1.2 Types of Software and Development Methodologies

Total Marks

OCR H446 · Calculators not allowed · about 125 minutes · Write-in edition

Name: _____ Date: ____ / ____ / _____

Answer ALL questions. Show all your working.

1 A company must decide whether to buy off-the-shelf software or commission bespoke (custom-written) software.

(a) State what is meant by the term 'bespoke software'. **(2)**

(b) State what is meant by the term 'off-the-shelf software'. **(2)**

(c) A regional hospital trust wants a new system to manage patient appointments across 12 sites, with several unusual workflow requirements that are not met by any existing product. Discuss whether the trust should choose off-the-shelf or bespoke software. **(6)**

(Total for Question 1 is 10 marks)

2 Software can also be classified by whether or not its source code is publicly available.

(a) State what is meant by the term 'open source software'. **(2)**

(b) State what is meant by the term 'closed source (proprietary) software'. **(2)**

(c) A small business is choosing a content management system for its website and is considering an open source option. Give two advantages and two disadvantages of using open source software in this context. **(4)**

(Total for Question 2 is 8 marks)

3 Utility software carries out specific maintenance tasks on a computer system.

(a) State what is meant by the term 'utility software'. **(2)**

(b) For each scenario below, identify the type of utility software that would be used. **(4)**

i) Compressing a large video file before emailing it.

ii) Protecting a computer from malicious software downloaded from the internet.

iii) Reorganising fragmented files on a hard disk drive to improve access speed.

iv) Creating a copy of important files in case the originals are lost or damaged.

(c) Explain why defragmentation utility software improves performance on a hard disk drive (HDD) but has little or no benefit on a solid-state drive (SSD). **(3)**

(Total for Question 3 is 9 marks)

4 Rapid Application Development (RAD) focuses on fast delivery through prototyping.

(a) State two characteristics of RAD. **(2)**

(b) Explain why RAD may not be suitable for a project where users are not available to give regular feedback. **(3)**

(c) Compare the use of prototyping in RAD with the waterfall model. **(3)**

(Total for Question 4 is 8 marks)

5 Agile methodologies are an alternative to traditional models such as waterfall.

(a) State three principles that underpin agile approaches to software development, as set out in the agile manifesto. **(3)**

(b) Explain two ways in which agile development handles changing requirements differently from the waterfall model. **(4)**

(c) A government department is planning a large IT project with a fixed contract value and a fixed delivery date set in law. Discuss whether an agile methodology would be appropriate for this project. **(6)**

(Total for Question 5 is 13 marks)

6 The spiral model combines iterative development with a strong focus on risk management.

(a) Describe the four phases carried out in each loop (cycle) of the spiral model. **(4)**

(b) Explain why the spiral model is well suited to large, high-risk projects. **(3)**

(c) Explain one disadvantage of using the spiral model. **(2)**

(Total for Question 6 is 9 marks)

7 Extreme Programming (XP) is an agile methodology that emphasises technical practices.

(a) State three practices associated with Extreme Programming. **(3)**

(b) Explain one benefit of pair programming. **(3)**

(c) Explain one benefit of using continuous integration in a development team. **(3)**

(Total for Question 7 is 9 marks)

8 A start-up company is developing a mobile app for online banking. The app must be launched within 4 months to meet a marketing deadline, but the founders expect features to change as user feedback comes in after each release. Security and regulatory compliance are critical.

(a) Recommend and justify a suitable software development methodology for this project. **(8)**

(Total for Question 8 is 8 marks)

9 A small independent bookshop wants a new till (point-of-sale) and stock management system. The owner has a very limited budget and standard requirements similar to many other small shops, but wants some control over any future changes to the software.

(a) Discuss which combination of software type (bespoke vs off-the-shelf) and licensing model (open source vs closed source) would best meet the bookshop's needs, and justify your recommendation. **(8)**

(Total for Question 9 is 8 marks)

10 Evaluate the extent to which agile methodologies have made the waterfall model obsolete for modern software development.

(Total for Question 10 is 9 marks)

11 A university is replacing its student records system, which is used by over 20,000 students and staff and must comply with strict data protection law. Errors in the new system could result in incorrect exam results being issued. The university has stated that it wants to combine techniques from more than one development methodology.

(a) Suggest one reason why a purely agile approach might be considered too risky for this project. **(3)**

(b) Suggest one reason why a purely waterfall approach might also be unsuitable. **(3)**

(c) Describe a hybrid approach the university could use, combining features of two methodologies, and explain how it addresses the risks identified in (a) and (b). **(6)**

(Total for Question 11 is 12 marks)

Mark scheme · 1.2 Types of Software and Development Methodologies

Question 1

(a) Software written specifically for a single customer/organisation (oe) **B1**

(a) ...to meet their exact requirements/specification (oe) **B1**

(a) Answer: Bespoke software is software written specifically for a single customer or organisation to meet its exact requirements.

(b) Software that already exists / has been pre-written (oe) **B1**

(b) ...and is available for anyone to buy/use, usually without modification (oe) **B1**

(b) Answer: Off-the-shelf software is pre-written software that is already available to buy and use, generally without modification, by any customer.

(c) **L3** A well-balanced discussion that considers both bespoke and off-the-shelf options with clear application to the hospital scenario (unusual workflows, patient safety, multi-site use, budget, timescale) and reaches a reasoned, justified conclusion. Technical terms used accurately throughout. **5-6**

(c) **L2** Some discussion of both options with partial application to the scenario; points may be generic or only loosely linked to the hospital's specific needs. A conclusion may be present but not fully justified. **3-4**

(c) **L1** Basic, mostly one-sided or generic statements about bespoke/off-the-shelf software with little or no application to the scenario given. **1-2**

(c) **L0** No creditable content. **0**

(c) Indicative content:

- Bespoke may be more suitable because the unusual workflows are not met by existing products, so an off-the-shelf package would not fully support the trust's operations.
- Bespoke can be tailored exactly to the trust's processes across all 12 sites and to patient-safety requirements, but is likely to cost more and take longer to develop and test.
- Off-the-shelf is cheaper and quicker to obtain, and is already tested/proven in use elsewhere, but may not support the unusual workflows without costly, risky customisation or manual workarounds.
- A conclusion should weigh patient safety/fitness for purpose against cost and time constraints, e.g. recommending bespoke if the unusual workflows are safety-critical, or a customised off-the-shelf product as a compromise.

(c) Answer: Model answer: bespoke software is likely the better recommendation given the unusual, safety-relevant workflows across 12 sites that no existing package supports, despite the higher cost and longer development time; off-the-shelf remains a valid alternative only if it can be customised to cover the unusual requirements within budget.

Question 2

(a) Source code is made publicly available/can be viewed by anyone (oe) **B1**

(a) ...and can usually be modified and redistributed freely (often at no cost) (oe) **B1**

(a) Answer: Open source software is software whose source code is publicly available so that it can be viewed, modified and redistributed freely.

(b) Source code is kept private/hidden by the owner/developer (oe) **B1**

(b) ...and users cannot legally view, modify or redistribute it without permission/a licence (oe) **B1**

(b) Answer: Closed source (proprietary) software is software whose source code is kept private by the owner, and cannot legally be viewed, modified or redistributed by users without permission.

(c) Advantage 1: any one from - free/low cost to acquire; can be customised to fit the business's needs; large community support/development; code can be inspected for security flaws; no vendor lock-in **B1**

(c) Advantage 2: a second distinct point from the list above **B1**

(c) Disadvantage 1: any one from - may lack official/guaranteed technical support; may require in-house technical expertise to customise/maintain; documentation may be poor or inconsistent; security vulnerabilities may be publicly known before being patched **B1**

(c) Disadvantage 2: a second distinct point from the list above **B1**

(c) Answer: Advantages: low/no cost; can be freely customised to the business's needs. Disadvantages: no guaranteed official support; may need in-house technical skill to maintain.

Question 3

(a) Software designed to help maintain, configure, analyse or optimise a computer system (oe) **B1**

(a) ...it carries out a single specific task, rather than a general task requested by the user (oe) **B1**

(a) Answer: Utility software is software designed to carry out a specific task that helps maintain, configure or optimise a computer system.

(b) i) Compression/archiving utility (oe) **B1**

(b) ii) Anti-virus/anti-malware utility (oe) **B1**

(b) iii) Defragmentation utility (oe) **B1**

(b) iv) Backup utility (oe) **B1**

(b) Answer: i) Compression utility ii) Anti-virus/anti-malware utility iii) Defragmentation utility iv) Backup utility

(c) HDDs use a moving read/write head over spinning platters, so fragmented data (split into non-contiguous blocks) increases seek/access time (oe) **B1**

(c) Defragmentation reorders data into contiguous blocks, reducing head movement and so reducing access time (oe) **B1**

(c) SSDs have no moving parts and can access any memory location in (near) constant time, so fragmentation does not increase access time / reorganising data brings little benefit and may cause unnecessary wear (oe) **B1**

(c) Answer: HDDs suffer from fragmentation because the read/write head must physically move further to reach scattered data, increasing seek time; defragmentation groups files into contiguous blocks to reduce this. SSDs have no moving head and access any location in roughly the same time, so fragmentation barely affects performance and defragmenting can add unnecessary wear.

Question 4

(a) One correct characteristic of RAD, from: heavy/repeated use of prototyping, refined based on user feedback; short development time/fast delivery; makes use of reusable components/tools; requires active, continuous user involvement **B1**

(a) A second, different correct characteristic from the same list. Maximum 2 marks; do not credit the same characteristic twice **B1**

(a) Answer: RAD makes heavy use of prototyping that is refined through repeated user feedback, and aims for fast delivery of a working system.

(b) RAD depends on regular/continuous input and feedback from users to refine prototypes at each stage (oe) **B1**

(b) Without this feedback, prototypes cannot be validated or improved against real user needs (oe, ft) **B1**

(b) This can lead to a final system that does not meet requirements, or to delays while the team waits for input, undermining the speed that is RAD's main benefit (oe, ft) **B1**

(b) Answer: RAD relies on continuous user feedback to refine each prototype; without available users, prototypes cannot be validated against real needs, risking a final system that does not meet requirements or delays that remove RAD's main speed advantage.

(c) RAD uses prototyping throughout/repeatedly, central to the process, refined via user feedback across the whole project (oe) **B1**

(c) Waterfall does not typically use prototyping, or uses it (if at all) only briefly at the start, since each stage is completed once, in order, before moving on (oe) **B1**

(c) Comparative point: RAD can therefore adapt to user feedback more easily/quickly than waterfall, which is more rigid once requirements/design are signed off (oe) **B1**

(c) Answer: RAD uses prototyping repeatedly and continuously throughout the project, refined by ongoing user feedback, whereas waterfall makes little or no use of prototyping since each stage is completed once before moving to the next; as a result RAD can adapt to user feedback far more easily than the more rigid waterfall model.

Question 5

(a) One correct agile principle, from: individuals and interactions valued over processes and tools; working software valued over comprehensive documentation; customer collaboration valued over contract negotiation; responding to change valued over following a fixed plan; iterative/incremental delivery in short cycles (sprints); continuous customer/user feedback **B1**

(a) A second, different correct agile principle from the same list **B1**

(a) A third, different correct agile principle from the same list. Maximum 3 marks; do not credit the same principle twice **B1**

(a) Answer: e.g. working software is valued over comprehensive documentation; customer collaboration is valued over contract negotiation; responding to change is valued over following a fixed plan.

(b) Point 1: agile develops software in short iterations/sprints, each delivering a working increment (oe) **B1**

(b) Development of point 1: so requirements can be reviewed and adjusted between sprints without restarting the whole project, unlike waterfall (oe, ft) **B1**

(b) Point 2: agile involves continuous/frequent feedback from the customer/user throughout development (oe) **B1**

(b) Development of point 2: so changes can be incorporated as they arise, rather than (as in waterfall) only being possible before design begins (oe, ft) **B1**

(b) Answer: Agile develops software in short sprints, each producing a working increment, so requirements can be reviewed and changed between sprints without restarting the project; agile also involves continuous customer feedback throughout, so changes can be incorporated as they arise, unlike waterfall where requirements are effectively fixed once the design stage begins.

(c) **L3** A well-balanced discussion that applies agile's characteristics directly to the fixed-price, fixed-deadline, legally constrained context, weighing genuine advantages and disadvantages, and reaching a reasoned, justified conclusion. **5-6**

(c) **L2** Some balanced discussion of agile's suitability, but with limited application to the fixed contract/legal deadline context; conclusion may be weak or generic. **3-4**

(c) **L1** Basic or one-sided statements about agile with little or no application to the scenario. **1-2**

(c) **L0** No creditable content. **0**

(c) Indicative content:

- Agile's flexible/evolving scope is hard to reconcile with a fixed-price contract, since the final requirements and total cost cannot be fully fixed upfront, which could lead to disputes over what has been delivered.
- A fixed legal deadline could still be met by fixing the time and cost (time-boxing sprints) and allowing scope/features to flex, delivering the highest priority features first (a 'fixed time, fixed cost, variable scope' approach).
- Agile's frequent releases and feedback can reduce the risk of building the wrong system, which is valuable in a large project.
- However, a project needing full upfront agreement, sign-off and heavy documentation for legal/audit purposes may be better suited to waterfall or a hybrid approach, since agile produces less upfront documentation.
- A reasoned conclusion should weigh the risk of scope/cost disputes against agile's ability to deliver a prioritised, working system by a fixed date.

(c) Answer: Model answer: agile could be made to work within a fixed contract by time-boxing sprints and letting scope flex (delivering the highest-priority features first to meet the legal deadline), but its evolving scope sits awkwardly with a fixed-price contract and heavy documentation/audit requirements, so a hybrid approach (e.g. fixed requirements and budget agreed upfront, delivered via agile sprints) may be the most balanced recommendation.

Question 6

(a) 1. Determine objectives, alternatives and constraints for that iteration (oe) **B1**

(a) 2. Identify and evaluate/resolve risks (risk analysis), often using prototyping (oe) **B1**

(a) 3. Develop and test/verify the next level (increment) of the product for that iteration (oe) **B1**

(a) 4. Review the iteration and plan the next loop (oe) **B1**

(a) Answer: Each loop: (1) determine objectives, alternatives and constraints; (2) identify and analyse/resolve risks, often via prototyping; (3) develop and test the next increment of the product; (4) review the results and plan the next loop.

(b) Each loop/iteration includes an explicit risk analysis phase, often using prototypes (oe) **B1**

(b) This allows risks to be identified and addressed early, before large amounts of time/money have been committed (oe, ft) **B1**

(b) Reducing the chance of costly failure later in a large, unfamiliar or technically complex project (oe, ft) **B1**

(b) Answer: Because every loop of the spiral includes a dedicated risk-analysis phase, often supported by prototyping, risks are identified and dealt with early, before large amounts of time and money have been committed, reducing the chance of costly failure later in a large or technically complex project.

(c) Requires specialist risk-assessment expertise/skills to be carried out properly (oe) **B1**

(c) It can be more expensive and time-consuming than simpler models due to the repeated risk analysis and prototyping, which may not suit smaller or low-risk projects (oe; any distinct valid disadvantage acceptable) **B1**

(c) Answer: The spiral model requires specialist risk-assessment expertise and repeated prototyping/analysis in every loop, making it more expensive and time-consuming than simpler models, which is not justified for smaller or low-risk projects.

Question 7

(a) One correct XP practice, from: pair programming; test-driven development (writing tests before code); continuous integration; refactoring; collective code ownership; small/frequent releases; on-site customer; simple design; coding standards **B1**

(a) A second, different correct XP practice from the same list **B1**

(a) A third, different correct XP practice from the same list. Maximum 3 marks; do not credit the same practice twice **B1**

(a) Answer: e.g. pair programming, test-driven development, continuous integration.

(b) Description: two developers work together at one workstation, one writing code (driver) while the other reviews it in real time (navigator) (oe) **B1**

(b) Errors/bugs are spotted and corrected earlier, as code is reviewed as it is written (oe) **B1**

(b) Knowledge of the codebase is shared between developers, reducing dependency on a single person and aiding training (oe; any one further distinct benefit acceptable) **B1**

(b) Answer: In pair programming, two developers share one workstation, with one writing code (driver) and the other reviewing it as it is written (navigator). Errors are caught earlier because the code is reviewed continuously, and knowledge of the codebase is shared between developers, reducing reliance on any one individual.

(c) Description: code changes are merged into a shared repository and automatically built/tested frequently, e.g. several times a day (oe) **B1**

(c) Integration problems/conflicts between different developers' code are identified quickly, while they are small and easy to fix (oe) **B1**

(c) This reduces the risk of large, difficult-to-diagnose integration problems occurring near the end of the project (oe) **B1**

(c) Answer: Continuous integration merges developers' code into a shared repository and automatically builds/tests it frequently. This means integration problems between different developers' code are found quickly while they are small, reducing the risk of large, difficult integration problems appearing near the end of the project.

Question 8

(a) **L4** A clear, justified recommendation that explicitly applies methodology characteristics to multiple scenario constraints (tight deadline, evolving features, security/regulatory needs), with balanced consideration of at least one alternative and a coherent, well-structured argument. **7-8**

(a) **L3** A reasonable recommendation with justification linked to most scenario factors, but with less developed evaluation of alternatives. **5-6**

(a) **L2** Some relevant points made with partial application to the scenario; justification is limited or one-sided. **3-4**

(a) **L1** Basic or generic statements about a methodology with little or no application to the scenario. **1-2**

(a) **L0** No creditable content. **0**

(a) Indicative content:

- An agile approach (e.g. Scrum) is likely most appropriate because features are expected to change based on user feedback, and agile's iterative sprints allow the app to be adapted after each release.
- Short sprints could deliver a minimum viable product within the 4 month deadline, with regular releases allowing early and continuous testing.
- However, security and regulatory compliance requirements are often non-negotiable/fixed, so these may need to be gathered and specified up front, and/or validated through a formal, dedicated review stage, suggesting a hybrid approach (agile development combined with a fixed security/compliance specification and review process) rather than pure agile.
- Waterfall is less suitable since it does not easily accommodate the expected changes to features, and risks missing the fixed 4 month deadline if all requirements must be fully fixed upfront.
- RAD could also be considered given the tight deadline and reliance on prototyping, provided users are available to give regular feedback; however it may not provide the rigorous documentation needed for regulatory compliance.
- A well-argued answer might recommend Scrum with an embedded, non-negotiable security/compliance review at each sprint, or a hybrid model combining agile delivery with waterfall-style compliance sign-off.

(a) Answer: Model answer: recommend an agile (e.g. Scrum) approach for the evolving features and tight 4 month deadline, combined with a fixed, non-negotiable security/regulatory specification validated at each sprint (a hybrid model), since pure waterfall would not meet the deadline or accommodate expected changes, and pure agile alone risks under-specifying compliance requirements.

Question 9

(a) **L4** A clear, well-justified recommendation combining a software type and licensing model, with detailed application to the bookshop's budget, standard requirements and desire for future control, and balanced consideration of trade-offs. **7-8**

(a) **L3** A reasonable recommendation linked to most scenario factors, with some consideration of trade-offs, but less fully developed. **5-6**

(a) **L2** Some relevant points with partial application to the scenario; justification limited or one-sided. **3-4**

(a) **L1** Basic or generic statements about software types/licensing with little or no application to the scenario. **1-2**

(a) **L0** No creditable content. **0**

(a) Indicative content:

- Off-the-shelf is likely appropriate given the limited budget and standard/typical requirements shared with other small shops, avoiding the high cost of bespoke development.
- Open source licensing could reduce ongoing licence costs and, since the source code is available, could give the owner (or a hired developer) some ability to make or commission future changes without being tied to a single vendor, addressing the desire for control.
- However, open source solutions may need technical expertise to adapt or maintain, which could add cost the owner did not expect.
- Closed source, off-the-shelf point-of-sale packages are common, cheaper up front than bespoke, well tested and supported, but the owner would have no direct control over future changes and would depend on the vendor's roadmap and licence terms.
- A balanced answer might recommend an open source, off-the-shelf EPOS (electronic point of sale) package as a compromise: low cost, standard functionality already built in, and some flexibility for future changes, while noting the trade-off in guaranteed technical support.

(a) Answer: Model answer: recommend an open source, off-the-shelf point-of-sale package, since off-the-shelf suits the limited budget and standard requirements, while the open source licence gives the owner some ability to adapt the software in future without being locked to one vendor; the main trade-off is reduced guaranteed technical support compared with a closed-source commercial product.

Question 10

Level 3 A sustained, well-evidenced evaluation considering multiple project contexts (e.g. safety-critical/regulated systems vs consumer apps), with a balanced argument on both sides and a clear, well-supported overall judgement. Technical vocabulary used accurately throughout. **7-9**

Level 2 Some evaluation with valid points on both sides, applied to at least one context; an overall judgement is present but less developed or not fully consistent. **4-6**

Level 1 Mostly one-sided or a generic list of features of agile and/or waterfall with little evaluative comment; a weak or absent judgement. **1-3**

Level 0 No creditable content. **0**

Indicative content:

- Case for agile making waterfall obsolete: agile's flexibility better matches fast-changing markets and evolving user expectations; iterative delivery gets working software to users sooner and incorporates feedback, reducing the risk of building the wrong product; agile fits well with modern continuous deployment/DevOps practices.
- Case against: waterfall's clear documentation, fixed scope/cost/schedule and formal sign-off stages remain valuable, or required, for safety-critical, highly regulated, or contractually fixed-price projects (e.g. medical devices, government/defence systems, embedded firmware) where requirements are well understood in advance and cannot change late in development.
- Some organisations lack the culture or skills for agile, or need extensive audit trails that waterfall's documentation-heavy stages naturally provide.
- Hybrid models (e.g. spiral, or agile combined with staged formal reviews) are often used in practice instead of either pure model, showing that the choice is not simply binary.
- Overall judgement: agile has become dominant in many sectors (e.g. web/app/consumer software) but waterfall, or waterfall-like staged approaches, persists and remains appropriate for certain project types, so it is not fully obsolete; the correct choice depends on the nature of the project rather than one model being universally superior.

Question 11

(a) Agile does not require a full specification/documentation to be produced upfront (oe) **B1**

(a) For a system handling sensitive personal data and exam results under strict legal (data protection/compliance) requirements, incomplete upfront analysis could result in security or compliance issues (or errors) being missed (oe, ft) **B1**

(a) Agile's flexible/evolving scope makes it harder to guarantee, in advance, that all legal/compliance requirements will definitely be met by a fixed deadline (oe; any distinct valid risk acceptable) **B1**

(a) Answer: Agile does not require a complete specification to be documented upfront; for a system with strict data-protection and exam-accuracy requirements, this could allow compliance or security issues to be missed until late in development, and agile's evolving scope makes it hard to guarantee in advance that all legal requirements will be met.

(b) Waterfall fixes requirements early and does not allow easy return to an earlier stage (oe) **B1**

(b) For a large, complex system like this, it is likely that not all requirements can be foreseen at the start, so errors/omissions in the specification may not be discovered until late testing, where they are costly to fix (oe, ft) **B1**

(b) Stakeholders (students/staff) get no working system to give feedback on until very late in the project, increasing the risk of a mismatch with actual needs (oe; any distinct valid risk acceptable) **B1**

(b) Answer: Waterfall fixes requirements early with no easy return to a previous stage; for such a large, complex system it is unlikely all requirements can be foreseen at the start, so errors may only be found in late testing, where they are expensive to fix, and stakeholders get no working system to react to until very late in the project.

(c) **L3** A coherent, plausible hybrid approach is described with clear features drawn from two named methodologies, and the answer clearly explains how it mitigates both the agile-related risk from (a) and the waterfall-related risk from (b). **5-6**

(c) **L2** A hybrid approach is described with some relevant features from two methodologies, with partial explanation of how it addresses the risks in (a) and/or (b). **3-4**

(c) **L1** A vague or generic reference to combining methodologies, with little detail or explanation of how the risks are addressed. **1-2**

(c) **L0** No creditable content. **0**

(c) Indicative content:

- An initial waterfall-style stage is used to gather and formally document full requirements and the legal/compliance/security specification up front, giving the rigour, audit trail and sign-off needed for a compliance-critical system.
- The system is then developed using agile sprints/iterations within that fixed overall specification, allowing incremental delivery, regular testing and stakeholder (student/staff) feedback on usability, while the core compliance requirements stay fixed.
- This reduces the agile-related risk from (a): legal/compliance requirements are fixed and documented upfront rather than being left to evolve.
- This reduces the waterfall-related risk from (b): because users see and give feedback on working increments throughout development rather than only at the very end, undiscovered requirement errors are more likely to be found and fixed early, while still being cheap to correct.

(c) Answer: Model answer: use a waterfall-style stage first to fix and formally document the legal/data-protection/exam-accuracy requirements, then build the system through agile sprints within that fixed specification, giving stakeholders working increments to review throughout. This fixes compliance requirements upfront (addressing the agile risk from (a)) while still giving early, iterative feedback (addressing the waterfall risk from (b)).