/Security/Voter` directory). For example, the `CourseInstanceVoter` performs various database queries to determine if the current user has

edit or view privileges for the `CourseInstance` under review (see appendix B.1). If a `Voter` returns `false`, a 403 response is returned by the `Controller`. Another essential security check is performed by `CourseController::checkValidLabSurveyReferrer`. This checks that survey page has been accessed via the previous survey page to avoid flow of control errors.

4.4 The XY Grid Component

The Vue.js javascript framework was used to encapsulate the XY grid's HTML, CSS and Javascript functionality as components [66]. The parent component, `XYQuestion`, is formed of many `XYQuestionRange`s, which are in turn formed of many `XYQuestionCell`s.

The `XYQuestionCell` applies custom CSS styles to a native HTML checkbox so it can indicate multiple student responses when `XYQuestion` is in class response mode. When `XYQuestion` is used as a survey form item, its `XYQuestionRange`s allow clickthroughs for cell selection, but when used as a danger zone form, the ranges themselves can be clicked to set their risk weight. This is achieved by adding a click handler to an conditionally positioned `div` which overlaps the region's cells. The above is demonstrated in figures 4.1 and 4.2.

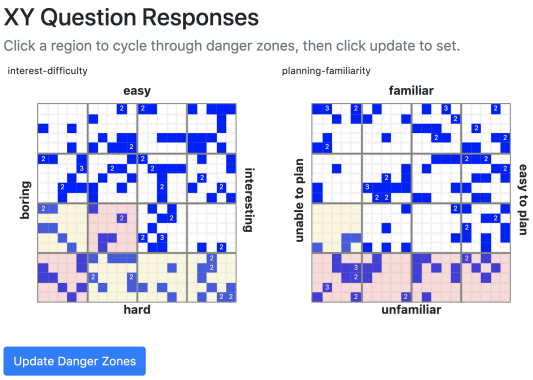


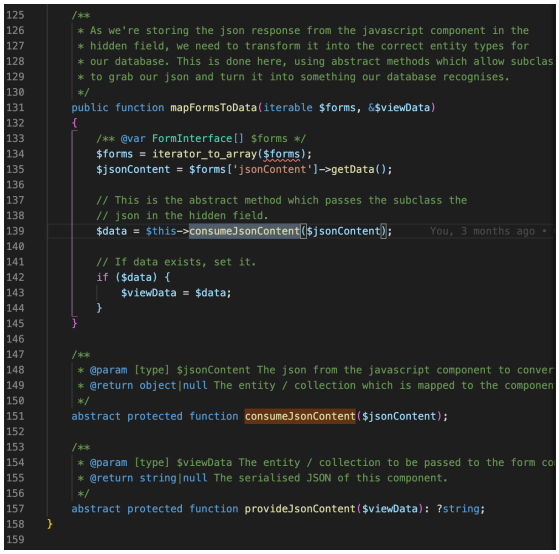
Figure 4.1: The resulting XY grid component being used as a danger zone form

```
<template>
  <div class="range">
    <!-- Overlay region selector -->
    <div
      :class="{
        selectable: !disabledRegions,
        warning: riskLevel() === 1 && !disabledColors,
        danger: riskLevel() === 2 && !disabledColors,
      }"
      @click="handleRegionClick($event)"
    ></div>
    <!-- Cell Range -->
    <div>
      <div class="y-row" v-for="row in yMax - yMin + 1" :key="row">
        <XYQuestionCell
          v-for="col in xMax - xMin + 1"
          :key="col"
          :name="name"
          :disabled="disabledPoints"
          :count="getCellCount(xVal(col), yVal(row))"
          :x-value="xVal(col)"
          :y-value="yVal(row)"
          @change="handlePointChange($event)"
        ></XYQuestionCell>
      </div>
    </div>
  </div>
</template>
```

Figure 4.2: The XYQuestionRange's template code. Note conditional placement of a region selector over cells depending on the parent component's mode

Once the frontend component was implemented, the next step was to map its data to backend `Entity` classes. Symfony's mechanism for doing this is extending the form API's `AbstractType` class. However, implementing this idiomatically with a Vue component presented several challenges. Firstly, Symfony's form views are rendered from `.twig` partials. Javascript inside these is rendered inline, which would block page loading. The solution was to create an `addLoadEvent` function, which delays javascript execution until the `document` element is fully loaded. The second challenge was serialising `Entity` classes into JSON which could be consumed by `XYQuestion`, and vice versa. This was solved using the `AbstractXYComponentType` class. This renders a hidden form field which is used to populate the `XYQuestion` component on a GET

request and pass its contents to the backend on a POST request. This process is shown in appendix B.2. `AbstractXYComponentType` also adopts the abstract template pattern so its subclasses (`LabXYQuestionDangerZoneType` and `XYCoordinatesType`) can implement their own JSON-`Entity` serialisation / deserialisation routines. This is shown in figures 4.3 and 4.4.



```
125     /**
126      * As we're storing the json response from the javascript component in the
127      * hidden field, we need to transform it into the correct entity types for
128      * our database. This is done here, using abstract methods which allow subclas
129      * to grab our json and turn it into something our database recognises.
130      */
131     public function mapFormsToData(iterable $forms, &$viewData)
132     {
133         /** @var FormInterface[] $forms */
134         $forms = iterator_to_array($forms);
135         $jsonContent = $forms['jsonContent']->getData();
136
137         // This is the abstract method which passes the subclass the
138         // json in the hidden field.
139         $data = $this->consumeJsonContent($jsonContent);        You, 3 months ago •
140
141         // If data exists, set it.
142         if ($data) {
143             $viewData = $data;
144         }
145     }
146
147     /**
148      * @param [type] $jsonContent The json from the javascript component to conver
149      * @return object|null The entity / collection which is mapped to the componen
150      */
151     abstract protected function consumeJsonContent($jsonContent);
152
153     /**
154      * @param [type] $viewData The entity / collection to be passed to the form co
155      * @return string|null The serialised JSON of this component.
156      */
157     abstract protected function provideJsonContent($viewData): ?string;
158 }
159
```

Figure 4.3: AbstractXYComponentType's abstract template methods. Note how consumeJsonContent is used in the mapFormsToData method.

```
78      public function provideJsonContent($viewData): ?string
79      {
80          if (!$viewData) {
81              return null;
82          }
83
84          // Serialise the XYQuestionDangerZone collection, ignoring the labXYQuesti
85          // as this won't be set by the javascript component and would result in ci
86          // serialization.
87          $out = $this->serializer->serialize(
88              $viewData,
89              'json',
90              [AbstractNormalizer::IGNORED_ATTRIBUTES => ['labXYQuestion']]
91          );
92
93          return $out;
94      }
95
96      public function consumeJsonContent($jsonContent)
97      {
98          if (null === $jsonContent) {
99              return null;
100         }
101
102         // Converts a json string into XYQuestionDangerZone entities.
103         return $this->serializer->deserialize($jsonContent, LabXYQuestionDangerZon
104     }
105 }
```

Figure 4.4: LabQuestionDangerZoneType's concrete implementations of AbstractXYComponentType's abstract template methods for serialising and deserialising JSON

4.5 Calculating and Displaying Risk

The application needs to determine three different levels of risk. Each is associated with a particular `Entity` class or interface. A `SurveyQuestionResponseInterface` has a risk weight, a student's `LabResponse` has a risk factor, and a student's `Enrolment` has a risk flag. These classes' `*Repository::get*Risk` methods (where * is the entity's classname) perform specialised database queries to calculate risk. These return `*Risk` classes which encapsulate the query result for visualisation and additional modification. For example, when `LabXYQuestionResponseRepository::getSurveyQuestionResponseRisk` is called with a `LabXYQuestionResponse` object, a doctrine query language (DQL) query determines if the response's coordinates lie within any of the corresponding `LabXYQuestion`'s `LabXYQuestionDangerZone`s. If so, the `LabXYQuestionDangerZone`'s risk weight is returned, wrapped in the `LabXYQuestionResponseRisk` class with the original `LabXYQuestionResponse` (see figure 4.7). A similar process can be followed to obtain a `LabResponseRisk` object. However, this contains an array of `SurveyQuestionResponseRisk`s rather than a risk weight.

Nesting `*Risk` container classes in this way simplifies view rendering, as Twig macros and functions can iterate over the risk object tree to create visualisations. For example, `lab_summary.html.twig` displays all student `LabResponseRisk`s in a table, with associated `SurveyQuestionResponseRisk`s rendered as "traffic lights" in its *Responses* column (see figures 4.5 and 4.6). These traffic light interfaces display detailed response information on hover or click. A similar pattern to

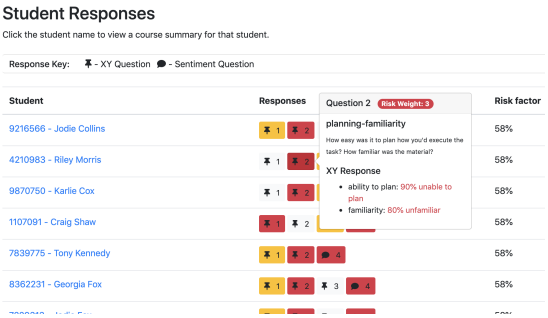


Figure 4.5: The lab summary page's student table built from LabResponseRisk classes

```twig
<tbody>
    {% for risk in labResponseRisks %}
        {% set student = risk.labResponse.student %}
        {% set href = path(
        constant('App\\Controller\\StudentSummaryPageController::ROUTE'),
        {
            courseId: courseInstance.course.code,
            instanceIndex: courseInstance.indexInCourse,
            studentId: student.guid
        }
    ) %}
        <tr>
            <td data-sort={{student.guid}}>
                <a href="{{href}}">{{student.guid}}
                    -
                    {{student.user.fullname}}</a>
            </td>
            <td class="risk-cell"
                data-sort={{risk.weightedRiskFactor}}>
                {{surveyQuestionResponseRisksToTrafficLights(
                    risk.surveyQuestionResponseRisks)}}
            </td>      You, a few seconds ago • Uncommitted changes
            <td data-sort={{risk.weightedRiskFactor}}>
                {% if risk.weightedRiskFactor > 0 %}
                    {{risk.weightedRiskFactor | number_format(0, '.', ',')}}%
                {% endif %}
            </td>
        </tr>
    {% endfor %}
</tbody>
```

Figure 4.6: Rendering LabResponseRisk classes in a .twig template

Symfony's form API was implemented to provide a common interface for rendering the specific information required by each question type. Subclasses of `SurveyQuestionResponseRisk` implement abstract methods which return the path of a Twig template partial and provide a context dictionary to populate it (see figure 4.8). `SurveyQuestionResponseRisk` objects can then be passed to the custom Twig template function `renderRisk`, which renders the required HTML fragment using these methods.

```php
    public function getSurveyQuestionResponseRisk(SurveyQuestionResponseInterfac
    {
        $entityManager = $this->getEntityManager();

        $query = $entityManager->createQuery(
            'SELECT dz.riskLevel
            FROM App\Entity\LabXyQuestionResponse xyr
            JOIN xyr.labXYQuestion xyq
            JOIN xyq.dangerZones dz
            WHERE xyr = :questionResponse AND
                dz.xMin <= xyr.xValue AND
                dz.xMax >= xyr.xValue AND
                dz.yMin <= xyr.yValue AND
                dz.yMax >= xyr.yValue'
        )->setParameter('questionResponse', $questionResponse);

        try {
            $labXYQuestionResponseRisk = new LabXYQuestionResponseRisk(
                $query->getSingleScalarResult(),
                $questionResponse
            );
        } catch (\Doctrine\ORM\NoResultException $e) {
            // No result, so we know there is no risk.
            $labXYQuestionResponseRisk = new LabXYQuestionResponseRisk(
                SurveyQuestionResponseRisk::LEVEL_NONE,
                $questionResponse
            );
        }

        return $labXYQuestionResponseRisk;
    }
}
```

Figure 4.7: Obtaining a LabXYQuestionResponseRisk object from the LabXYQuestionResponseRepository

```php
class LabSentimentQuestionResponseRisk extends SurveyQuestionResponseRisk
{
    public function getTwigTemplate(): string
    {
        return 'risk_summary/sentiment_response.html.twig';
    }

    public function getContext(): array
    {
        /**
         * @var LabSentimentQuestionResponse
         */
        $surveyQuestionResponse = $this->getSurveyQuestionResponse();

        return [
            'text' => $surveyQuestionResponse->getText(),
            'classification' => $surveyQuestionResponse->getClassification(),
            'confidence' =>  $surveyQuestionResponse->getConfidence(),
        ];
    }
}
```

Figure 4.8: Concrete implementations of SurveyQuestionResponseRisk's abstract template methods which are used to render sentiment question responses

Chapter 5

Testing and Evaluation

This chapter discusses the testing and evaluation phases of the project. Firstly, how the application generated test data is discussed. This is followed by the manual and automated testing methodologies and their insights. Following this, the evaluation strategy and its remote setup are described. The latter is pertinent due to COVID-19 contact restrictions. Finally, the evaluation results are detailed.

5.1 Data Fixtures

Artificial data fixtures were used throughout development to populate application's database with entities indicative of the target environment (students, courses, etc.). This allowed progressively increasing the complexity and quantity of data while testing features. Use of real data may have slowed development, as it requires data subject consent.

The first fixture class, `TestFixtures`, loads an exactly reproducible application state for manual testing (see section 5.3). The second, `EvaluationFixtures`, loads an environment representative of 9 weeks into a computer science course with 300 students and 30 instructors. Both extend Symfony's *Doctrine Fixture Bundle*'s base classes and use the `EntityCreator` class extensively. `EntityCreator` was written to reduce routine coupling by automating database flushes and encapsulating association routines when creating entities. Additionally, `EvaluationFixtures` uses the *Faker* library to ensure uniqueness of randomly generated primary keys and create realistic attributes [67].

5.2 Automated Testing

Automated tests were split into unit tests and "functional" tests (Symfony nomenclature for integration tests). As full test coverage was not possible due to project time constraints, automated testing was restricted to essential calculations and functionality which would be difficult to observe manually. This was:

- Risk calculation in container classes (unit tests).
- Error checking in entities to avoid invalid database state (unit tests).
- Route privileges and application security (functional tests).
- Database queries in repository classes (functional tests).

All unit tests inherited from *PHPUnit*'s `TestCase` class. This provides common assertions, an object mocking API, and annotations for data provider injection. Data provider functions and mock

objects were used extensively for a clean testing API and separation of concerns (see appendix B.3). Using mocks also revealed some tightly coupled data fetching implementations. For example, `SurveyQuestionResponseRisk::getDefaultContext` required several nested mock classes to emulate correct data fetching. Such nesting could be avoided by favouring repository methods with join queries over entity methods in future work.

All functional tests inherited from the `FunctionalTestCase` abstract class, which was written to set up and provide the `EntityCreator` in tests. This allowed creation of specific database fixtures within the methods themselves. These transactions were then rolled back after each test using the *Doctrine Test Bundle* [31]. This process ensured immutable, self-contained functional tests which would not be possible with a shared fixtures. `FunctionalTestCase` also extended Symfony's `WebTestCase` class, which provided it with a web crawler and common web response assertions. These permitted testing user interactions, such as logging in and attempting to access unauthorised content (see appendix B.4). To overcome Symfony's restriction that a web client cannot be initialised after interacting with the service container (required for `EntityCreator`), `WebTestCase::createClient` was overridden to ensure a kernel shutdown prior to client creation.

All automated tests pass at the time of writing. All test code can be found in the compressed folder's `/tests/` path. An HTML coverage report is located in the `docs/testing/coverage` folder. The `README.md` file details how to run the test suites.

5.3 Manual Testing

Manual testing was used to test the functional requirements of the application and the user interface. Test tables correspond to the software's main components:

- The dashboard interface (a table for each page).
- The early warning system and manual flagging functionality.
- The survey instrument.

Each table describes the requirement / feature under test, along with individual test cases for that feature (see figure 5.1). To ensure valid, reproducible tests, specific test steps were documented which correspond to the deterministic data created by the `TestFixtures` class. Prerequisite test case IDs which must be completed prior to undertaking a test are also documented.

Manual testing asserted all features function as intended, with the exception of minor interface bugs. These included incorrect table sorting for time columns, displaying PM time as AM times in isolated contexts, and some misnomers. These do not impede the application's required functionality. All tables and results can be reviewed in the compressed folder's `/docs/testing/manual_tests.pdf`.

User: Instructor
Module: Student Course Summary Dashboard

Before each:
1. Login using 'test@student.gla.ac.uk' and 'password'.

Test Scenario	ID	Test case	Test steps	Pre conditions	Expected result(s)	Status	Actual Result
Verify Breadcrumbs	SCI1	View breadcrumbs	1. Navigate to 'courses/COMPSCI101/1/1234567'. 2. View 'breadcrumb' interface at top of page.		1. Displays 'Courses / COMPSCI101 – 1 / 1234567 – Test Student'. 2. 'COMPSCI101- 1' is a valid link. 3. 'Courses' is a valid link.	PASS	(expected)
	SCI2	Breadcrumb links functioning	1. Navigate to 'courses/COMPSCI101/1/1234567'. 2. View 'breadcrumb' interface at top of page. 3. Click 'Courses'. 4. Navigate to 'courses/COMPSCI101/1/1234567'. 5. View 'breadcrumb' interface at top of page. 6. Click 'COMPSCI101-1'.		1. 'Courses' redirects to '/courses'. 2. 'COMPSCI101 – 1' redirects to '/courses/COMPSCI101/1'.	PASS	(expected)
Verify Pending Lab Surveys	SCI3	Pending lab survey table displays incomplete surveys up to current date	1. Navigate to 'courses/COMPSCI101/1/1234567'. 2. View 'Pending Lab Surveys'.		1. Displays lab information in Table 7. 2. Displays no other lab surveys. 3. Default order by date, latest first.	FAIL	Order is earliest first. All else as expected.
	SCI4	Instructor cannot navigate to survey	1. Navigate to 'courses/COMPSCI101/1/1234567'.		1. No button to navigate to lab survey exists.	PASS	(expected)

Figure 5.1: A section of the Course Summary Dashboard testing table used during manual testing.

5.4 Evaluation Strategy

The first stage of the evaluation required the user to login and use the application to assert the software's functional requirements. This followed a scenario-testing approach. Each scenario described a typical usage context based on the user's role (instructor or student) and then stated a goal for the user to achieve using the software. Users were instructed to indicate if they *completed*, *partially completed*, or were *unable to complete* the objective following their attempt. This methodology was chosen as it tests several requirements within the same scenario and helps stakeholders better relate to the testing process [61]. This was important, as user feedback from an initial evaluation pilot (2 participants) indicated difficulties in understanding concepts and executing tasks when context was omitted.

The second stage required users to reflect on their experience of the software. The users were first asked to complete the User Experience Questionnaire (UEQ). This instrument was chosen because it has been previously validated, offers benchmarking against 452 previous studies, and can evaluate hedonic as well as pragmatic qualities of software [27, 50, 49]. The latter is important, as Jivet et al. have previously criticised learning dashboard evaluations for focusing on perceived usability at the expense of emotional impact [23]. The UEQ measures 6 scales with 26 7-point Likert questions. These scales are [49]:

- **Attractiveness:** Overall impression of the product. Do users like or dislike the product?
- **Perspicuity:** Is it easy to get familiar with the product? Is it easy to learn how to use the product?
- **Efficiency:** Can users solve their tasks without unnecessary effort?
- **Dependability:** Does the user feel in control of the interaction?
- **Stimulation:** Is it exciting and motivating to use the product?
- **Novelty:** Is the product innovative and creative? Does the product catch the interest of users?

Following the UEQ, users were asked two questions relating to the project's pedagogical goals. Again, this follows Jivet et al.'s recommendations, as they state such goals are often neglected in learning dashboard evaluations [23]. The questions use a 7 point Likert-scale with optional comment fields for elaboration. Instructors were asked if they felt the tool would support interventions and integrate with their pedagogy. Students were asked if the tool would influence their help-seeking behaviour and give them confidence that the faculty were supporting them. The final survey item was a comment box for general observations regarding the software.

The complete survey including scenario tests can be found in the compressed folder's `/docs/evaluation/evaluation_survey.pdf` document.

5.5 Evaluation Setup

The user evaluation process needed to be fully remote due to pandemic restrictions. To achieve this, an evaluation version of the application was hosted on Symfony cloud and populated by the `EvaluationFixtures` class. As *DoctrineFixtureBundle* is not available on a production server, the database needed to be populated via SSH tunnelling (see [62] for further discussion). To avoid debilitating load times, `EvaluationFixtures` was optimised to avoid database flushes and queries during fixture creation. This resulted in increased coupling, though such practices are not discouraged by the Symfony community [54]. All evaluation users shared the same application instance to simulate actual use.

The survey was hosted via Google Forms. A particular survey branch was presented depending on the user's chosen demographic. The login process involved directing users to Google Sheets containing role-specific credentials. These credentials were hidden programatically when selected to prevent other users sharing the same test accounts. The survey was then disseminated to instructors at the University of Glasgow as well as computer science students from various year groups and institutions.

5.6 Evaluation Results

Key findings from the evaluation will now be discussed. Further analysis figures can be found in the compressed folder's `/docs/evaluation/evaluation_results_analysis.pdf`. Raw data can be found in `/docs/evaluation/evaluation_results_raw.xlsx`.

5.6.1 Demographics

The evaluation collected 15 responses from 6 CS students and 9 CS instructors. The instructors consisted of 7 lecturers and 2 tutors. The students consisted of 3 undergraduates in second year or above and 3 students on taught postgraduate courses. All respondents were from the University of Glasgow.

5.6.2 Scenario Testing

4 students (67%) completed all scenarios and 2 students (33%) achieved 91% completion. Student 5 was *unable to complete* scenario 3 (review submitted survey results) and student 6 was *unable to complete* scenario 7 (confirm correct sentiment analysis). The latter result could indicate incorrect analysis or an inability to locate the results.

Only 1 instructor (11%) achieved 100% scenario completion. 5 instructors (56%) achieved 92% completion, 1 instructor (11%) achieved 83% completion, 1 instructor (11%) achieved 50% completion, and 1 instructor (11%) achieved just 33% completion. This last instructor (instructor 8) did not complete scenarios 5-12 due to time constraints, commenting that the evaluation was too long. Future scenario test responses should include a *skip* option to distinguish such cases from being *unable to complete* scenarios.

Only instructor scenarios 4, 5 and 11 presented difficulties for multiple users (discounting instructor 8's skipped responses). Scenario 4 (find a 'risk factor' explanation) was the most problematic, receiving *partially completed* responses from 11% of users (1) and *unable to complete* responses from 22% of users (2). Next was scenario 5 (change the XY danger zones for a lab), which received *partially completed* responses from 33% of users (3). Finally, scenario 11 (review sentiment question responses) received *partially completed* responses from 11% of users (1) and *unable to complete* from 11% of users (1). Such results indicate the user interface and task clarity inhibited task completion, rather than functionality (which would have resulted in more constant failures for particular items). This is confirmed by user experience comments in section 5.6.3.

Instructor scenario testing also revealed an unforeseen error with automatic flagging. During a mid-evaluation inspection, one course was observed as having all 250+ students automatically flagged (see appendix B.5). It is extremely unlikely this was intentional. An instructor may have been experimenting with a course's risk factor threshold, setting it to 0% immediately before the `FlagStudentsTask` cron job ran. This would apply flags to all students, rather than simply previewing those affected by the configuration change. While rare, this is an extremely disruptive bug which would cause undue concern among students.

5.6.3 User Experience

Due to the sample size, much of the second stage analysis has large variances and is not statistically significant. For example, no UEQ scale p-values fall below 0.05. The insights below are therefore discussed in terms of confidence intervals, though they should be viewed as inconclusive pending further evaluations. Schrepp et al.'s interpretation guidelines state that values between -0.8 and 0.8 represent a more or less neutral evaluation, values ¿ 0.8 represent a positive evaluation, and values ¡ -0.8 represent a negative evaluation. Answers above +2 or below -2 are extremely unlikely due to response distribution and answer tendencies (e.g. avoidance of extreme categories) [49]. The data below only showed minor inconsistencies between users' UEQ item responses and their corresponding UEQ scales. This asserts the responses' validity [49] (see *Item Response Inconsistencies* in the shared folder's `/docs/evaluation/evaluation_results_analysis.pdf`).

All instructor evaluations' 95% confidence intervals fall in the neutral-positive range (see figures 5.2 and 5.4). However, when benchmarked against other software, these responses indicate generally negative reception, particularly with respect to perspicuity, efficiency, and dependability (see

UEQ Scales (Mean and Variance)		
Attractiveness	0.907	1.01
Perspicuity	0.250	2.47
Efficiency	0.472	1.04
Dependability	0.528	0.48
Stimulation	0.833	0.72
Novelty	0.833	0.17

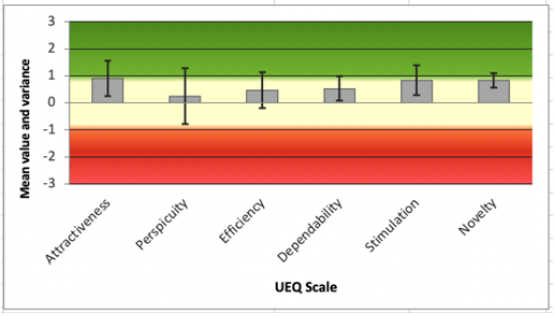


Figure 5.2: Instructor UEQ scale responses

UEQ Scales (Mean and Variance)		
Attractiveness	1.500	0.19
Perspicuity	1.042	2.34
Efficiency	1.708	0.69
Dependability	1.625	0.24
Stimulation	1.458	0.24
Novelty	0.750	0.73

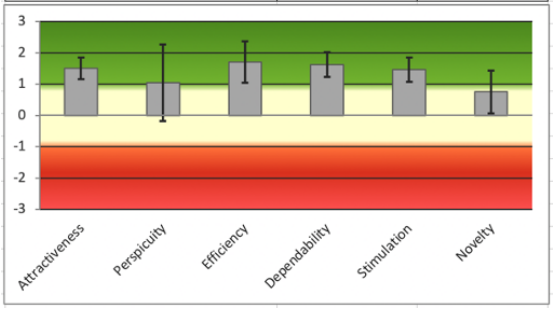


Figure 5.3: Student UEQ scale responses

Confidence intervals (p=0.05) per scale						
Scale	Mean	Std. Dev.	N	Confidence	Confidence interval	
Attractiveness	0.907	1.007	9	0.658	0.249	1.566
Perspicuity	0.250	1.571	9	1.027	-0.777	1.277
Efficiency	0.472	1.019	9	0.666	-0.193	1.138
Dependability	0.528	0.690	9	0.451	0.077	0.978
Stimulation	0.833	0.848	9	0.554	0.279	1.387
Novelty	0.833	0.415	9	0.271	0.562	1.104

Figure 5.4: Instructor UEQ p=0.05 confidence intervals

Confidence intervals (p=0.05) per scale						
Scale	Mean	Std. Dev.	N	Confidence	Confidence interval	
Attractiveness	1.500	0.435	6	0.348	1.152	1.848
Perspicuity	1.042	1.528	6	1.223	-0.181	2.264
Efficiency	1.708	0.828	6	0.662	1.046	2.371
Dependability	1.625	0.494	6	0.395	1.230	2.020
Stimulation	1.458	0.485	6	0.388	1.070	1.847
Novelty	0.750	0.851	6	0.681	0.069	1.431

Figure 5.5: Student UEQ p=0.05 confidence intervals

figure 5.6). Student's confidence intervals were more favourable, with all scales having a positive lower bound except novelty and perspicuity, whose lower bounds were neutral (see figures 5.3 and 5.5). Benchmarks indicate students compared the software favourably to other products, with the exception of its perspicuity and novelty, which were below average. These results indicate that immediate areas for improvement generally concern the dashboard and configuration complexity, particularly for instructors. This is supported by user comments. 2 instructors (22%) commented on navigational difficulties and 4 instructors (44%) and 1 student (17%) commented on difficulties understanding the interface or application operation.

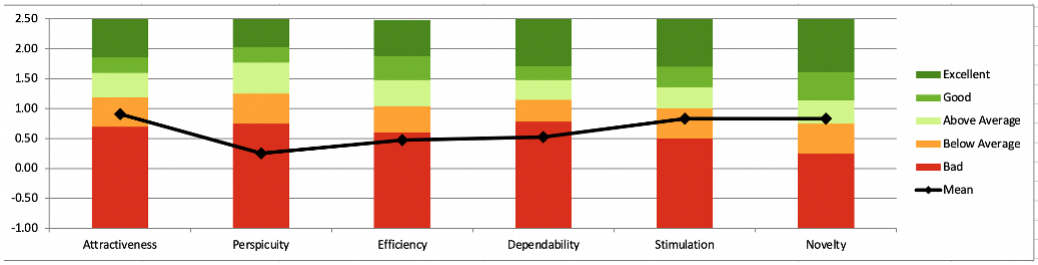


Figure 5.6: Instructor UEQ responses against the UEQ's benchmark dataset []

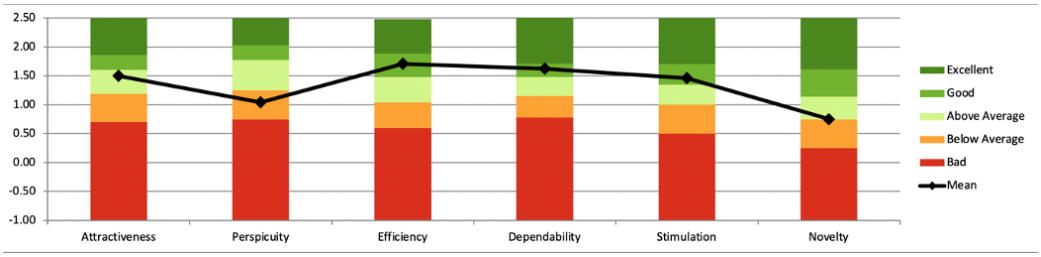


Figure 5.7: Student UEQ responses against the UEQ's benchmark dataset []

Confidence intervals (p=0.05) per question						
Question	Mean	Std. Dev.	N	Confidence	Confidence interval	
Detect and plan interventions	4.78	1.20	9	0.785	3.993	5.563
Integrate into pedagogy	4.56	1.24	9	0.808	3.748	5.363

Figure 5.8: Instructor project goal questions p=0.05 confidence intervals

Confidence intervals (p=0.05) per question						
Question	Mean	Std. Dev.	N	Confidence	Confidence interval	
Seek support	5.83	0.98	6	0.787	5.047	6.620
Instructor support confidence	5.83	0.75	6	0.602	5.231	6.436

Figure 5.9: Student project goal questions p=0.05 confidence intervals

5.6.4 Project Goals

The 95% confidence intervals for instructors' pedagogical question responses ranged from *neither agree nor disagree* to *slightly agree* (see figure 5.8). This reticence correlates with their perceived complexity of the application. Instructor 6 was concerned about the configuration required before they could reach "useful insights" and instructor 2 stated it was "too complicated" to integrate in their teaching. However, instructor 5, 8, and 9's comments were enthusiastic about its potential utility.

Students generally agreed that the software would both promote their help-seeking behaviour and instill confidence that they were supported by the faculty. Both confidence intervals ranged between *slightly agree* to just above *agree* (see figure 5.9. Feedback tended toward interface improvement suggestions to reduce cognitive load. For example, student 3 suggested replacing the 'explain this page' tour with localised 'hover' documentation to avoid unnecessary explanations and student 4 suggested colour coding the XY questions to help identify negative regions faster. Student 4 also observed that to feel fully supported, they would need an indication that their flags were being followed up by instructors.

5.7 Evaluation Limitations

Several limitations were identified during and following the evaluation analysis. Firstly, no first year undergraduates responses were collected, despite these being the software's target demographic. Another limitation was that instructors had no knowledge of the student dashboard and how it was used. Instructor 4 commented that this inhibited their understanding of the application. Finally, the artificial fixtures were not fully indicative of real data. For example, lab XY responses were randomly distributed (see figure 4.1), rather than clustering as would be expected with real cohorts.

Chapter 6

Conclusion

The project goal was to develop a dual dashboard for CS1 students and instructors which administers affective surveys to students following course labs and synthesises the responses so instructors can identify students at risk rapidly. A secondary goal was to facilitate interactions between students and instructors pertaining to academic support.

Requirements were determined through a literature survey, client interviews, and analysis of existing software products. Following this, a web application was designed and implemented using the Symfony web framework. The current version contains several key and novel features. A configurable automatic flagging system determines students at risk by applying sentiment analysis to qualitative survey responses and identifying 2 dimensional grid responses falling in 'danger zones'. This latter aspect is a novel extension of Haden et al. and Parsons et al.'s affective survey research [19, 42]. In the survey instrument, a nudge UI encourages appropriately sized qualitative responses for sentiment analysis. Finally, the dual dashboard ensures transparency between stakeholders. It features built in documentation via the "Explain this page" interactive tours, a traffic light UI to quickly determine problematic responses, and response summaries at the course, lab, and student levels.

Automated and manual testing verified the current product against its requirements and user evaluations were undertaken to determine its validity. The software was received well by students, with appraisals ranging from positive to very positive. Instructors were more reserved, with appraisals ranging from neutral to positive, indicating that the software's interface and usage was overly complex. Despite this, several instructors commented on the utility and potential of the software. This indicates that the project shows promise with respect to its aims, though modifications and further evaluation is needed.

6.1 Future work

6.1.1 Addressing Complexity

The application's perspicuity UEQ scale scored lowest among both stakeholder groups, but it also had the largest variance. This suggests familiarity with dashboards may affect how a user perceives complexity. Future work should investigate progressive disclosure of features, as this makes software easier to learn and less error-prone [39]. For example, configuration forms for danger zones and automatic flagging thresholds could be deferred to a secondary screen for more adept users. Additional work may also focus on statistical analysis for more intuitive risk calculation requiring less configuration. For instance, rather than using a simple threshold to determine students at risk, they could be identified using a (configurable) upper percentile of a lab's risk factors. This would identify weaker students regardless of lab difficulty and reduce the configuration needed to 'hone in' on struggling students. Similar approaches could be explored for XY danger zones.

6.1.2 Evaluations

Further evaluations targeting CS1 undergraduates are needed to validate the project for its intended demographic. Additionally, larger sample sizes are required statistically significant evaluation results. To provide the student's perspective and improve instructors' understanding of the application, future instructor evaluation surveys could include a short video demonstration of student use or the user manual (see compressed folder's `/docs/user_manual.pdf`). Ultimately, longitudinal evaluations during a CS1 course with authentic usage and data would determine if the project truly meets its goals. Adding automated error, debug, and user interaction logging throughout the application would support analysis of such evaluations. Additionally, a mechanism for importing and/or synchronising user and course data would be needed. This could be achieved using .CSV data, or synchronisation with a university's LDAP server or learning management system, for example via Moodle's web service API [37]. A final concern for large evaluations and production deployment is efficiency. Presently, the course summary page load times can be very long when determining students at risk with large cohorts across many labs. This is because much of the implementation fetches data via bidirectional associations on entities, resulting in many small database hits which can bottleneck performance [15] (see figure 6.1). Reducing bidirectional associations into more involved, but less frequent, DQL or SQL queries in repository classes would drastically improve performance.

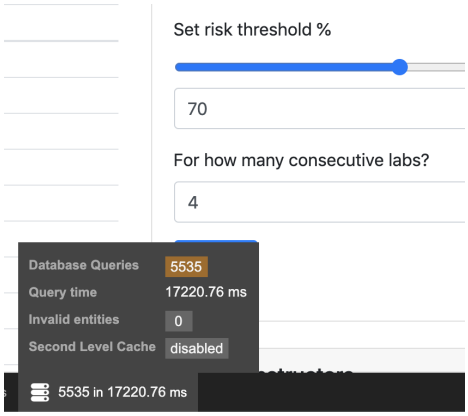


Figure 6.1: Significant load times caused by bi-directional associations on entities

6.1.3 Extensibility

While premature at this stage, a long term goal of the application is extensibility. The application was designed and implemented to allow for future customisation of surveys with little modification. The database was normalised so existing survey questions could be added to and removed from individual lab surveys. Additionally, survey forms were written using Symfony's form idiom, which easily allows modifying and substituting the underlying data structures. This foundation could be used by a future survey builder feature, allowing instructors to create and reuse surveys which align with their teaching style. Researchers could also experiment and assess different survey formats, as no validated affective instruments exist to date (see section 2.1.1). Finally, such a feature would open the application to courses and schools beyond CS.

Appendix A

Additional Documentation

Requirement	Description / Justification	Priority
An XY interface for querying students affective responses	The project intends to build on Parsons et al's proposals for future work, including further instrument validation, student facing dashboards, and interface refinements such as showing multiple responses on the class view [42]. Additionally, affective 'danger zones' show potential as an early indicator of student risk.	Must
Students can enter qualitative feedback	This would provide concrete insights into a student's experience, assisting with instructors' interpretations of quantitative responses and may provide insights into students' "auxiliary emotional load" [26]	Must
Qualitative feedback is analysed by sentiment analysis tools	Quantitative classifications of qualitative text can rapidly indicate students at risk. Without them, processing responses would be prohibitively time consuming.	Must
Instructors can create their own survey formats	Many affective surveying approaches remain unvalidated and further research may lead to differing implementations. Such a feature would facilitate adapting to such changes and allow instructors to modify the tool to suit their pedagogy.	Could
Qualitative responses can be automatically classified by topic	This would allow rapid identification of content concerning topics proven to indicate a student at risk. For example, poor time management [25, 44].	Would

Table A.1: Survey Instrument Requirements

Requirement	Description / Justification	Priority
Calculate a risk factor for students based on their lab's survey response	This will help lecturers identify and prioritise students in need of support, as well as forming the basis for automatic detection.	Must
Automatically flag students at risk at regular time intervals on a course by course basis	Automatic flagging is important, as manual identification of students at risk is often hindered by very high student-lecturer ratios [46]. This should reoccur at regular intervals to ensure support lists are up to date.	Must
Instructors can define and configure danger zones for question responses	Haden et al suggest that grid danger zones may be highly contextual. For example, the familiarity dimension is less pertinent at the beginning of a course [42]. Danger zones should be configurable to ensure the EWS accounts for this.	Should
Course thresholds for risk are configurable	Adjusting a course's risk calculation thresholds will allow instructors to experiment and adapt the EWS to the student cohort.	Should
Instructors can manually flag students at risk	Instructors may become aware of students at risk in contexts outside of the data captured by the system. Flagging the student would share such information easily with other instructors.	Could
Students can flag themselves as at risk	This feature was the second most desired feature in the student requirements survey (see the compressed folder's `/docs/requirements/student_survey_results.pdf`). Circumstances unique to the student may put them at risk and in such situations, students may feel like there are no support available [44]. This feature could reassure students and nudge them to report problems at an earlier stage [17].	Could

Table A.2: Early Warning System Requirements

Requirement	Description / Justification	Priority
Instructors can see a summary of all students at risk on a course.	This helps instructors quickly determine where to direct interventions.	Must
Students can see if they have been flagged as at risk for a course.	This ensures system transparency and helps students prioritise their studies. It was the most desired feature for students according to the student requirements survey (see the compressed folder's `/docs/requirements/student_survey_results.pdf`).	Must
Instructors can review an individual students' survey responses. Students can review their own. Multi-level representations indicate response risk.	Reviewing student responses in detail provides instructors with additional context for interventions. Additionally, 'traffic light' signals allow instructors identify problematic areas at a glance.	Should
Users are given a reason for why a student has been flagged.	Instructors can use such information to direct and prioritise interventions. It ensures transparency with students and allows them to act on or contest flags.	Should
Students can see graph summaries of their own lab responses over the course. Instructors can see all graph summaries.	Viewing trends over courses may assist students with self-regulation and time management. Instructors could use the information to review student progress and for course diagnostics, such as pacing of labs.	Would

Table A.3: Dual Dashboard Requirements

Requirement	Description / Justification	Priority
The system can be populated by course data from an external source	Information about courses, including instructors and students, is often held on existing university databases. Importing this into the software's database using common data formats (such as .CSV or .JSON) would reduce the administration required to setup the application.	Should
The system is synchronised with course data from an external source	This would prevent stale data being included on the EWS platform. It would also prevent potential security flaws from such users still having access.	Would

Table A.4: General Requirements

Requirement	Description / Justification	Priority
Availability - Interaction with the tool should not be bound to a specific time, location, or device.	Labs may take place in different parts of the university. Students indicated a preference for using their own devices over lab workstations (see the compressed folder's `/docs/requirements/student_survey_results.pdf`).	Must
Usability - Students should be able to rapidly access their dashboard and complete each survey	Survey length, timing and mode of access are factors which can affect student response rates [57]. 64% of students in the requirements survey indicated they would spend less than 4 minutes completing a survey (see the compressed folder's `/docs/requirements/student_survey_results.pdf`).	Must
Usability - Instructors should be able to determine students at risk rapidly with little cognitive load	Due to the time pressures put upon instructors, rapid identification of students at risk is a priority.	Must
Usability - Clear documentation and descriptions are provided in-app to ensure users can understand the tool and its operation	"Perceived ease-of-use is a significant factor that influences the adoption of learning analytics tools" [23]. Moreover, instructors struggle to adopt opaque systems [11]. Clear, accessible instructions would address these concerns.	Should
Privacy & Compliance - Users have access only to information which pertains to their role	Due to the sensitive nature of the survey responses and student details, it is imperative that data is only shared to those to whom it pertains. For example, an instructor should only be able to view survey data given by students they teach.	Must
Privacy & Compliance - The application is GDPR compliant	As the system will be used in the UK, it should be compliant with data protection regulations. This includes displaying compliance notices and opt-in consent forms.	Could
Privacy & Compliance - Different types of instructor have different security privileges	Course administrators, lecturers, tutors and academic support staff may all use the system. Specialist retention staff may wish to have access to all course instances, whereas lecturers should only see the courses they teach.	Would
Extensibility - The system should be extendable to other courses and schools	Interventions based on affective surveys may be effective in contexts outside CS1. A flexible implementation would open the tool up to other contexts.	Would

Table A.5: Non-Functional Requirements.

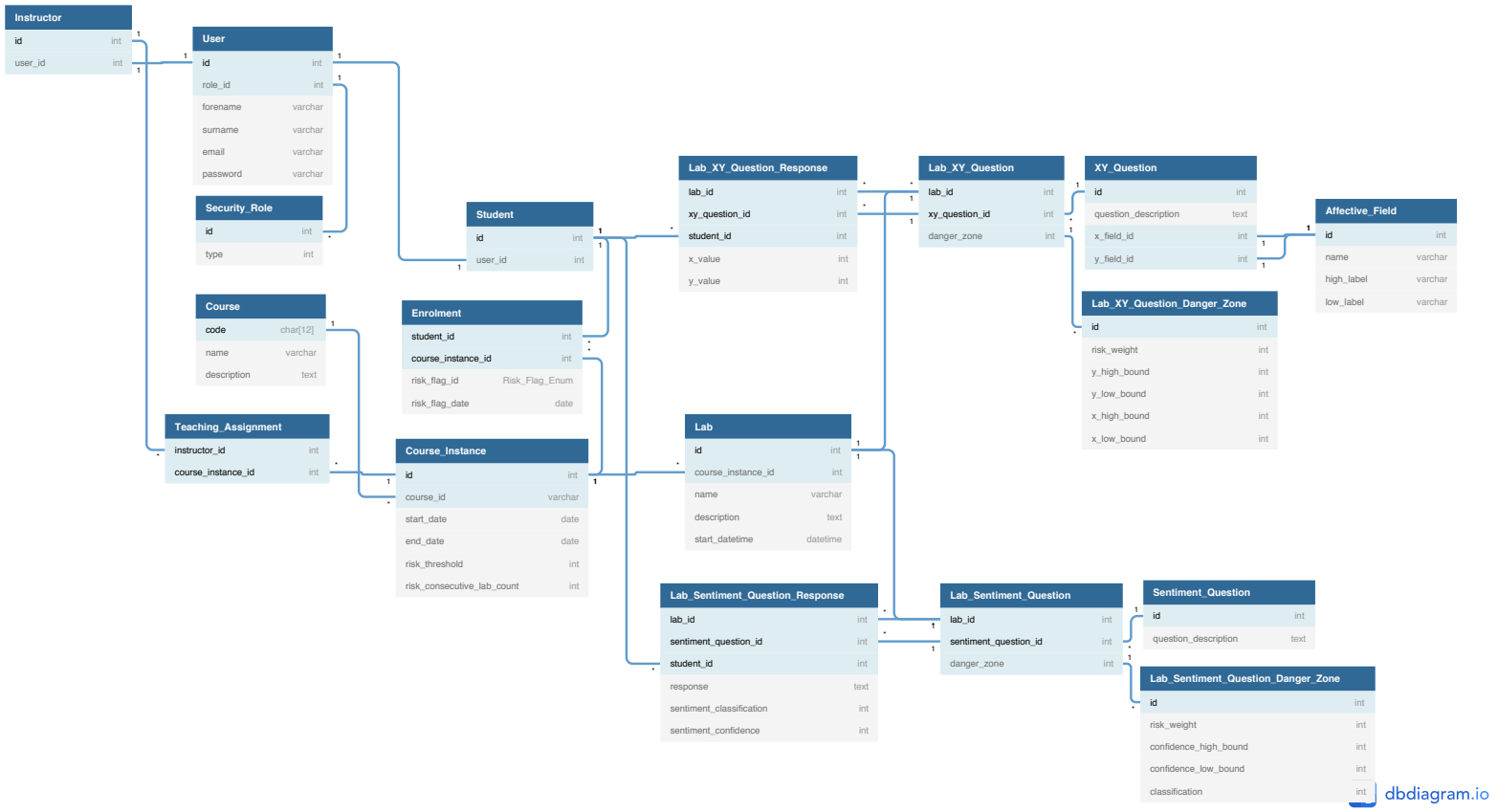


Figure A.1: Relational Schema

Library	Source / Attribution	Usecase	Project-specific modifications / configurations (with compressed folder paths)
TwigBundle and TwigExtraBundle	Symfony	View template engine. Provide format_time filter to show *DateTime* times only	Custom *renderRisk* function (`/src/Twig/AppExtension`). Link to custom form template partials and global icons (`/config/packages/twig.yaml`).
DoctrineBundle	Symfony	Provide ORM modules	
SecurityBundle	Symfony	Provide security modules and encryption	Access control to routes (`/config/packages/security.yaml`). Custom *Voter* classes (`/src/Security/Voter/`).
DoctrineFixturesBundle	Symfony	Create application fixtures	*Fixture* classes (`/src/DataFixtures/`).
WebpackEncoreBundle	Symfony	Module bundling for JS and CSS	Custom entry points and SASS and Vue.js loader configuration (`webpack.config.js`).
DAMADoctrineTestBundle	David Maicher [31]	Create and rollback database transactions for each functional testcase.	Integrated with service container in *FunctionalTestCase* (`tests/Functional/`).
RewieerTaskSchedulerBundle	Anthony Cyrille [13]	Schedule CRON tasks in Symfony services	*FlagStudentsTask* class (`/src/Task/`).
fzaninotto/faker	Francois Zaninotto [67]	Populate fixtures with fake data.	
monkeylearn/monkeylearn-php	MonkeyLearn [36]	Client library for making sentiment analysis API requests	
Vue.js	Evan You [66]	Encapsulate HTML, JS and CSS into components	Project *.vue* components (`/assets/js/vue/`).
Bootstrap 4	Bootstrap [9]	Provide base CSS classes and components	
shepherd.js	ShipShape [59]	Implement 'tour' documentation using javascript components	Configuration and global function access in *tour.js* module (`/assets/js/lib/`). Implemented in *.twig* page templates (`/templates/`).
tablesort.js	Tristen Brown [12]	Table sorting functionality	Additional sort filters provided in javascript modules (`/assets/js/lib/tablesort`).

Table A.6: Third party software libraries, with attributions, descriptions of their usecases, and project-specific modifications/configurations.

Appendix B

Additional Figures

```php

    private function canView(CourseInstance $courseInstance, User $user)
    {
        // If they can edit, they can view
        if ($this->canEdit($courseInstance, $user)) {
            return true;
        }

        // If the user is a student, are they enrolled?
        if ($user->hasRole(Roles::STUDENT)) {
            return $courseInstance->getEnrolments()->exists(function ($key, Enrolment $enrolment) use ($user) {
                return $enrolment->getStudent() === $user->getStudent();
            });
        }

        return false;
    }

    private function canEdit(CourseInstance $courseInstance, User $user)
    {
        // Instructors who teach the course can edit
        if ($user->hasRole(Roles::INSTRUCTOR)) {
            return $courseInstance->getInstructors()->contains($user->getInstructor());
        }
    }

```

Figure B.1: CourseInstanceVoter methods which determine user access when passed a CourseInstance object

```twig
{# A custom template for rendering the xy_coordinates form widget #}
{% block xy_coordinates_widget %}

    {# Render hidden fields #}
    {% for child in form.children|filter(child => not child.rendered) %}
        {{ form_widget(child) }}
    {% endfor %}

    {# Variable with widget ID #}
    {% set widgetId = id ~ '_xywidget' %}

    {# Dom element for JS Component #}
    <div id="{{widgetId}}">Loading...</div>      You, a few seconds ago • Uncommitted changes

    <script>
    {% set jsonFieldLabel = constant('App\\Form\\Type\\AbstractXYComponentType::JSON_FIELD') %}
    {% set jsonElement = form[jsonFieldLabel].vars.id %}

    // Waits until js and everything else is ready and loads an instance of the XY selector.
    // This is required because this <script> tag will be inserted in line.
    addLoadEvent(() => {
        const jsonField = {{jsonElement}};

        XYQuestionWidgetFactory('#{{ widgetId }}', {
            // Set mode to 'point', i.e. only one sell can be selected
            mode: 'point',
            points: {
                // If data is provided in a form option, populate it here. Otherwise it is bound to the j
                // Note: this data is a 'point'
                data:{% if initial_data %}{{ initial_data|raw }}{% else %}JSON.parse(jsonField.value){% en
                onChange: (points) => { jsonField.value = JSON.stringify(points); }
            },
            xLabelHigh: '{{ x_label_high }}',
            xLabelLow: '{{ x_label_low }}',
            yLabelHigh: '{{ y_label_high }}',
            yLabelLow: '{{ y_label_low }}',
            cellSizeInRem: {{ cell_size }}
        });
    });
    </script>
{% endblock %}
```

Figure B.2: The twig partial used to idiomatically use the XYQuestion Vue component with Symfony's form API. Note how JSON is bound to the hidden form field via callback functions and the *addLoadEvent* function is used to avoid page blocking.

```php
    public function allRisksAboveFalseProvider()
    {
        yield [[1.0, 0.2, 0.7], 0.21];
        yield [[1.7, 3.5, 2.5], 4.0];
        yield [[-0.1, -0.1, 0.1], 0.1];
        yield [[], 0.1]; // No risk responses
    }

    /**
     * @dataProvider allRisksAboveFalseProvider
     */
    public function testAreAllRisksAboveFalse(array $weightedRiskFactors, float $areAbove)
    {
        $mockLabResponseRisks = $this->createLabResponseRiskMocks($weightedRiskFactors);
        $mockEnrolment = $this->createMock(Enrolment::class);
        $enrolmentRisk = new EnrolmentRisk($mockLabResponseRisks, $mockEnrolment);
        $this->assertFalse($enrolmentRisk->areAllRisksAbove($areAbove));
    }
```

Figure B.3: Unit test on the EnrolmentRisk class using mocks and a data provider function to assert correct risk calculation

```php
    /**
     * Creates the course instance under test before each test case
     */
    protected function setUp()
    {
        $creator = $this->getEntityCreator();
        $dateTimeProvider = new DateTimeProvider();
        $courseStart =  ($dateTimeProvider->getCurrentDateTime()->modify("- 1 week"));
        $courseEnd = ($dateTimeProvider->getCurrentDateTime()->modify("+ 1 week"));
        $testCourseOne = $creator->createCourse('CS101','Programming',null);
        $courseInstance = $creator->createCourseInstance($testCourseOne,$courseStart,$courseEnd);
        $courseInstance->setIndexInCourse(1);
        $this->courseInstance = $courseInstance;
    }

    public function testAnonymousUserRedirected()
    {…
    }

    public function testNonMemberStudentDenied()
    {
        $creator = $this->getEntityCreator();
        $testNonMemberStudent = $creator->createStudent('testFirstname','testSurname','2345678');
        $client = static::createClient();
        $client->loginUser($testNonMemberStudent->getUser());
        $client->request('GET', '/courses/CS101/1');
        $this->assertResponseStatusCodeSame(403);
    }
```

Figure B.4: Functional test testing non-member students are denied access to a course summary page. Note the specific database state created prior to each test and interactions with the browser via the $client object.

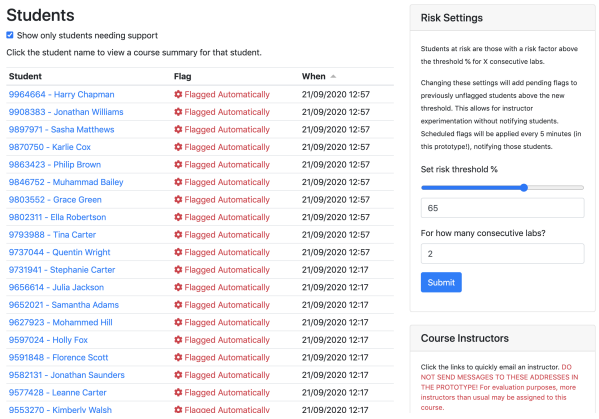


Figure B.5: A course state which shows an evaluation user accidentally having flagged all students. This is possible when experimenting with configurations while the flag students cron task runs simultaneously.

Bibliography

[1] Mireilla Bikanga Ada and Katarina Turinicova. Developing a Dual Dashboard Early Detection System. In *IEEE International Conference on Advanced Learning Technologies (ICALT)*, pages 155–157, 2020.

[2] Ángel F. Agudo-Peregrina, Santiago Iglesias-Pradas, Miguel Ángel Conde-González, and Ángel Hernández-García. Can we predict success from log data in VLEs? Classification of interactions for learning analytics and their relation with performance in VLE-supported F2F and online learning. *Computers in Human Behavior*, 31(1):542–550, 2014.

[3] Ángel F. Agudo-Peregrina, Santiago Iglesias-Pradas, Miguel Ángel Conde-González, and Ángel Hernández-García. Can we predict success from log data in VLEs? Classification of interactions for learning analytics and their relation with performance in VLE-supported F2F and online learning. *Computers in Human Behavior*, 31(1):542–550, 2014.

[4] Nabeela Altrabsheh, Mihaela Cocea, and Sanaz Fallahkhair. Learning Sentiment from Students' Feedback for Real-Time Interventions in Classrooms. In Abdelhamid Bouchachia, editor, *Adaptive and Intelligent Systems*, pages 40–49, Cham, 2014. Springer International Publishing.

[5] Ryan S.J.d. Baker, Sidney K. D'Mello, Ma Mercedes T. Rodrigo, and Arthur C. Graesser. Better to be frustrated than bored: The incidence, persistence, and impact of learners' cognitive-affective states during interactions with three different computer-based learning environments. *International Journal of Human Computer Studies*, 68(4):223–241, 2010.

[6] F F Balahadia, M C G Fernando, and I C Juanatas. Teacher's performance evaluation tool using opinion mining with sentiment analysis. In *2016 IEEE Region 10 Symposium (TENSYMP)*, pages 95–98, may 2016.

[7] Theresa Beaubouef and John Mason. Why the high attrition rate for computer science students. *ACM SIGCSE Bulletin*, 37(2):103–106, jun 2005.

[8] Adam Bermingham and Alan Smeaton. Classifying sentiment in microblogs: Is brevity an advantage? *International Conference on Information and Knowledge Management, Proceedings*, pages 1833–1836, 2010.

[9] Bootstrap. Bootstrap · the most popular html, css, and js library in the world. `https://getbootstrap.com/`. (Accessed on 10/12/2020).

[10] Nigel Bosch and Sidney D'Mello. The Affective Experience of Novice Computer Programmers. *International Journal of Artificial Intelligence in Education*, 27(1):181–206, 2017.

[11] Michael Brown. Seeing students at scale: how faculty in large lecture courses act upon learning analytics dashboard data. *Teaching in Higher Education*, 25(4):384–400, may 2020.

[12] Tristen Brown. tristen/tablesort: A small tablesorter in plain javascript. `https://github.com/tristen/tablesort`. (Accessed on 10/12/2020).

[13] Anthony Cyrille. rewieer/taskschedulerbundle: Task scheduler with cron for symfony. `https://github.com/rewieer/TaskSchedulerBundle`. (Accessed on 10/12/2020).

[14] V Dhanalakshmi, D Bino, and A M Saravanan. Opinion mining from student feedback data using supervised learning algorithms. In *2016 3rd MEC International Conference on Big Data and Smart City (ICBDSC)*, pages 1–5, mar 2016.

[15] Doctrine. Doctrine: Php open source project. `https://www.doctrine-project.org/index.html`. (Accessed on 10/11/2020).

[16] Martin Fowler. *Patterns of Enterprise Application Architecture*. Addison-Wesley Longman Publishing Co., Inc., USA, 2002.

[17] John Fritz. Using Analytics to Nudge Student Responsibility for Learning. *New Directions for Higher Education*, 2017(179):65–75, 2017.

[18] Michail N. Giannakos, Ilias O. Pappas, Letizia Jaccheri, and Demetrios G. Sampson. Understanding student retention in computer science education: The role of environment, gains, barriers and usefulness. *Education and Information Technologies*, 22(5):2365–2382, oct 2017.

[19] Patricia Haden, Dale Parsons, Joy Gasson, and Krissi Wood. Student affect in CS1: Insights from an easy data collection tool. In *ACM International Conference Proceeding Series*, pages 40–49, New York, New York, USA, nov 2017. Association for Computing Machinery.

[20] HESA. Percentage of UK domiciled undergraduate entrants who are no longer in HE by subject and entry qualification 2015/16 to 2017/18, mar 2019.

[21] Penelope M. Huang and Suzanne Gage Brainard. Identifying determinants of academic self confidence among science, math, engineering, and technology students. *Journal of Women and Minorities in Science and Engineering*, 7(4):24, 2001.

[22] Dirk Ifenthaler and Chathuranga Widanapathirana. Development and validation of a learning analytics framework: Two case studies using support vector machines. *Technology, Knowledge and Learning*, 19(1-2):221–240, 2014.

[23] Ioana Jivet, Maren Scheffel, Marcus Specht, and Hendrik Drachsler. License to evaluate: Preparing learning analytics dashboards for educational practice. In *Proceedings of the 8th International Conference on Learning Analytics and Knowledge*, LAK '18, page 31–40, New York, NY, USA, 2018. Association for Computing Machinery.

[24] Kyle M. L. Jones and Chase McCoy. Reconsidering data in learning analytics: opportunities for critical research using a documentation studies framework. *Learning, Media and Technology*, 44(1):52–63, jan 2019.

[25] Päivi Kinnunen and Lauri Malmi. Why students drop out CS1 course? In *ICER 2006 - Proceedings of the 2nd International Computing Education Research Workshop*, volume 2006, pages 97–108, 2006.

[26] Päivi Kinnunen and Beth Simon. Experiencing programming assignments in CS1: The emotional toll. In *ICER'10 - Proceedings of the International Computing Education Research Workshop*, pages 77–85, New York, New York, USA, 2010. ACM Press.

[27] Bettina Laugwitz, Theo Held, and Martin Schrepp. Construction and evaluation of a user experience questionnaire. In Andreas Holzinger, editor, *HCI and Usability for Education and Work*, pages 63–76, Berlin, Heidelberg, 2008. Springer Berlin Heidelberg.

[28] Philipp Leitner, Mohammad Khalil, and Martin Ebner. Learning analytics in higher education—a literature review. In *Studies in Systems, Decision and Control*, volume 94, pages 1–23. Springer International Publishing, 2017.

[29] Alex Lishinski, Aman Yadav, and Richard Enbody. Students' emotional reactions to programming projects in introduction to programming: Measurement approach and influence on learning outcomes. In *ICER 2017 - Proceedings of the 2017 ACM Conference on International Computing Education Research*, pages 30–38, New York, NY, USA, aug 2017. Association for Computing Machinery, Inc.

[30] Martín Liz-Domínguez, Manuel Caeiro-Rodríguez, Martín Llamas-Nistal, and Fernando A. Mikic-Fonte. Systematic literature review of predictive analysis tools in higher education. *Applied Sciences (Switzerland)*, 9(24), 2019.

[31] David Maicher. dmaicher/doctrine-test-bundle: Symfony bundle to isolate your app's doctrine database tests and improve the test performance. `https://github.com/dmaicher/doctrine-test-bundle`. (Accessed on 10/12/2020).

[32] Aigul Mavletova and Mick P. Couper. Mobile web survey design: Scrolling versus paging, sms versus e-mail invitations. *Journal of Survey Statistics and Methodology*, 2(4):498–518, dec 2014.

[33] Dawn McKinney and Leo F. Denton. Houston, we have a problem: There's a leak in the CS1 affective oxygen tank. In *SIGCSE Bulletin (Association for Computing Machinery, Special Interest Group on Computer Science Education)*, volume 36, pages 236–239, mar 2004.

[34] Mingnan Liu. Pros and cons of scrolling and multiple pages in surveys — SurveyMonkey.

[35] Brojo Kishore Mishra and Abhaya Kumar Sahoo. Evaluation of Faculty Performance in Education System Using Classification Technique in Opinion Mining Based on GPU. In Himansu Sekhar Behera and Durga Prasad Mohapatra, editors, *Computational Intelligence in Data Mining—Volume 2*, pages 109–119, New Delhi, 2016. Springer India.

[36] MonkeyLearn. Monkeylearn - text analysis. `https://monkeylearn.com/`. (Accessed on 10/12/2020).

[37] Moodle. Web service api functions - moodledocs. `https://docs.moodle.org/dev/Web_service_API_functions`. (Accessed on 10/13/2020).

[38] Myriam Munezero, Calkin Montero, Maxim Mozgovoy, and Erkki Sutinen. Exploiting sentiment analysis to track emotions in students' learning diaries , 2013.

[39] Jakob Nielsen. Progressive disclosure. `https://www.nngroup.com/articles/progressive-disclosure/`, 12 2006. (Accessed on 10/13/2020).

[40] Matthew W. Ohland, Sheri D. Sheppard, Gary Lichtenstein, Ozgur Eris, Debbie Chachra, and Richard A. Layton. Persistence, engagement, and migration in engineering programs. *Journal of Engineering Education*, 97(3):259–278, 2008.

[41] Ilias O. Pappas, Michail N. Giannakos, and Letizia Jaccheri. Investigating factors influencing students intention to dropout computer science studies. In *Annual Conference on Innovation and Technology in Computer Science Education, ITiCSE*, volume 11-13-July, pages 198–203, New York, New York, USA, jul 2016. Association for Computing Machinery.

[42] Dale Parsons, Krissi Wood, Joy Gasson, and Adon C.M. Moskal. Development of a Self-Reporting Tool for Capturing Student Emotions during Programming Activities. In *ACM International Conference Proceeding Series*, pages 64–68, New York, New York, USA, jan 2019. Association for Computing Machinery.

[43] A. Pears, P. East, R. McCartney, M. Ratcliffe, I. Stamouli, A. Berglund, Päivi Kinnunen, Jan-Erik Moström, C. Schulte, A. Eckerdal, L. Malmi, L. Murphy, B. Simon, and L. Thomas. What's the problem?: teachers' experience of student learning successes and failures. 2007.

[44] Andrew Petersen, Michelle Craig, Jennifer Campbell, and Anya Tafliovich. Revisiting why students drop CS1. *ACM International Conference Proceeding Series*, pages 71–80, 2016.

[45] Qualtrics. Qualtrics - leading experience management & survey software. `https://www.qualtrics.com/uk/?rid=ip&prevsite=en&newsite=uk&geo=GB&geomatch=uk`. (Accessed on 10/13/2020).

[46] Keith Quille and Susan Bergin. CS1: how will they do? How can we help? A decade of research and practice. *Computer Science Education*, 29(2-3):254–282, jul 2019.

[47] S. Rani and P. Kumar. A Sentiment Analysis System to Improve Teaching and Learning. *Computer*, 50(5):36–43, may 2017.

[48] Bart Rienties, Christothea Herodotou, Tom Olney, Mat Schencks, and Avi Boroowa. Making sense of learning analytics dashboards: A technology acceptance perspective of 95 teachers. *International Review of Research in Open and Distance Learning*, 19(5):187–202, 2018.

[49] Martin Schrepp. *User Experience Questionnaire Handbook*, 09 2015. (Accessed on 10/13/2020).

[50] Martin Schrepp, Andreas Hinderks, and Jörg Thomaschewski. Construction of a benchmark for the user experience questionnaire (ueq). *International Journal of Interactive Multimedia and Artificial Intelligence*, 4:40–44, 06 2017.

[51] Clara Schumacher and Dirk Ifenthaler. Features students really expect from learning analytics. *Computers in Human Behavior*, 78:397–407, 2018.

[52] Clara Schumacher and Dirk Ifenthaler. The importance of students' motivational dispositions for designing learning analytics. *Journal of Computing in Higher Education*, 30(3):599–619, dec 2018.

[53] Beat A. Schwendimann, Maria Jesus Rodriguez-Triana, Andrii Vozniuk, Luis P. Prieto, Mina Shirvani Boroujeni, Adrian Holzer, Denis Gillet, and Pierre Dillenbourg. Perceiving learning at a glance: A systematic literature review of learning dashboard research, jan 2017.

[54] SensioLabs. Doctrinefixturesbundle (symfony bundles docs). `https://symfony.com/doc/master/bundles/DoctrineFixturesBundle/index.html`. (Accessed on 10/13/2020).

[55] SensioLabs. What is Symfony?

[56] Mahsood Shah, Chenicheri Sid Nair, and John T.E. Richardson. Chapter 10 - closing the loop: An essential part of student evaluations. In Mahsood Shah, Chenicheri Sid Nair, and John T.E. Richardson, editors, *Measuring and Enhancing the Student Experience*, pages 113 – 122. Chandos Publishing, 2017.

[57] Mahsood Shah, Chenicheri Sid Nair, and John T.E. Richardson. Chapter 5 - engaging students and staff in feedback and optimising response rates. In Mahsood Shah, Chenicheri Sid Nair, and John T.E. Richardson, editors, *Measuring and Enhancing the Student Experience*, pages 47 – 58. Chandos Publishing, 2017.

[58] Esther Shein. The CS teacher shortage. *Communications of the ACM*, 62(10):17–18, sep 2019.

[59] ShipShape. Shepherd — guide your users through a tour of your app. `https://shepherdjs.dev/`. (Accessed on 10/12/2020).

[60] G Siemens. 1st International Conference on Learning Analytics and Knowledge 2011 — Connecting the technical, pedagogical, and social dimensions of learning analytics, feb 2011.

[61] Ian Sommerville. *Software Engineering*. Pearson Education Limited, Harlow, 10 edition, 2016.

[62] SymfonyCasts. Database tricks on symfonycloud ¿ blackfire.io: Revealing performance secrets with profiling — symfonycasts. `https://symfonycasts.com/screencast/blackfire/sf-cloud-database`. (Accessed on 10/13/2020).

[63] Laura Toma and Jan Vahrenhold. Self-efficacy, cognitive load, and emotional reactions in collaborative algorithms labs - A case study. In *ICER 2018 - Proceedings of the 2018 ACM Conference on International Computing Education Research*, volume 10, pages 1–10, New York, NY, USA, aug 2018. Association for Computing Machinery, Inc.

[64] UK Commission for Employment and Skills. Working Futures 2014-2014 Headline Report. Technical report, UK Commission for Employment and Skills, apr 2014.

[65] UK Commission for Employment and Skills. High level STEM skills requirements in the UK labour market. Technical report, UK Commission for Employment and Skills, 2015.

[66] Evan You. Vue.js. `https://vuejs.org/`. (Accessed on 10/12/2020).

[67] Francois Zaninotto. fzaninotto/faker: Faker is a php library that generates fake data for you. `https://github.com/fzaninotto/Faker`. (Accessed on 10/12/2020).