LESSON 71

Staged Design

40 minutes | Software Engineering and Testing | Outcome 1.1 | Outcome 1.19 | Outcome 1.21 | Outcome 1.23 | Outcome 2.19 | Outcome 2.22

WHAT THIS LESSON DOES

In this lesson, you'll explore a structured approach to software development through distinct phases. Follow step-by-step guidance to understand each stage, from gathering requirements to deployment and maintenance, and reflect on applying this method to a simple project.

CURRICULUM OUTCOMES

Outcome	What the specification asks for
1.1	describe a systematic process for solving problems and making decisions
1.19	identify features of both staged and iterative design and development processes
1.21	identify alternative perspectives, considering different disciplines, stakeholders and end users
1.23	reflect and communicate on the design and development process
2.19	test solutions and decisions to determine their short-term and long-term outcomes
2.22	explain the different stages in software testing

WHAT STUDENTS SHOULD BE ABLE TO DO

- Grasp the fundamental concept of staged design and its linear, sequential approach to software development.
- Identify the key phases of the waterfall model and their specific roles in the development process.
- Analyse the advantages and limitations of using staged design in software projects.
- Apply the staged design methodology to conceptualise a simple project through its distinct stages.
- Reflect on the suitability of the waterfall model for projects with fixed versus evolving requirements.

BEFORE THE LESSON

- This is a discussion and theory lesson with no coding. No environment or accounts needed. Have the six phase names ready to display or write up so students can see the whole sequence at once.
- Decide the shared example project you will map to the stages during the reflection, for instance a task-tracking app or a personal website, so the class has a common reference point.

Staged Design

HOW THE LESSON RUNS

Introduction
Understanding Staged Design
Requirements Gathering
Design
Implementation
Testing
Deployment
Maintenance
Reflection Activity
Wrap Up

TEACHING MOVES

- 2. Understanding Staged Design. Draw the waterfall as steps going down and stress the one rule that defines it: each phase finishes before the next starts, and you do not easily go back. That single constraint explains every advantage and limitation that follows.
- 3. Requirements Gathering. Anchor this as the foundation. Ask what happens if a requirement is missed here and only discovered at testing. The cost of a late change is the whole argument for getting this stage right first.
- 4. Design. Separate design from coding explicitly. Design is the blueprint: architecture, interface, data structure, decided before a line is written. Students often assume you design by coding, so name the difference.
- 5. Implementation. Point out that in waterfall coding happens only after design is signed off, unlike the trial-and-error most students use. Link to their own project work later where planning before coding pays off.
- 6. Testing. Name the levels: unit, integration, system. Stress that in strict waterfall testing is a distinct phase after all code is written, not something interleaved. Ask why finding a bug this late is expensive.
- 7. Deployment. Distinguish deployment from finishing the code. Releasing means servers, configuration, user training, a real handover. Keep this brief unless students ask.
- 8. Maintenance. Break maintenance into corrective, adaptive and perfective so students see it is ongoing work, not just bug fixing. Note that most of a product's lifespan is here.
- 9. Reflection Activity. Have students map the shared example project onto all six stages, then state one situation where waterfall suits the job and one where it does not. Circulate and push for the reasoning, not just labels.

WHAT TO WATCH FOR

- Students treat the six phases as things that happen at once or overlap freely, missing that waterfall's defining feature is strict sequence with no easy return. Ask directly: can you start coding while requirements are still being gathered? In strict waterfall, no. Contrast this with how they actually work to make the rigidity visible.
- Students think waterfall is simply the correct or best way to build software, rather than one model with clear trade-offs. In the reflection, require one project where fixed requirements make waterfall sensible and one where changing requirements make it a poor fit. Force both sides.
- Students conflate the design phase with writing code, assuming design means starting to build. Show a wireframe or architecture sketch and point out no code exists yet. Design is deciding what to build before building it.

DIFFERENTIATION

- Emerging: Give the six phase names in order and ask the student to write one sentence per phase in their own words before attempting the reflection.
- Developing: Ask them to identify which single phase, if rushed, would most damage the whole project, and justify the choice.