

Component 2
Written assessment
Problem Analysis and Solution Development

Module: Analysing & Solving Problems

Submitted By: Amir Bux

ID: 2526189

Words Count (2000)

List of Contents

Introduction	3
1. Problem Deconstruction.....	3
1.1 Variables and Constraints	4
1.2 Requirement Prioritisation	4
1.3 Assumptions	5
2. Proposed Solution and Explanation.....	5
2.1 Step One – collection availability	6
2.2 Step Two – compute the intersection of availability	6
2.3 Step Three – does a common slot exist before the deadline?	6
2.4 Step Four – decision-can quorum be met instead?	6
2.5 Step Five – escalation fallback.....	7
2.6 Design alternatives considered	7
2.7 Worked illustration	7
3. Reflection and Justification.....	8
3.1 Worked illustration	8
3.2 Systems Thinking	9
3.3 Justification of MoSCoW.....	9
3.4 Limitations and Assumptions.....	9
4. Conclusion.....	10

Introduction

This report addresses a scheduling problem affecting a college computing group, whose members' capacity to meet is constrained by part-time employment, conflicting timetables, and personal commitments. Left unresolved, this fragmentation of availability has produced delayed task completion and an uneven distribution of workload across group members. The purpose of this report is to analyse the underlying causes of this coordination failure and to design a generic, reusable procedure for identifying viable meeting times. The analysis proceeds in three stages. Section 1 deconstructs the problem into its constituent variables and constraints, applying the MoSCoW technique to prioritise requirements. Section 2 presents a flowchart-based solution, together with a detailed explanation of its logic, decision points, and underlying assumptions. Section 3 reflects critically on the problem-solving techniques employed, justifies the chosen approach with reference to established literature, and identifies the limitations of the proposed solution.

1. Problem Deconstruction

The scenario presents an ill-structured, real-world problem, a college computing group cannot reliably convene because members' available time is fragmented by part-time employment, differing timetables, and personal commitments, with the consequence that project tasks stall and workload is distributed unevenly. Before a solution can be designed, the problem must be reduced to its constituent parts; a process of decomposition, which involves breaking a complex problem into smaller, more manageable sub-problems so that each can be reasoned about independently before being recombined [4].



1.1 Variables and Constraints

The scenario can be described using the following core variables:

The number of group members (n); each member's availability, expressed as a set of free time blocks per week; hard constraints that remove time from availability (paid employment shifts, timetabled classes); soft constraints that reduce willingness without strictly prohibiting attendance (personal commitments); the recurring weekly project deadline, which fixes an upper bound on when a meeting must occur; and a minimum quorum, defined here as the smallest proportion of members whose presence renders a meeting valid [1].

Framed this way, the scenario is a constraint satisfaction problem, a search for a time value that simultaneously satisfies every member's availability constraint while remaining bounded by the deadline. This framing is consistent with the operational research literature on timetabling, in which the allocation of events (here, meetings) to time slots subject to competing constraints is a long-established and formally studied problem class, with well-documented computational difficulty as the number of constraints and participants increases [2].

1.2 Requirement Prioritisation

To ensure the solution addresses what matters most within the constraints of the scenario, requirements are prioritised using the MoSCoW technique, a structured method for categorising requirements by criticality that was developed to give project stakeholders a shared, unambiguous basis for prioritisation under time pressure:

- **Must have:** a mechanism that identifies a time slot in which all members are simultaneously free before each weekly deadline, respecting hard constraints.
- **Should have:** minimisation of the number of communication iterations needed to reach agreement on a time, and advance notice of at least 24–48 hours once a slot is confirmed [2].
- **Could have:** a documented fallback allowing partial attendance where full attendance cannot be achieved, and integration with members' existing digital calendars.
- **Won't have (this iteration):** a guarantee of full attendance in every week regardless of constraint severity, and resolution of conflicts that arise after a meeting has already been confirmed [4].

1.3 Assumptions

The deconstruction rests on several assumptions that are made explicit here rather than left implicit that members can report their availability in a consistent, structured format; that “available” denotes genuine freedom from conflicting obligations rather than mere willingness to attend; and that all members have access to a shared digital channel through which availability and confirmations can be exchanged. These assumptions are revisited critically in the Reflection and Justification section [7].

2. Proposed Solution and Explanation

The proposed solution is a generic scheduling algorithm, represented as a flowchart which any group of n members can apply regardless of the specific composition of their individual timetables [1]. Its generic character is deliberate, the assignment requires a solution that generalises beyond the scenario described, and the algorithm below takes as inputs only the variables identified in the Problem Deconstruction section, member availability, group size, quorum threshold, and deadline making it applicable to any comparable group-coordination problem [7].

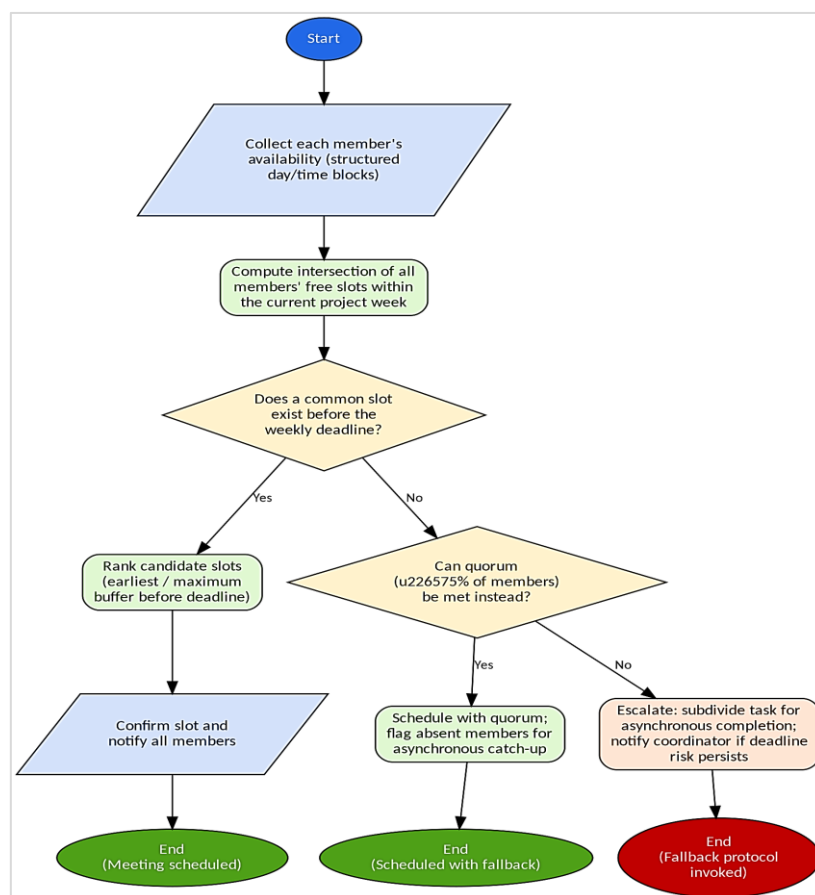


Figure 1-Generic Group-meeting Scheduling Algorithm

2.1 Step One – collection availability

Each member submits their free time blocks for the current project week in a structured format (day and time range). This step converts an informal, distributed problem everyone privately knowing their own schedule into a single, explicit dataset that can be processed algorithmically. This is itself a form of abstraction: representing the messy reality of individual timetables as a simplified, comparable data structure that strips away irrelevant detail (the reason a member is unavailable) while retaining what is operationally necessary (whether they are available) [4] [10].

2.2 Step Two – compute the intersection of availability

The algorithm computes the intersection of all members' free-time sets, producing the subset of time blocks in which every member is simultaneously free. This is the computational core of the solution and directly addresses the first stated objective finding meeting times that accommodate all members' schedules. Framing this step as a set intersection rather than, for instance, a majority vote or informal negotiation is deliberate: it treats the scheduling task as the constraint-based search problem it structurally is, consistent with established approaches to automated timetabling in which feasible allocations are those satisfying the full set of hard constraints simultaneously [5].

2.3 Step Three – does a common slot exist before the deadline?

If the intersection produces one or more-time blocks that fall before the relevant weekly deadline, the algorithm proceeds to rank the candidate slots, for example, by preferring the slot offering the greatest buffer before the deadline, which reduces the risk that task delays cascade into missed submissions. The selected slot is then confirmed and all members are notified. This branch directly satisfies the second and third stated objectives. It minimises timetable conflict by construction (only genuinely conflict-free slots are considered) and structurally guarantees that any confirmed meeting precedes the deadline, since slots beyond the deadline are excluded from candidacy at this decision point [1] [10].

2.4 Step Four – decision-can quorum be met instead?

If no fully common slot exists, the algorithm does not fail outright but checks whether a quorum threshold (for example, at least 75 per cent of members present) can be satisfied.

If so, the meeting proceeds with the available majority, and absent members are flagged for an asynchronous catch-up mechanism, for instance, reviewing minutes and submitting outstanding input separately[9]. This branch is a deliberate design trade-off: it accepts a lower standard of attendance in exchange for preventing complete scheduling failure, on the basis that a meeting with most members present and a defined follow-up process is preferable, in terms of maintaining project momentum against a fixed deadline, than no meeting at all [5].

2.5 Step Five – escalation fallback

If neither full availability nor quorum can be achieved, the algorithm does not attempt further automated negotiation, since doing so would risk consuming time the group cannot spare against the deadline. Instead, it triggers an escalation, the group's task is subdivided for asynchronous completion and where the deadline is genuinely at risk, the module tutor or coordinator is notified. This final branch reflects a recognition that not every instance of the scheduling problem is solvable by scheduling logic alone, a point developed further in the Reflection [6].

2.6 Design alternatives considered

Two simpler alternatives were considered and rejected. A majority-vote model, in which members vote on a shortlist of proposed times, was rejected because it does not guarantee that the winning slot is genuinely conflict-free for every member, which fails the assignment's first stated objective outright rather than degrading gracefully. An unstructured, informal negotiation model (members messaging back and forth until a time emerges) was rejected because it does not bound the number of communication iterations, which directly conflicts with the Should-have requirement identified in the Problem Deconstruction and risks consuming time that a fixed weekly deadline does not allow. The intersection-and-quorum structure adopted here was preferred because each of its branches produces a bounded, predictable outcome within a small, fixed number of steps [9].

2.7 Worked illustration

A brief illustrative pass through the logic clarifies its behaviour. Consider a group of four members with the following weekly free-time blocks: Member A is free Tuesday 14:00–17:00 and Thursday 10:00–12:00; Member B is free Tuesday 15:00–18:00 and Friday 09:00–11:00; Member C is free Tuesday 14:00–16:00 only, owing to a part-time shift

beginning at 16:00; Member D is free Tuesday 15:00–17:00 and Thursday 09:00–11:00 [8]. Intersecting these sets yields a single common window: Tuesday 15:00–16:00. Provided this falls before the week's submission deadline, Step 3 confirms it directly. If Member C's shift had instead begun at 14:30, removing the Tuesday overlap entirely, no full intersection would exist; Step 4 would then check whether the remaining three members (75 per cent of the group) constitute quorum and, if so, proceed with Member C flagged for asynchronous catch-up rather than allowing the week to pass without any meeting at all [3].

Together, these five steps constitute a decision procedure that is deterministic, generalisable to any group size or constraint configuration, and directly traceable to each of the three objectives set out in the assignment brief: it actively searches for slots accommodating all members (Step 2), it minimises conflict by treating only genuinely free slots as valid candidates rather than approximating agreement through negotiation (Steps 2–3), and it structurally enforces the deadline constraint by excluding non-compliant slots from consideration at the point of selection (Step 3), while providing a graceful, non-blocking response rather than simple failure when full accommodation proves impossible (Steps 4–5).

3. Reflection and Justification

The approach taken in this assignment relies principally on two problem-solving techniques, decomposition and abstraction, and systems thinking, supported by MoSCoW as a prioritisation instrument [6].

3.1 Worked illustration

The problem was addressed by breaking the overall coordination failure into discrete, independently analysable components availability, constraints, deadline, and quorum before recombining them into a single procedure. This mirrors the account of computational thinking advanced by Wing, in which decomposition and abstraction are presented not as programming-specific techniques but as generalisable habits of problem-solving, reducing a complex situation to its essential structure while deliberately discarding irrelevant detail. This was appropriate here because the original scenario, described narratively, conflated several distinct sub-problems (data collection, constraint

checking, deadline enforcement, and exception handling) that needed to be separated before a coherent algorithm could be designed [7].

3.2 Systems Thinking

A mechanical, optimisation-only reading of the scenario would treat scheduling as a solvable search problem and stop there. However, the underlying difficulty described in the brief, uneven workload, delayed submissions, and strained group dynamics is a human and organisational issue as much as a computational one [3]. The distinction between “hard” systems, which can be engineered towards a defined objective, and “soft” systems, in which human perception, motivation, and disagreement genuinely shape outcomes, is directly applicable. The scheduling algorithm proposed here is a hard-systems solution to what is, in part, a soft-systems problem; this is precisely why the design deliberately avoids treating scheduling failure as terminal (Steps 4–5), instead building in escalation and human referral rather than assuming the algorithm alone can resolve every case.

3.3 Justification of MoSCoW

MoSCoW is chosen over unranked requirement-listing because the scenario involves genuine, unavoidable trade-offs full attendance cannot always be guaranteed against a fixed deadline and MoSCoW forces those trade-offs to be made explicit and defensible rather than left implicit, which was its original purpose in rapid, time-constrained project settings. This proved directly useful in distinguishing the Must-have requirement (a valid meeting before the deadline) from the Could-have refinement (partial-attendance accommodation), which shaped the branching structure of the flowchart itself [7].

3.4 Limitations and Assumptions

Several limitations should be acknowledged. the solution assumes availability data is reported accurately and does not model deliberate misreporting or last-minute changes, which would require a more dynamic, re-computation-capable design [3]. The quorum fallback introduces an unresolved equity question, whose absence is treated as acceptable, and on what basis which the current design does not adjudicate. Most significantly, timetabling, and scheduling problems of this general class are known in the operational research literature to become computationally difficult as the number of participants and constraints increases, meaning the simplicity of this generic procedure,

appropriate for a small project group, may not scale directly to larger or more heavily constrained settings [8]. These limitations do not undermine the solution's suitability for the scenario as specified, but they mark the boundary of its applicability and indicate where further development like, dynamic re-scheduling or weighted fairness in quorum selection would be required for broader use [2].

4. Conclusion

This report has deconstructed a real-world scheduling conflict into its constituent variables, prioritised requirements using MoSCoW, and translated the analysis into a generic, constraint-based algorithm capable of accommodating any group's availability, deadline, and quorum parameters. The proposed solution meets all three objectives set out in the brief while incorporating a non-blocking fallback for cases where full attendance cannot be achieved. Its principal limitation is scalability as demonstrated in the Reflection, the underlying scheduling problem class becomes computationally harder as group size and constraint complexity grow, marking the boundary beyond which this generic procedure would require further refinement.

References (IEEE Style)

- [1] R. Ross, “4 classroom organisation tips for the upcoming school year,” *Maths – No Problem! Maths Mastery for Primary School Education*, Jan. 05, 2021.
- [2] “Allocation Problems | Overview & Research Examples,” *Perlego.com*, 2016.
Available: <https://www.perlego.com/index/mathematics/allocation-problems>.
- [3] D. de Werra, “An introduction to timetabling,” *European Journal of Operational Research*, vol. 19, no. 2, pp. 151–162, 1985.
- [4] D. Clegg and R. Barker, *Case Method Fast-Track: A RAD Approach*. Wokingham, U.K.: Addison-Wesley, 1994.
- [5] J. M. Wing, “Computational Thinking,” *Communications of the ACM*, vol. 49, no. 3, pp. 33–35, 2006.
- [6] X. Zhang, P. Parikh, and L. Xiao, “Course Scheduling Problem with Cohort Group and Room Considerations,” *International Journal of Industrial and Systems Engineering*, vol. 1, no. 1, p. 1, 2022, doi: <https://doi.org/10.1504/ijise.2022.10047812>
- [7] L. Team, “Examples of Abstraction in Everyday Life | Learning.com,” *Learning*, Oct. 18, 2022. Available: <https://www.learning.com/blog/examples-of-abstraction-in-everyday-life/>.
- [8] A. Z. Azhar, A. Segal, and K. Gal, “Optimizing Representations and Policies for Question Sequencing Using Reinforcement Learning,” *International Educational Data Mining Society*, 2022, Available: <https://eric.ed.gov/?id=ED624101>.
- [9] E. K. Burke and S. Petrovic, “Recent research directions in automated timetabling,” *European Journal of Operational Research*, vol. 140, no. 2, pp. 266–280, 2002.
- [10] P. B. Checkland, *Systems Thinking, Systems Practice*. Chichester, U.K.: Wiley, 1981.