Change Report

Cohort 2 Team 1

Ahmet Abdulhamit
Zoey Ahmed
Tomisin Bankole
Alanah Bell
Sasha Heer
Oscar Meadowcroft
Alric Thilak

Change Process + Tools

To plan the changes required for “Grep the Exit”, our team first held a group discussion to determine a high-level approach to the new requirements. This included identifying significant deliverables such as new events, player achievements and creating a functioning leaderboard. After this overview concluded and a high-level plan was agreed upon, we used new tools to break these tasks down into individual changes, making it easier to assign and track work.

To track and manage these changes, we used GitHub’s issue tracker. Issue trackers provide a structured way to manage tasks within a codebase, allowing each team member to create issues. Each issue contained a title and a description specifying a feature request or a bug report with exact steps given to reproduce the bug. This provided the team a single central location for all change requests and bug reports, avoiding scattered messages and documents. Related issues could also be linked together using custom labels which helped the team identify dependencies and determine task priority.

GitHub also enabled effective coordination of task ownership. Each issue could be tagged according to the type of task it involved, such as new positive or negative events, artwork or bug fixes. This allowed the programmers to filter issues according to their assigned roles and focus on tasks in areas of the code they were most familiar with. After triaging, issues were then assigned to specific team members, making it clear who was responsible for each change. This approach prevented duplicated work which was an issue flagged by the team in our reflection of Assessment 1 when multiple coders unknowingly worked on the same large task.

For both communication and planning we used Discord. Unlike a single group chat on a platform such as Whatsapp, Discord allowed us to create dedicated channels for different topics and deliverables such as architecture discussion, CI changes and general deliverable document communication. This structure improved team coordination and made discussion more efficient than with previous methods like a Whatsapp chat.

Overall, by combining high-level planning, GitHub issue tracking and structured communication through Discord, the team was able to systematically plan, implement and review changes made. This approach ensured that all work was clearly documented, responsibilities were assigned, dependencies were managed and the team could collaborate efficiently throughout the process.

Requirements Changes

Following the handover of Assessment 1 deliverables from another team, the requirements were reviewed against the updated exam brief. Several gaps were identified where the original requirements no longer fully reflected the required gameplay scope, mechanics and constraints. As a result, new user and functional requirements were introduced and some existing requirements were refined to improve clarity, completeness and alignment with the updated brief. All changes were documented using structured tables to help preserve

traceability with the original deliverables and newly added requirements. The following changed requirements combined with the new requirements and the previous team's unchanged requirements form the full set of requirements for Assessment 2.

Changed Requirements:

ID	Previous Description	New Description	Justification for Change
UR_MAP	The user shall be able to explore a single large maze in the map	The user shall be able to explore the large maze main map alongside a smaller maze in the hidden room map	Updated to support the addition of a new hidden-room map
UR_PASSAGE	The user shall be able to skip parts of the maze by accessing hidden passages	The user shall be able to teleport to a hidden room via a portal which remains hidden whilst in the main maze map	Replaces an undefined mechanic with a clearly specified map traversal method of portals
FR_PROMPT	The game shall be able to display boxes at the bottom of the screen displaying text when an interaction begins	Interaction prompts are displayed near the player's inventory at bottom centre of the screen	Updated to improve UI clarity and consistency with common interface conventions whilst making space for the player's inventory
FR_BUS	The user shall have catch a bus to unlock the next point of the game	The user shall have to catch the bus in order to escape from the maze and complete the game	Better reflects narrative closure and end-game progression
UR_EXAM	The user shall have to overcome an exam to unlock the next point of the game	Exams open an interaction screen where the user must answer an exam question to continue, a correct answer gives a buff whilst an incorrect answer gives a debuff to the player	Expanded to clearly define exam interaction and consequences

New Requirements:

ID	Description	Justification for Addition	UR Links
UR_LOCKER	The user shall be able to interact with lockers around the map which will add items to the player's inventory	Items within the lockers are a hidden event and the lockers support item collection	n/a
UR_HIDDEN_MAP	The user shall be able to explore a hidden-room maze containing a locker, the Dean and an exit portal	The hidden map is one of the hidden events specified in the brief and it also adds progression depth	n/a
UR_INVENTORY	The user shall have a 3 slot item inventory mapped to keys 1-3	Necessary to manage collectible items and choose when to apply their effects	n/a
UR_PORTAL	The user shall be able to teleport between the two maps via a portal by using the T key	Enable travel between the two maps	n/a

UR_LOCKED_DOOR	The user shall only be able to pass through the locked door after inputting the correct 3-digit keycode on the door's keypad	This feature introduces structured progression to the game as well as puzzle-solving	n/a
UR_EVENTS	The game shall offer several different events to interact with, the event types shall be hidden, negative and positive	Added to align with the updated exam brief's requirements for varied event-driven gameplay	n/a
FR_APPLE	Lockers give an apple 50% of the time, apples give temporary increased player speed	One of the positive events which helps the player progress around the maze	UR_SPEED UR_LOCKER UR_INVENTORY
FR_COOKIE	Lockers give a cookie 30% of the time, cookies add 15 seconds to the player's clock	One of the positive events which supports the time-based game mechanics	UR_LOCKER UR_INVENTORY
FR_ROTTEN_APPLE	Lockers give a rotten apple 20% of the time, rotten apples temporarily slow the player	One of the negative events to help balance all the buffs the player is able to receive	UR_LOCKER UR_INVENTORY
FR_DIFF_LOCKER	The hidden room locker shall always contain a student card	Ensures progression from the hidden room map	UR_LOCKER UR_INVENTORY
FR_FULL_INVENTORY	Items cannot be collected if the inventory is already full	Prevents item hoarding and enforces strategic decisions	UR_LOCKER UR_INVENTORY
FR_STUDENT_CARD	A student card allows teleportation back to the main map from the hidden room map	Makes progression rely on finding an item and ensures the player has to actively think to continue playing	UR_LOCKER UR_HIDDEN_MAP UR_INVENTORY UR_PORTAL
FR_DEAN	The Dean traps the player in the hidden room map until the player finds their student card	Creates a small antagonist which helps the narrative and counts as a negative event	UR_HIDDEN_MAP
FR_KEYPAD_SCREEN	A keypad UI allows the player to input codes, clear inputs and gives feedback on correctness	Required to support the locked door interaction as the right code opens the door	UR_LOCKED_DOOR
FR_FIND_KEYCODE	Each NPC provides one digit of the keypad door code	Encourages the player to explore the map and interact with NPCs	UR_LOCKED_DOOR
FR_EXAM_NOT_HARD	To help ensure UR_FAMILY the exam questions shall be general knowledge and not too hard	Ensures compliance with accessibility requirements by making sure the exams are accessible to all ages	UR_FAMILY UR_EXAM
FR_EXAM_CORRECT	Correct exam answers grant the player temporary guard immunity	Rewards the player's skill and helps engage the player with the game, positive event	UR_EXAM
FR_EXAM_INCORREC	Incorrect exam answers	Adds a consequence based on	UR_EXAM

T	temporarily speed up the guards	the player's skill level and counts as a negative event	
FR_POSITIVE_EVENTS	The player shall be able to encounter 3 positive events, each event shall benefit the player in some way	Added to align with the updated exam brief's requirements for varied event-driven gameplay	UR_EVENTS
FR_NEGATIVE_EVENTS	The player shall be able to encounter 5 negative events, each event shall hinder the player from progressing	Added to align with the updated exam brief's requirements for varied event-driven gameplay	UR_EVENTS
FR_HIDDEN_EVENTS	The player shall be able to encounter 3 hidden events, each hidden until triggered	Added to align with the updated exam brief's requirements for varied event-driven gameplay	UR_EVENTS

New requirements were added to reflect mechanics and constraints specified in the updated exam brief that were not present in the original team's Assessment 1 deliverables. These additions, along with refinements to existing requirements, were necessary to fully specify newly required gameplay systems such as inventory management, hidden maps, full exam interaction, event-driven mechanics, etc. The revised requirements improve clarity, reduce ambiguity and ensure full alignment with the updated brief while maintaining traceability with the original documentation.

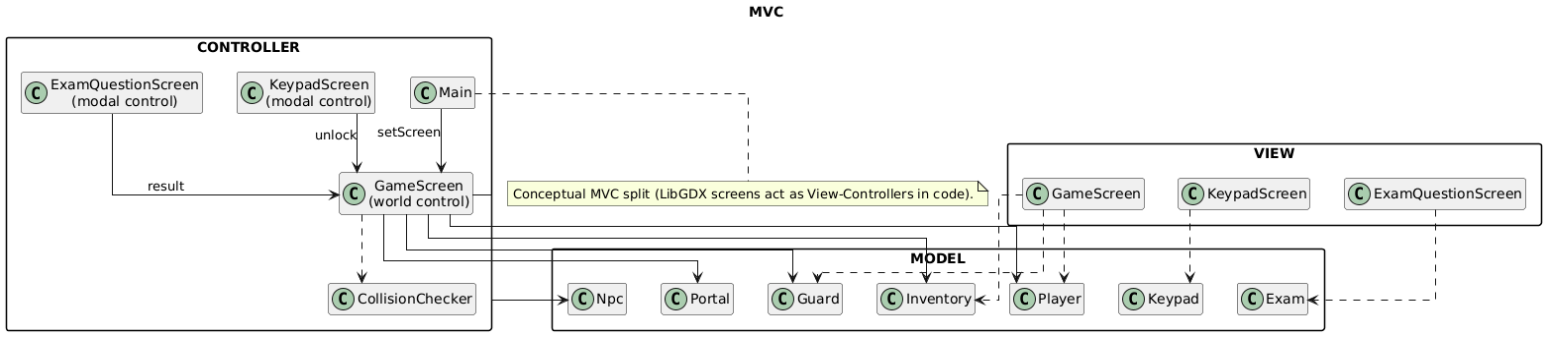
Architecture Changes

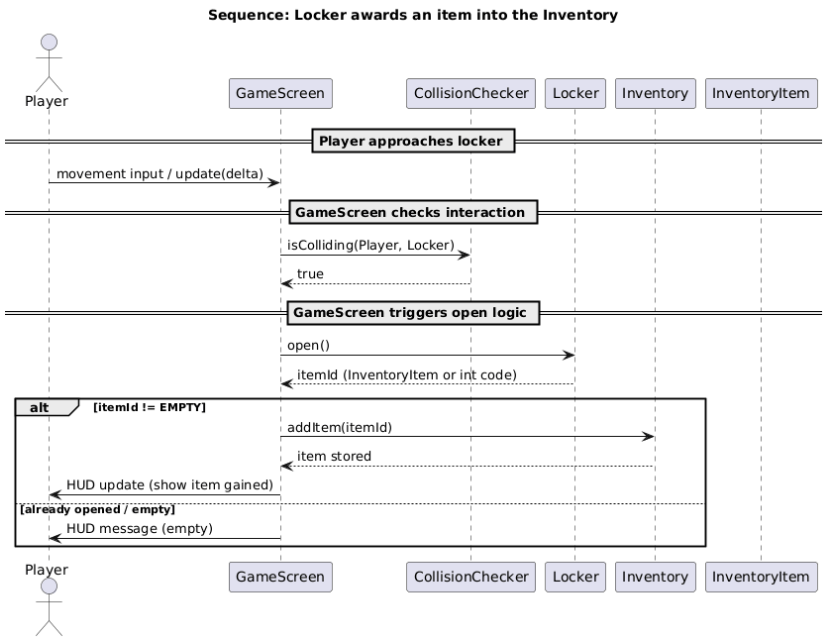
Our architecture changes were made to fulfil our new set of requirements, while building off the architecture of the original team's work. The original architecture was based on the Model-View-Controller pattern, which our new architecture changes adhered to. We refined the architecture through a set of linked structural decisions aimed at improving the modularity, clarity of responsibility and overall maintainability of the codebase, whilst fostering a more immersive and interactive gameplay experience for the player. In particular, our implementation steered away from handling all interactions inline inside the main gameplay loop, by introducing modal, screen-based workflows for UI-heavy features, alongside a clearer runtime mechanism for world transitions through portals and dynamic map switching. Additionally, we expanded the domain model and improved its structure through additional specialised interactable object types, a more interactive exam model, mediated by dedicated UI screens. Furthermore, the introduction of a persistent inventory subsystem that supports state-based progression and gating. While our project remains MVC inspired in its use of models (entities, objects, state) and screens (KeyPadScreen, GameOverScreen, etc), our architecture strengthens separation of concerns by distributing input ownership across specialised interaction screens, resulting in a more layered, screen-driven game architecture. All architecture diagrams and CRC cards can be found at Previous Deliverables | Grep The Exit under 'Graphs' and 'CRC Cards'

Title	Original	Change Implemented	Justification for Change	Requirements Traceback
Improved Game Flow and State Management	There were implicitly fewer transition types and less structured state flow handled, primarily through GameScreen.	New types of game flow transitions: Pausing and resuming the game via 'PauseScreen', A reward/punishment flow for exams.	Changes improved state management by making flow transitions explicit and supporting richer game progression and consequences.	UR_PORTAL UR_HIDDEN_MAP UR_MAP UR_EXAM FR_EXAM_CORRECT FR_EXAM_INCORRECT
Modal Interaction Architecture	Most interactions were handled inside GameScreen while the mainloop was running. So, the gameplay loop handled responsibilities like, world simulation, input processing and UI feedback.	We moved UI-heavy interactions to modal screens like: - ExamQuestionScreen - KeypadScreen - TutorialScreen - WinScreen	Changes improved separation of concerns because those screens focus on input and UI, so GameScreen can focus on world simulation. Also increases modularity due to higher cohesion per screen, thus, improving maintainability of code.	UR_EXAM FR_KEYPAD_SCREEN UR_LOCKED_DOOR FR_PROMPT
Domain Modelling of NPCs	There was no dedicated NPC implementation. The moving character model was represented through entity subclasses, while stationary interactables were represented by the Object superclass.	We implemented NPCs as a subclass of Object, purposefully reflecting their nature as stationary interactables.	Improves semantic accuracy in our domain model by separating moving agents from static interactables. So, NPC avoids inheriting redundant movement logic, reducing complexity and improving modularity by keeping object responsibilities cohesive.	FR_FIND_KEYCODE UR_EVENTS UR_MAP UR_HIDDEN_MAP
Introduction of an Inventory Subsystem	Player progression was limited to transient effects like score updates and basic powerups which did not persist beyond their immediate use. Restricting, the ability to model more complex gameplay	Introduced a dedicated inventory subsystem, which enabled items to be collected, stored and referenced across multiple player interactions. Decoupling item management from temporary gameplay effects and encapsulating it within a persistent domain model.	The system now supports a more diverse range of gameplay mechanics and more flexible progression logic. As item-based progression and gated interactions support varied event-driven gameplay through collectible item effects.	UR_INVENTORY UR_LOCKER FR_FULL_INVENTORY FR_APPLE FR_COOKIE FR_DIFF_LOCKER FR_STUDENT_CARD

	mechanics.			
Expansion of Interactable Object Types	Original architecture had a smaller set of interactables and relied heavily on GameScreen to orchestrate primary game logic.	We implemented more interactable types like: - Portal - Npc - Keypad Representing a decomposition of a previously monolithic interaction mechanism into a smaller set of specialised components, each of which encapsulates its own interaction state (e.g. opened/activated).	Improved cohesion as each object is responsible for one mechanic, reducing the amount of logic hard-coded in the main loop. It also supported the expansion of hidden/positive/negative events and structured progression through object-based interactions, as opposed to hard-coded logic.	UR_LOCKER UR_PORTAL UR_LOCKED_DOOR UR_HIDDEN_MAP UR_EVENTS FR_POSITIVE_EVENTS FR_NEGATIVE_EVENTS FR_HIDDEN_EVENTS FR_DEAN

MVC Diagram and Locker Interaction Sequence Diagram:





Method Selection and Planning Changes

Summary of Changes:

We updated the previous team's method selection and planning processes to reflect changes in our development tools, team communication platforms and planning approaches. These changes were made to improve team coordination, development efficiency and clarity in project tracking.

Detailed Changes and Justifications:

1. Communication Tool: Whatsapp -> Discord

- Original: WhatsApp was used by the previous group for its simplicity and pre-existing familiarity
- Change: Our team migrated to Discord for structured and effective communication
- Justification: Discord offers structured communication through the use of channels within a single server, allowing discussions to be organised by topic (e.g. a channel for each deliverable). This reduced chat clutter and improved the visibility of important updates. Discord also offers built-in voice channels and screen sharing features which facilitated real-time meetings without the need for external apps alongside efficient collaboration and problem solving. As the majority of the group were already familiar with Discord, the transition was smooth and significantly improved our communication during Assessment 2. In addition to using Discord as the primary communication platform, WhatsApp was kept as a secondary backup channel in cases where a member had temporary access issues with Discord.

2. IDE: IntelliJ -> VS Code

- Original: IntelliJ was chosen by the previous group for its Java-specific tooling and integration with Gradle/Git
- Change: We used VS Code as the primary IDE for development
- Justification: All programmers were already proficient with VS Code which eliminated time spent learning a new IDE and allowed for a clear focus on development. Through the use of extensions, VS Code fully supports Git version control and Gradle automation, offering equivalent functionality to IntelliJ. VS Code is also of a lightweight nature which ensured smooth operation on older hardware, enabling all team members to contribute effectively without any technical limitations. The use of VS Code also aligned well with our continuous integration and delivery workflow, as its Git integration allowed developers to frequently commit, review and merge changes as part of an iterative development process

3. Asset Creation: CC0 Pack -> Team Created Pixel Art

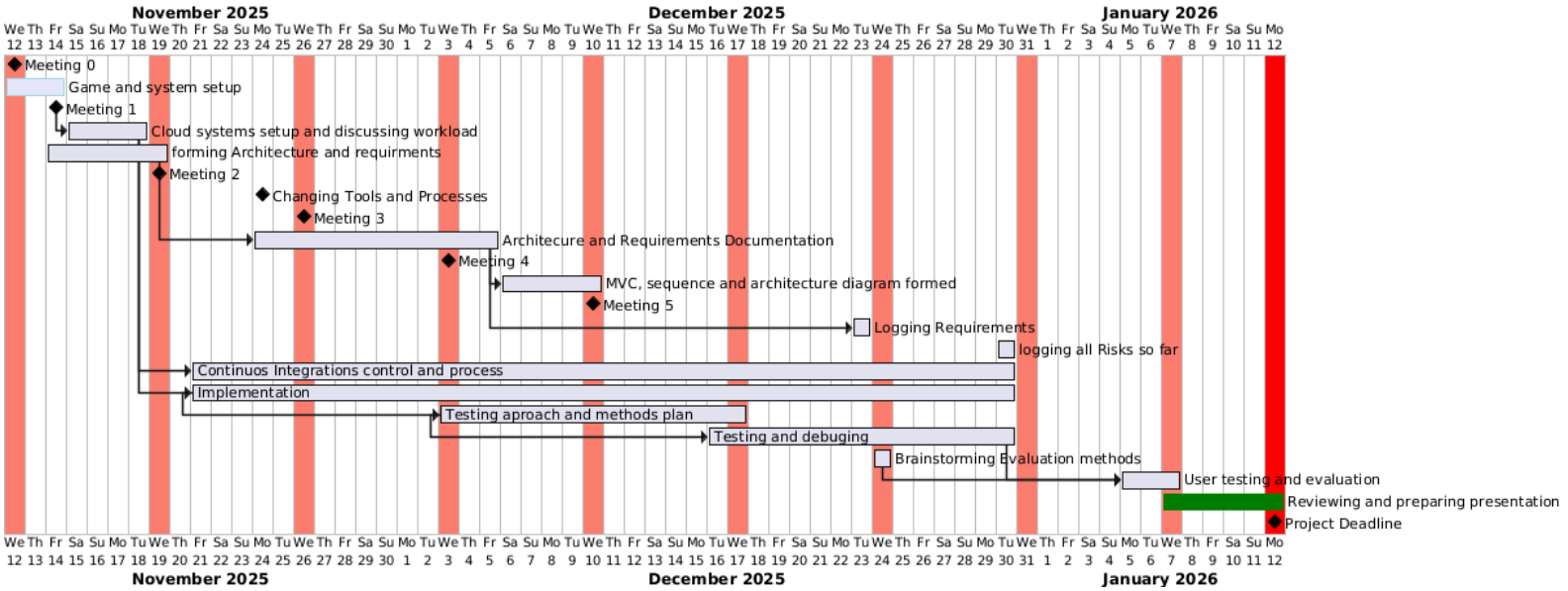
- Original: For the previous group, assets were sourced from a free CC0 pack after AI-generated assets proved inconsistent
- Change: A team member created custom pixel art assets using Pixilart (a free online tool)
- Justification: One member creating the new assets ensured consistency across all game elements and allowed the visuals to be tailored to the maze's theme more effectively. This resulted in a more cohesive and polished appearance compared to using externally sourced assets. Centralising asset creation with a single team member also reduced dependency conflicts and simplified integration during implementation, as assets followed a consistent style and resolution from the outset.

4. Team Structure: Flat hierarchy and subteams -> Agile /Scrum

- Original: The previous team used a flat structure with subteams and weekly meetings but did not give their methodology an explicit name
- Change: We adopted Agile/Scrum practices with defined roles (Scrum Master, Product Owner)
- Justification: This change allowed tasks to be assigned according to individual team members' strengths. Members confident in coding handled implementation and testing, while those with strong writing and communication skills led documentation such as risk management, planning and requirements changes. Responsibilities were regularly reviewed and adapted as the project evolved, allowing members to support each other across any overlapping areas and inability to complete segments of work. This structure reinforced accountability through regular check-ins and iterative reviews of individual roles and responsibilities - aligning with Agile principles of collaboration and adaptive planning. This approach helped maximise productivity by utilising individual expertise while maintaining team cohesion and adaptability - particularly valuable for the short time scale to complete the assessment. Rather than maintaining fixed subteams, responsibilities were organised around functional fields and task dependencies (e.g. backlog preparation, continuous integration and delivery, review and monitoring). These fields interacted iteratively throughout the project, ensuring that planning, implementation, testing and evaluation remained closely aligned. Monitoring was treated as a continuous activity across the entire project lifecycle, focusing on tracking progress, identifying risks, verifying that client requirements were satisfied and ensuring that tools and processes were functioning effectively.

Systematic Plan:

A Gantt chart was used to support systematic planning and scheduling throughout the project. The chart visualised tasks over time, showing their duration, overlap and dependencies across the project timeline. This allowed the team to plan work both sequentially and in parallel, making it easier to coordinate activities, manage workload distribution and identify potential scheduling risks early. Progress was tracked against the planned schedule, providing clear visibility of deadlines and task completion whilst enabling the team to adjust priorities where necessary to remain on track. Timeline snapshots available at: Timeline | Grep The Exit



Risk Assessment and Mitigation Changes

We changed the risks that occurred by using the approach of severity and likelihood rankings during the risk assessment. A new category we added was implementation since it differs from technological issues. Categories included people, estimation, technology, implementation and requirements. Additionally, we removed the ownership since risks are not managed by a single person alone but mostly the entire group instead. For our risk analysis, we changed the scoring system to resemble the range of difficulty:

Risk Analysis Scale:

- Very Low (1), Low (2), Moderate (3), High (4), Very High (5)

Risk Monitoring:

All members are informed about the Risks and monitoring is shared between members and is analysed by severity and likelihood. We mainly try to solve it or decrease inconvenience, then at every meeting, the member tasked with tracking risks monitors and assesses the situation to check if the problem has been solved.

ID	Category	Description	Severity	Likelihood	Mittigation
R1	People	People getting sick, travelling, or not being available due to personal matters.	4	4	Every member checks up on each other, or the person lets the group know to allow discussion of workload and deadlines.
R2	People	People having problems switching communication platforms and tools.	3	1	Members who report problems with switching tools or platforms would either receive support or would be checked upon regularly.
R3	Estimation & People	Over or under-estimating the time a task will take, mainly due to unequal workload distribution, unavailability, etc.	4	3	Whenever a task is in progress, the member(s) in charge of the task will be asked if support is needed or to reschedule deadlines.
R4	Implementation	Bugs or the game failing due to a lack of testing.	3	1	Our implementation team reviews every merge and tests it to minimise inconveniences.
R5	Technology	Programs, systems, or devices not functioning as expected, causing people to delay work.	4	1	Members affected to let the group know to troubleshoot and reschedule if needed.

Most of the risks in the people and estimation category are either subfields of each other or their severity, likelihood and mitigation changed. R1, 2, and 3 are equivalent to 2 or 3 risks mentioned by the previous group. The reason we combined some of them is that they're either almost the same risk or one is the continuation of the other. For our technological risk that we've changed into implementation (R4) we've managed to decrease the severity.

Our final technological risk is equivalent to the majority of the risks mentioned by the prior group. Our approach consisted of members being informed and those who either had experience or knew about the problem and knew how to fix it, prioritised troubleshooting.