**Functional Specifications**

**for the ReCode MVP**

****

**Version 0.1.2**

| **VERSION HISTORY** | | | | |
| --- | --- | --- | --- | --- |
| **VERSION** | **APPROVED BY** | **REVISION DATE** | **DESCRIPTION OF CHANGE** | **AUTHOR** |
| 0.0.1 |  | 18/08/2025 | Introduction | Tumi Phororo |
| 0.0.2 |  | 20/08/2025 | General description | Tumi Phororo |
| 0.0.3 |  | 22/08/2025 | External Interface Requirements | Sinovuyo Sondara |
| 0.0.4 |  | 24/08/2025 | Functional Requirements | Lwazi Lata |
| 0.1.1 |  | 27/08/2025 | Performance Requirements | Sinovuyo Sondara, Kayla Supra |
| 0.1.2 |  | 8/30/2025 | Formatting and Minor Modifications | Brandon van Vuuren |
|  |  |  |  |  |

**Authorization Memorandum**  
I have reviewed the *Functional Specifications Document for ReCode MVP*.

**Management Certification – please check one:**

* ☐ The document is accepted.
* ☐ The document is accepted pending the changes noted.
* ☐ The document is not accepted.

We authorise the initiation and continuation of work on this system, subject to the decisions indicated above.



Table of Contents

[1 INTRODUCTION 4](#_Toc207488880)

[1.1 Purpose 4](#_Toc207488881)

[1.2 Reference Documents 4](#_Toc207488882)

[1.3 Abbreviations and Acronyms 4](#_Toc207488883)

[1.4 Document Conventions 5](#_Toc207488884)

[2 GENERAL DESCRIPTION 6](#_Toc207488885)

[2.1 Product Context 6](#_Toc207488886)

[2.2 User Classes and Characteristics 6](#_Toc207488887)

[ Users: 6](#_Toc207488888)

[ Administrators 6](#_Toc207488889)

[2.3 Overview of Functional Requirements 7](#_Toc207488890)

[2.4 Overview of Data Requirements 8](#_Toc207488891)

[2.5 Operating Environment 8](#_Toc207488892)

[2.6 General Constraints, Assumptions, Dependencies, Guidelines 9](#_Toc207488893)

[2.7 Design and Implementation Constraints 9](#_Toc207488894)

[2.8 User Documentation 9](#_Toc207488895)

[3 REQUIREMENTS 10](#_Toc207488896)

[3.1 External Interface Requirements 10](#_Toc207488897)

[3.1.1 User Interfaces 10](#_Toc207488898)

[3.1.2 Hardware Interfaces 10](#_Toc207488899)

[3.1.3 Software Interfaces 10](#_Toc207488900)

[3.1.4 Communications Interfaces 11](#_Toc207488901)

[3.2 Functional Requirements 11](#_Toc207488902)

[3.3 Performance Requirements 13](#_Toc207488903)

[3.4 Security 14](#_Toc207488904)

[3.5 Usability 15](#_Toc207488905)

[3.6 Scalability 15](#_Toc207488906)

[ 3.6.1 System Architecture Expansion 16](#_Toc207488907)

[ 3.6.2 Data Storage Expansion 16](#_Toc207488908)

[ 3.6.3 Code Execution and Grading Scalability 16](#_Toc207488909)

[ 3.6.4 Data Analytics and Reporting Scalability 16](#_Toc207488910)

[ 3.6.5 User Growth and Concurrency 16](#_Toc207488911)

[ 3.6.6 Monitoring, Logging, and Fault Tolerance 17](#_Toc207488912)

[ 3.6.7 Future Scalability Considerations 17](#_Toc207488913)

[References 18](#_Toc207488914)

# INTRODUCTION

## Purpose

This document outlines all the functional specifications for the development of the ReCode project that the team will be building. It provides comprehensive information on how the project will be implemented, highlighting details specific to the client and their needs. It also includes the specific functionalities of the system itself, as well as its benefits to the client and the target users who will be using it. Additionally, it includes the scope, as well as the data requirements for the development of ReCode. The Minimum Viable Product (MVP) will fulfil requirements such as:

* Updating and selection of modules per semester,
* Uploading of module content on the admin side,
* Built-in coding challenges for students to participate,
* Student data and performance tracking,
* Challenge participation on the student side,
* Data storage and extraction on the admin side.

The system will use industry-standard communication protocols and a Python program on Windows to simulate interactions and data transmission.

## Reference Documents

* ReCode Technical Specifications
* ReCode Business Case
* ReCode Feasibility Study

## Abbreviations and Acronyms

* MVP – Minimum Viable Product
* IT – Information Technology
* API – Application Programming Interface
* HTTP/HTTPS – Hypertext Transfer Protocol / Secure
* POPIA – Protection of Personal Information Act
* JWT – JSON Web Token (used for auth)
* RBAC – Role-Based Access Control (used for permissions)
* UI – User Interface
* UX – User Experience
* DB – Database (PostgreSQL via Supabase)
* NLP – Natural Language Processing (topic extraction)
* OCR – Optical Character Recognition (possible future slide parsing)
* CI/CD – Continuous Integration / Continuous Deployment (pipeline, GitHub Actions)
* VM – Virtual Machine
* ELO – Rating system for student progression
* SLA – Service-Level Agreement
* TLS – Transport Layer Security (HTTPS requirement)
* HSTS – HTTP Strict Transport Security (web security)
* CLI – Command-Line Interface
* Judge0 – Open-source code execution API
* Supabase – Backend-as-a-service (Auth, DB, Storage; no need to expand, just define once)
* Redis – In-memory data store (again, define once)
* Celery – Task queue system (define once)

## 1.4 Document Conventions

To ensure consistency and readability, the following conventions are applied throughout this document:

* **Font**: All text is written in **Arial**.
* **Headings**:
  + Main headings (e.g. *1, 2, 3*) are in **bold Arial, size 12 pt**.
  + Sub-headings (e.g. *1.1, 1.2*) are in **bold Arial, size 11 pt**.
  + Sub-sub-headings (e.g. *1.1.1*) are in ***italic Arial, size 11 pt****.*
* **Body text**: Standard content is in **Arial, size 11 pt**, normal weight.
* **Lists**:
  + **Numbered lists** indicate steps or ordered processes.
  + **Bulleted lists** indicate unordered groupings of related items.
* **Tables**: Used for structured data such as requirements, version history, and acronyms. Column headers are bold.
* **Emphasis**: Important terms are shown in **bold**. Technical identifiers (e.g. function names, API endpoints, code snippets) are written in *monospace*.
* **Figures and Diagrams**: Labelled sequentially (e.g. *Figure 1: ERD for User Module*) with captions beneath in Arial 10 pt.
* **Page numbering**: Displayed in the bottom-right corner of each page.
* **Document references**: External documents are cited in-text using title-case and listed in the References section.
* **Change tracking**: All revisions are logged in the Version History table at the front of the document.

# GENERAL DESCRIPTION

## Product Context

We will be bringing a software system that goes by the name ReCode. This system is aimed at helping first-year students who are new to the world of IT, specifically with the practice of programming. It gives students with or without previous programming knowledge and experience an opportunity to learn coding through a more practical and engaging way of learning. Students often find it difficult to learn coding because it is usually taught in a more theoretical way rather than encouraging the students to engage with the module by practising it themselves, while testing themselves and having fun while they’re at it, so ReCode aims to provide this way of learning to the students.

## User Classes and Characteristics

### Users:

The users will mainly be the students; they will be utilising the system when participating in the coding challenges. They will be interacting with the system on the user interface, where they will be doing the challenges. They will also be able to view their performance through the marks they receive since the system automatically marks the students’ work. Based on the students’ performance, they receive different badges, which is an indication of what skill level the student is at, and they can view this and see where they can improve and sharpen their skills. This also incentivises and encourages the student to improve and receive the badges.

Another user for the system is lecturers. They will be handling the lecture side of the system where they are able to upload or select their module at the beginning of each semester. They will be uploading the slides at the beginning or at the end of each lecture that will be covering the work that was done in class on that day for the students to have access after class. Even though the system can identify topics for the coding challenges from the uploaded slides, lectures can review and correct these topics. The lectures can view the aggregated performance of the students, based on metrics such as the points they receive, and to a limited extent, individual student progress. The lecturer is also able to extract all the data collected throughout the semester to then improve the module content and the way of teaching, based on how the students perform.

### Administrators

ReCode distinguishes between two types of administrators:

1. **Module Administrators (Lecturers/HODs)**
   * Responsible for managing the academic side of the system.
   * Upload or select their module at the start of each semester.
   * Upload lecture slides before or after class so students can access content.
   * Monitor student performance using metrics such as marks, completion time, and badges earned.
   * Extract aggregated performance data at semester end to improve teaching and module content.
   * Create and manage course instances each semester, including the schedule of topics.
   * Add, edit, or remove coding challenges from the database, tagging them with appropriate topics and difficulty.
2. **System Administrators**
   * Responsible for the technical and operational side of the platform.
   * Create or deactivate user accounts and adjust user roles (e.g., student ↔ lecturer).
   * View and maintain system logs and metrics.
   * Manage platform configuration tasks such as updating API keys.
   * Provide technical support, including resetting student progress when required.

## Overview of Functional Requirements

* **Weekly Challenge Delivery:** The system automatically releases a new coding challenge to students each week based on the week's topic from the lecture slides.
* **Lecture Content Integration:** Enables lecturers to upload their course slides, which are then used by the system to help generate coding challenges.
* **Automated Topic Tagging (NLP):** Identifies key programming topics and concepts from the uploaded lecture content.
* **Challenge Database & Retrieval:** This component manages and retrieves coding challenges, ensuring they align with identified topics.
* **Code Submission & Judge0 Evaluation:** Automatically evaluates students' submitted coding solutions and provides them with instant feedback.
* **User Authentication & Roles:** Validates users' log-in details and provides them with different access to the system based on their roles.
* **Gamification Mechanics:** A reward system using tools like points, ranks, and badges based on a students’ performance on the coding challenges.
* **Student Progress Dashboard:** Provides students with a dedicated view of their personal progress, including scores, ranks, and collected achievements.
* **Admin/Lecturer Basic Tools:** Enables system administrators to have essential capabilities for platform management, such as assigning user roles and overseeing general data.
* **Lecture Analytics View**: The lecture can view an overview of the participation of students for each challenge
* **Lecture Customisation**: Lectures can enter module information, and can control the challenge topics

## Overview of Data Requirements

ReCode’s database is designed to be dynamic and self-updating, ensuring the platform can operate sustainably without constant manual intervention. Challenge data is not fixed; instead, the repository grows and adapts over time. One pipeline leverages Hugging Face models to generate or adapt coding questions based on lecturer slide content, automatically updating the challenge library. Additional pipelines supplement this by drawing from pre-curated question banks, lecturer-authored challenges, and open-source repositories.

This multi-source design prevents dependency on a single mechanism, ensures pedagogical diversity, and allows the system to remain responsive to new topics, modules, or curriculum changes. Automated updates occur asynchronously through background tasks (Redis + Celery), so that new challenges can be inserted without interrupting live student usage. Audit logs track every insertion, whether AI-generated or human-curated, preserving transparency and enabling lecturers to validate the material.

By maintaining this self-sustaining and dynamic structure, ReCode can continue to evolve alongside the academic programme, much like established LMS platforms, but with an emphasis on interactive coding practice and automated feedback.

## Operating Environment

ReCode will operate as a web-based platform, accessible through any modern standards-compliant web browser (e.g., Chrome, Firefox, Edge, Safari) on desktop and laptop devices. The system is optimised for online delivery, ensuring students and lecturers can access it securely from campus or remote locations without requiring specialised software installation. The platform will run on a cloud-hosted environment, ensuring scalability, availability, and fault tolerance. Services will be deployed within a containerised infrastructure, making the application portable across managed hosting providers. All client–server communication will take place over encrypted HTTPS/TLS connections. The Minimum Viable Product (MVP) is restricted to online use only, relying on stable internet connectivity for both student interaction and lecturer content management. Offline or desktop-based clients may be considered in future iterations, but they remain out of scope for the MVP.

## General Constraints, Assumptions, Dependencies, Guidelines

Assumptions:

* Students will have internet access.
* Lectures will manage and monitor students’ work as needed.
* Students have basic knowledge on how to use simple coding interface.

Constraints:

* The system can only work if you have internet access.
* Time constraints are limited to the module timeline and so there are specific deadlines.

Dependencies:

* Judge0 API is required for some of the key features of the system.
* A dockerised version of Judge0 might also be needed for online demos.

## Design and Implementation Constraints

The system must only be a web-based application because of time constraints; it needs to be implemented and deployed directly to a web server whereby the system will be hosted and run on to allow for easy and simple use for the student’s convenience.

## User Documentation

A comprehensive user manual will be provided for both students and lecturers. The manual will explain how to navigate the system, complete challenges, and access course materials. For lecturers, the documentation will include detailed guidance on administrator functions such as user management, course setup, and content uploads. The goal is to ensure all user groups can operate the system effectively and carry out their responsibilities with minimal support.

# REQUIREMENTS

## External Interface Requirements

### User Interfaces

The graphical user interface that ReCode will utilise is web-based and is made to be user-friendly for lecturers and students. To ensure continuous use, the interface will have a responsive design and be browser accessible on a variety of platforms, including desktop, tablet and mobile. Students can monitor their progress, get customised coding challenges and immediate feedback. An admin dashboard will be available for lecturers, which will enable them to submit resources such as lecture slides, view students' performance and generate automated feedback reports.

The user experience will be improved more especially for first year students, by using interactive components such as syntax-highlighted editors, drag and drop code blocks and code visual tools. Accessibility is given top priority in the design, which makes use of keyboard navigation and visual contrast.

### Hardware Interfaces

Since Recode is a web application hosted in the cloud, it does not need to be directly integrated with hardware. End users, such as students and lecturers, must utilise standard personal computers as well as their mobile cellphones that can run modern web browsers (e.g. Chrome, Firefox and Edge) with JavaScript enabled (if time allows we will look into making the system mobile-friendly, but for now it can be viewed in different screen sizes). On cloud platforms like Fly.io or Railway, backend services will operate in containers, utilising Redis for task queuing and Supabase for storage.

For internal development and deployment, the project will utilise:

* Workstations for developers that support Docker, Redis, and PostgreSQL
* Hardware for CI/CD pipelines on GitHub Actions or similar infrastructure

### Software Interfaces

FastAPI (Python) will be used to build the main application and microservices will communicate with each other via RESTful APIs. Important integrations with third-party software include:

* Supabase which is used for PostgreSQL database file storage and user authentication.
* Redis & Celery: for background tasks such as parsing slides or grading submissions.
* Judge0 API: for sandboxed code execution in a variety of programming languages.
* N8n-for automating workflows such as notifying students.
* JTW (JSON Web Tokens) will be used by the platform for API authentication and secure session management, Docker will be used to containerise all services and cloud deployment platforms will be used for control.

### Communications Interfaces

For online and API interactions, ReCodes’ system components use HTTP/HTTPS, which is a standard internet protocol. Key communication styles include:

* Internal REST API based service-to-service communication over secure HTTPS
* Browser sessions that allow the user to interact with external services.

The system will also be supported by webhooks (from third-party analytics) and real-time updates using WebSockets or long polling for specific components such as real-time progress tracking.

## Functional Requirements

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Function** | **Description** | **Inputs** | **Processing** | **Output** |
| **Weekly Challenge Delivery** | The system will release a coding challenge to students weekly | Lecture slides | Analyses the slides, generates a coding challenge | The system releases a weekly coding challenge corresponding to the assigned topic |
| **Lecture Content Integration** | Lectures can upload their slides for generating coding challenges | Lecture slides | Extracts text from the slides, analyses the texts |  |
| **Automated Topic Tagging (NLP)** | Determines key programming topic or concepts | Extracted text from lecturer slides | Extract topics and keywords from lecture text | Topics and keywords that will be used later |
| **Challenge Database & Retrieval** | The database containing coding challenges | Identified topics from NLP and by lecturer confirmation | Extracts coding challenges from the database, validates topics | The coding challenge is released to the student |
| **Code Submission & Judge0 Evaluation** | Students can submit their solutions for the weekly challenges | Code input from student | Evaluate code, validate code | Students get feedback on their code |
| **User Authentication & Roles** | An authentication system that is used for users to sign up and log in the system | Log-in details from users (students and lecturers) | Validates input, verifies email for login | Users gain access to the ReCode. |
| **Gamification Mechanics** | Tools and actions that make use of a reward system | Students’ progress and their current score for each challenge | Analyses input | Display the students’ score, rank and badges received |
| **Student Progress Dashboard** | The platform keeps a record of each student’s progress throughout the course | The user’s information from the semester | Internal operations | Displays students’ progress |
| **Lecturer Basic Tools** | Provides bare-bones capabilities for a lecturer to manage the platform, primarily to assign user roles and oversee data. | Input from the lecturer | Creates new course instances | Updated user permissions, a view of all students and their roles, and the creation of a new course instance in the database. |
| **Lecturer’s Analytics View** | The lecture can view an overview of the participation of students for each challenge | Stored student submission data, lecture requests | Aggregate and analyze student submission, generate report in the form of a graph | Students’ participation information, generated graph, computed participation metrics |
| **Lecture Customization** | Lectures can enter module information, and can accept the generated coding challenge topic or manually choose a topic | Selected module slides, lecture confirmation input | Capture lecturer’s decision, generate coding challenge | Display detected topic, store finalised topic to the system, and challenge available to students |

## Performance Requirements

The Recode platform must prioritise responsiveness and stability to support real-time coding challenge generation and automated grading during high-volume student usage.

Real-Time Responsiveness

* The system must generate and deliver coding challenges within 5000 milliseconds of a student request.
* Automated grading feedback must be returned within 3000 milliseconds to ensure smooth student interaction.

Scalability in Performance

* The MVP must support at least 200 concurrent student users during pilot deployment.
* The system should scale to 5,000 concurrent users with no degradation in grading speed or feedback quality. (Jayasekara, Harwood, & Karunasekera, 2020)

Reliability and Stability

* The system must operate continuously during academic sessions without unexpected crashes.
* It should recover gracefully from faults, log errors systematically, and ensure students do not lose progress.

Throughput

* The grading engine must handle at least 50 submissions per hour during peak loads without failure.
* Challenge generation and grading must operate in parallel to prevent bottlenecks.

Data Persistence Efficiency

* Student submissions, grading results, and challenge metadata must be written to storage asynchronously so that the user interface remains responsive.

Precision and Alignment

* Challenges must align precisely with extracted lecture content to ensure relevance and accuracy.

NLP-driven topic matching must maintain at least 90% accuracy during MVP testing.

## Security

The ReCode platform must ensure secure access, prevent misuse, and protect student and institutional data in line with POPIA and academic integrity standards.

Authentication and Authorisation

* Only registered lecturers and students may access the platform.
* A role-based access control (RBAC) system must enforce permissions, ensuring only lecturers can upload slides or modify challenge templates. (Stergiopoulos, Dedousis, & Gritzalis, 2020)

Communication Security

* All communication between the platform and backend services must use encrypted HTTPS protocols (TLS 1.2 or higher).

Access Control

* Only verified commands (challenge generation, grading, result retrieval) may be executed.
* Malicious code execution must be sandboxed using Judge0 to prevent security breaches.

Audit Logging

* All logins, challenge generations, submissions, and grading actions must be recorded with timestamps and user IDs for accountability.

Data Protection

* Sensitive data (student credentials, lecturer content, grading logs) must be encrypted at rest and securely stored in Supabase.
* Regular automated backups must be scheduled to protect against data loss.

## Usability

The ReCode platform is designed with usability as a top priority to support its target users: first-year students with varying levels of programming experience and lecturers with limited time to prepare study resources. The interface emphasises visual clarity, intuitive navigation, and reduced cognitive load to ensure efficient use without requiring extensive training.

For students, features such as inline feedback, real-time syntax validation, and guided tutorials lower entry barriers and provide a supportive learning experience. For lecturers, automated challenge generation from uploaded slides minimises manual effort while maintaining control over the accuracy and relevance of generated content.

Even though ReCode is designed with first years in mind, the system can also be used by students beyond first year. They can use the platform to strengthen their coding skills for the modules they are doing in their current years.

Usability testing will be conducted iteratively throughout development using an agile methodology. Metrics such as task completion rate, efficiency, error recovery, and subjective satisfaction will be evaluated during sprint cycles to continuously refine the user experience.

Key usability features include:

* User Interface & Experience:

Error messages are clear, actionable, and guide users to correct mistakes. Students benefit from real-time feedback and supportive learning pathways. Lecturers interact through simple, intuitive content management tools that streamline weekly challenge delivery.

* Command Line Interface (CLI) & Dashboard Design:

Developers and advanced users can access a script-friendly CLI to automate testing and workflows. Lecturers are provided with a web-based dashboard for uploading slides, reviewing analytics, and monitoring student progress.

* Documentation & Help Features:

Comprehensive documentation explains commands, features, and workflows in detail. Built-in help commands, tooltips, and context-sensitive guidance provide immediate support and reduce user frustration. (Bremers, et al., 2021)

By prioritizing usability, ReCode ensures an intuitive and supportive user experience that encourages adoption by both students and lecturers while maximizing educational outcomes.

## Scalability

Scalability is considered a core architectural requirement of ReCode, ensuring that the system is not limited to the immediate needs of the MVP but is capable of evolving into a robust, long-term platform. While the initial deployment will support a few hundred first-year students, the design anticipates expansion to thousands of concurrent users across multiple modules, faculties, and potentially institutions. The platform is therefore structured to grow in both capacity and functionality without significant re-engineering.

### 3.6.1 System Architecture Expansion

The architecture is intentionally modular, allowing components such as slide parsing, NLP-based topic extraction, and automated grading to be scaled independently. Containerisation (Docker) provides a mechanism to replicate services on demand, while load balancing distributes requests evenly across instances. By maintaining stateless APIs, additional application servers can be deployed without dependency on a single node, enabling smooth horizontal scaling during peak periods such as assignment deadlines.

### 3.6.2 Data Storage Expansion

The PostgreSQL database (hosted via Supabase) supports both vertical and horizontal scaling. Indexing, partitioning, and connection pooling strategies will ensure sustained performance as tables grow with submissions, progress logs, and challenge metadata. Cloud storage buckets expand automatically to accommodate uploaded lecture content, enabling long-term archival across multiple semesters. This creates the foundation for a continuously growing repository of challenges and course resources, similar to an evolving academic archive.

### 3.6.3 Code Execution and Grading Scalability

Judge0 will manage sandboxed code execution, with Redis and Celery providing a queuing mechanism to absorb spikes in demand. Should the hosted Judge0 API reach its limits, dockerised local instances can be deployed to provide redundancy and additional capacity. To optimise throughput, identical submissions will be cached and retrieved without requiring re-execution, ensuring that resources are used efficiently even at scale.

### 3.6.4 Data Analytics and Reporting Scalability

As adoption increases, lecturers will require more complex analytics. To prevent performance degradation, heavy reporting tasks will be offloaded to background jobs, with pre-computed aggregates and materialised views supporting real-time dashboards. Over time, comparative analytics may be introduced, enabling performance insights across different cohorts, modules, or semesters. This transforms the analytics function from a monitoring tool into a strategic resource for curriculum design.

### 3.6.5 User Growth and Concurrency

The MVP aims to support approximately 200 concurrent users, but the architecture targets scalability to 5,000 or more. Horizontal scaling of application servers, coupled with distributed session management and request throttling, ensures stability as usage expands. Authentication is handled through JWT-based sessions, which can be validated across multiple servers without compromising consistency or security.

### 3.6.6 Monitoring, Logging, and Fault Tolerance

The platform will be instrumented with real-time monitoring (Supabase logs, Sentry, and Prometheus or equivalent) to track latency, error rates, and resource utilisation. Autoscaling policies will trigger the deployment of additional service instances when load thresholds are exceeded. Redundant database replicas and automated backup schedules will safeguard against downtime, ensuring that no student submissions or lecturer uploads are lost during faults or failures.

### 3.6.7 Future Scalability Considerations

The scalability roadmap extends beyond raw capacity. Potential developments include multi-tenancy to support multiple faculties or institutions, microservices for AI-driven pipelines that scale independently of the core platform, and a continuously growing challenge repository accessible across academic years. In this vision, ReCode becomes not only a scalable teaching tool but a dynamic ecosystem: the more it is used, the richer its resources and the more valuable it becomes to both students and lecturers.

# References

Bremers, A. W., Yöntem, A. Ö., Li, K., Chu, D., Meijering, V., & Janssen, C. P. (2021). Perception of perspective in augmented reality head-up displays. *International Journal of Human-Computer Studies*.

Jayasekara, S., Harwood, A., & Karunasekera, S. (2020). A utilization model for optimization of checkpoint intervals in distributed stream processing systems. *Future Generation Computer Systems*.

Messer, M., Brown, N. C., Kölling, M., & Shi, M. (2024). Automated Grading and Feedback Tools for Programming Education: A Systematic Review. *ACM Transactions on Computing Education*.

Stergiopoulos, G., Dedousis, P., & Gritzalis, D. (2020). Automatic network restructuring and risk mitigation through business process asset dependency analysis. *Computers & Security*.