OVERVIEW

Development Approach

How a team chooses the way it builds software, in four parts. Methodology concepts: process vs. process model vs. methodology, what a methodology addresses, its ten elements, and the two life cycles of an IT project (PLC for project requirements, SDLC for product requirements). Software development processes: plan-driven waterfall (when it fits and how it fails) vs. evolutionary/iterative development with its four phases (inception, elaboration, construction, transition). Agile: the 4 values and 12 principles, incremental development, XP practices and Scrum (roles, artifacts, sprints, backlog). DevOps: closing the gap between development and operations, and how project management adapts to a CI/CD pipeline.

The exam angle is choosing an approach for a situation and justifying it: match stable, known requirements to waterfall and evolving, uncertain ones to iterative or agile, and say why.

Process → model → methodology

Process = the activities; process model (life cycle) = an abstract shape like waterfall or iterative; methodology = the concrete "how we work around here".

PLC vs. SDLC

The project life cycle targets project requirements (time, cost); the SDLC targets product requirements. You need both, integrated.

Waterfall fits the known

Clear stable requirements, proven technology, low complexity, adequate time. Fails late: late integration, late risk resolution.

Iterative fits the uncertain

Unclear or evolving requirements, new technology, high complexity, short schedule. Phases: inception → elaboration (most critical) → construction → transition.

Agile = 4 values, 12 principles

Individuals over process, working software over documents, collaboration over contracts, responding to change over the plan. Waterfall fixes scope; agile fixes time and cost.

XP vs. Scrum

XP prescribes engineering practices (TDD, CI, refactoring, pairing); Scrum is a planning framework (backlog, sprints, Product Owner, ScrumMaster) with no technical practices.

HOW THE LECTURE IS ORGANIZED

Slides 1–14 Methodology concepts, elements of a methodology, PLC vs. SDLC

Slides 15–32 Plan-driven waterfall vs. evolutionary/iterative development

Slides 33–47 Agile: manifesto, principles, incremental development

Slides 48–55 Extreme Programming practices

Slides 56–74 Scrum: roles, backlog, sprints, agile vs. waterfall

Slides 75–98 DevOps and DevOps project management

EXAM FOCUS

Midterm 251 (both versions) asked when an iterative approach is most beneficial (MCQ 6) and, in the Edugate case, whether to recommend Agile, Waterfall or Hybrid and why (Case Q2, from the instructor's key: Agile/Scrum because requirements evolve). Final 251 Q1.4 asked for a PM tool or technique for a complex software project; Final 242 Q1.2 needs the milestone definition from slide 10.

HOW TO READ THIS PAGE

Every original slide is shown unchanged. Yellow marks on a slide point to a numbered note: hover or tap a mark to jump to its note, and hover a note to see its marks. Questions tagged **Past exam** appeared on a real paper; **Prediction** questions are likely angles, not confirmed. Pick an answer to each question: a wrong pick tells you why it is wrong, so you can try again.

SLIDE 1 / 98 **Slide 1**



Development Approach

SE423: Software Project Management

1

NOTES

1 The deck has four parts: **methodology concepts**, **software development processes** (waterfall vs. iterative), **agile** project management (XP and Scrum), and **DevOps**. The middle two carry most of the exam questions.

SLIDE 2 / 98 **Outline**

Outline

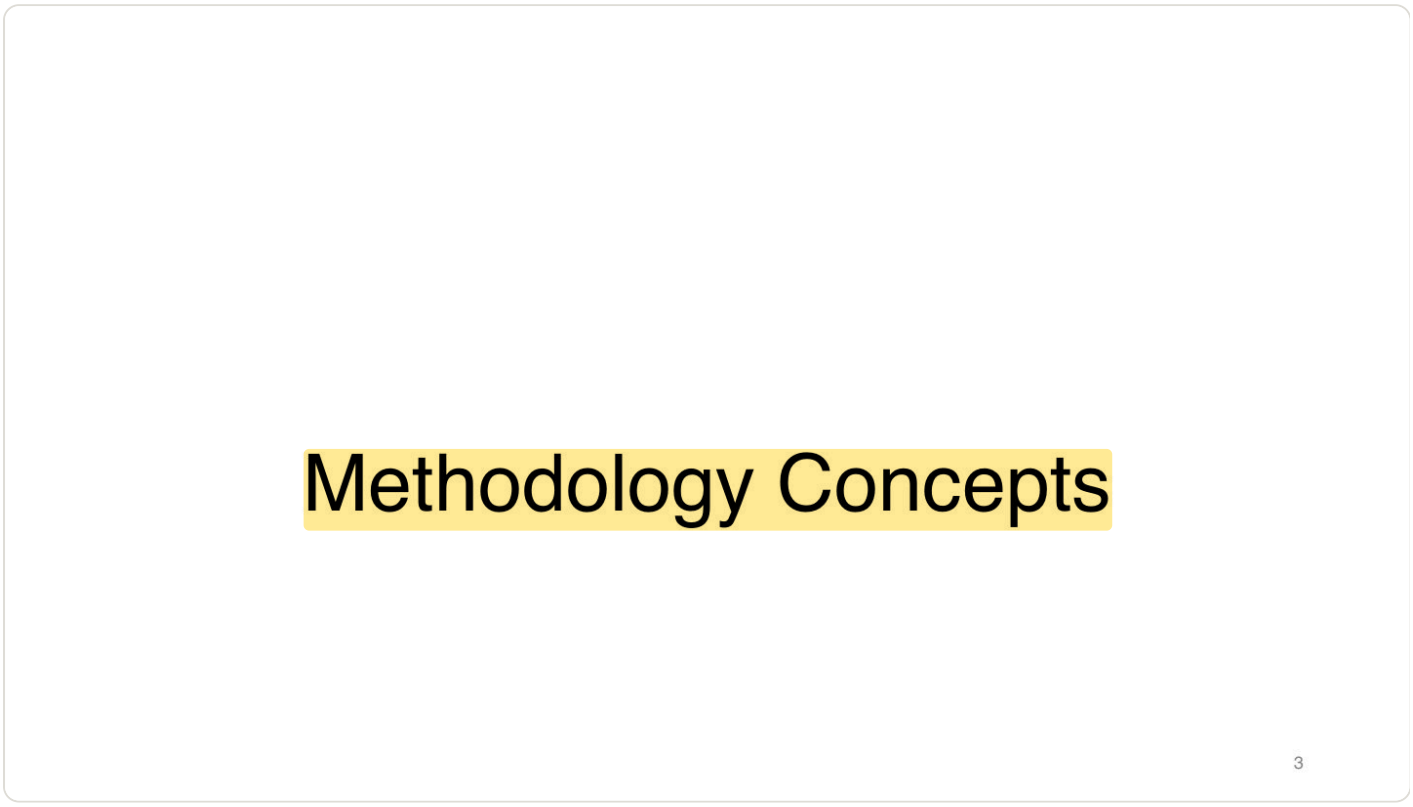
- Methodology Concepts
- Software Development Process
- Agile Project Management
- DevOps

2

NOTES

1. **Methodology concepts**: the vocabulary (process, model, methodology) and what a methodology contains.
2. **Software development process**: plan-driven waterfall vs. evolutionary/iterative.
3. **DevOps**: joining development and operations, and what that means for the PM.

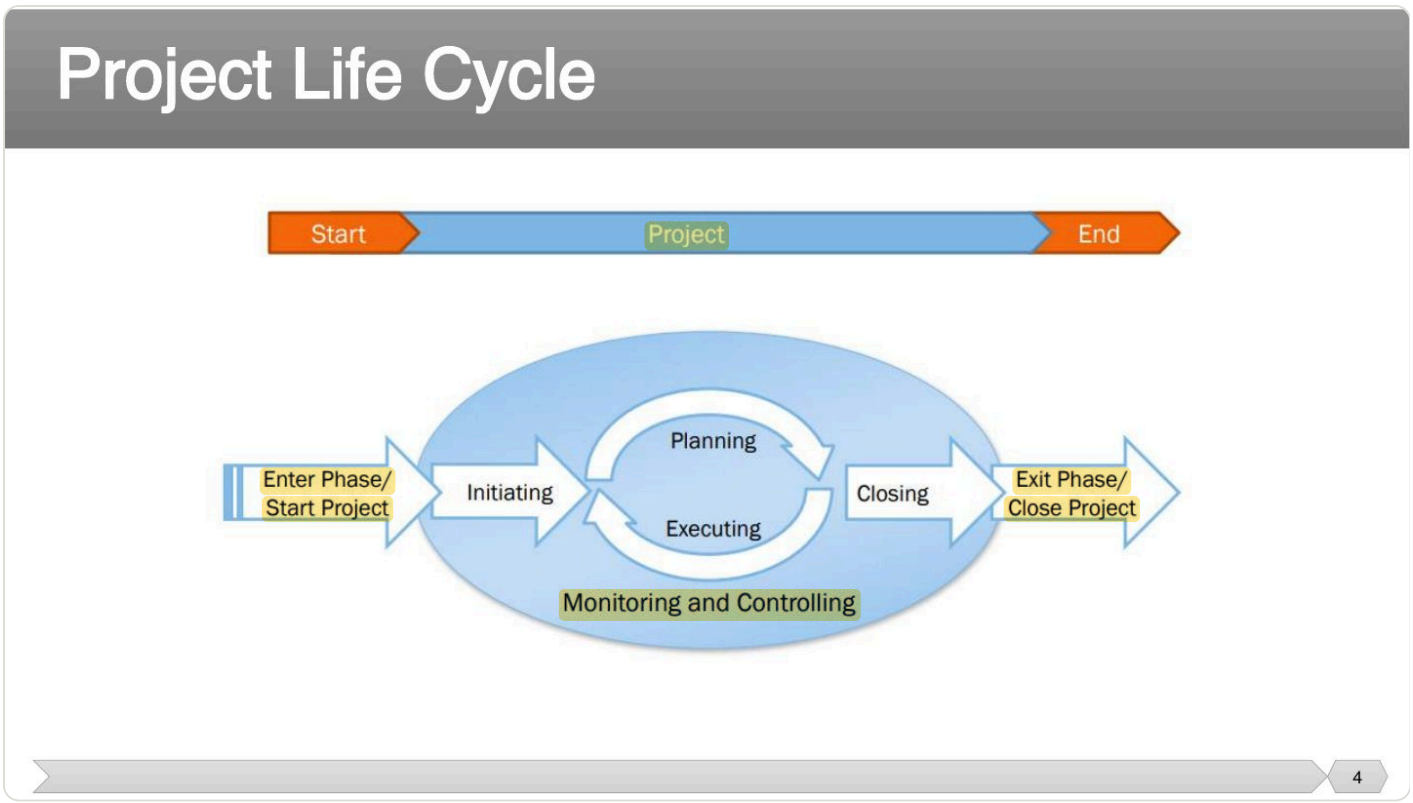
SLIDE 3 / 98 **Slide 3**



NOTES

1. Section divider: **Methodology Concepts**.

 Project Life Cycle



NOTES

1. The **project life cycle** as process groups. **Initiating** starts the phase or project (charter, stakeholders).
2. **Planning** and **executing** loop around each other, wrapped by **monitoring and controlling**, which compares progress with the plan and triggers re-planning.
3. **Closing** ends the phase or project (acceptance, lessons learned). Every development approach in this deck still sits inside this cycle.

EXAM ANGLE

Prediction Q1

Which process group compares progress with the plan and triggers re-planning?

A Executing

> That does the work.

B Closing

> That ends it.

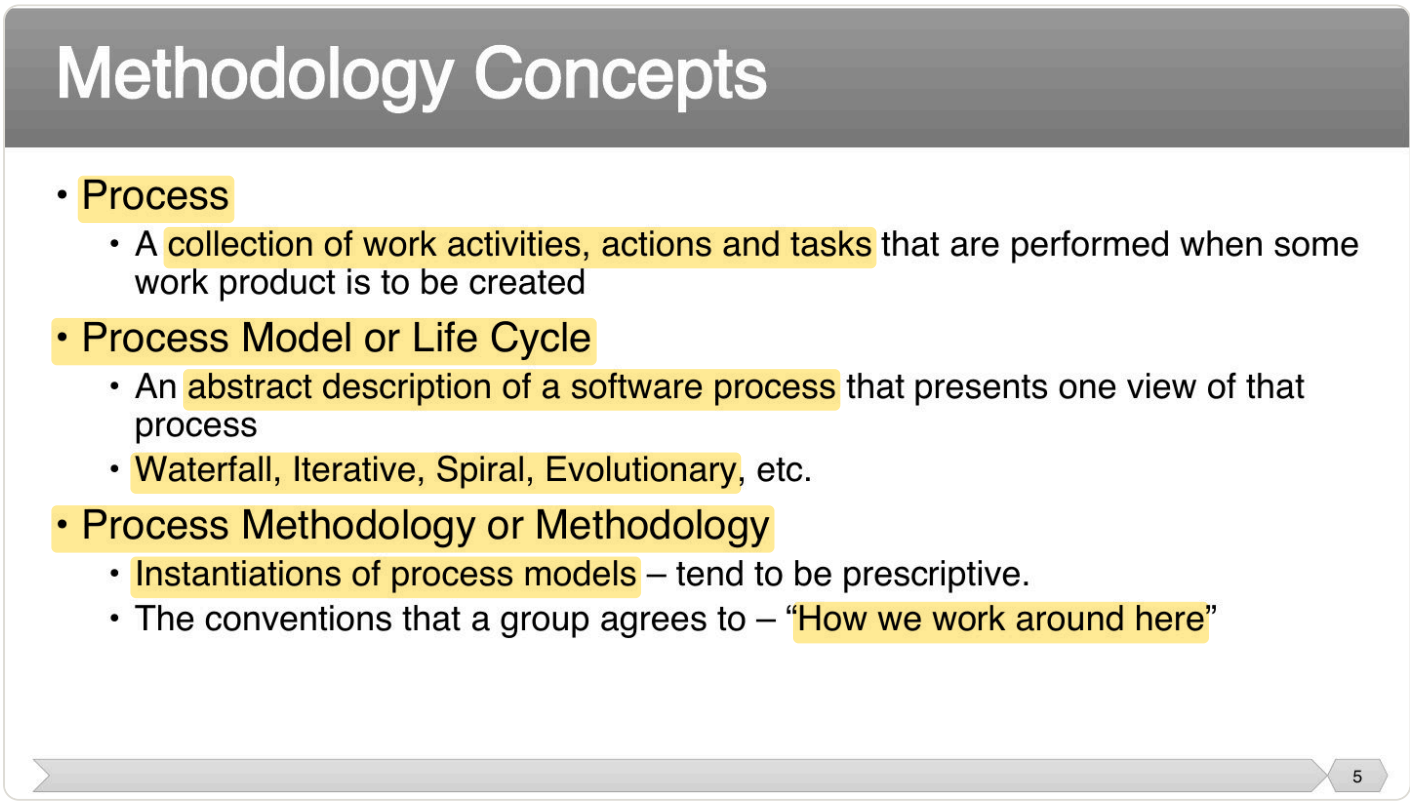
C Monitoring and controlling

> It wraps around planning and executing.

D Initiating

> That starts the phase or project.

SLIDE 5 / 98 **Methodology Concepts**



NOTES

1 A **process** is what actually gets done to produce something. Writing and reviewing a requirements document is a process: it's the real work, not a description of it.

2. A model is a general shape, like a recipe type: “iterative” tells you the idea without saying how your company does it. That’s why it’s called an **abstract description**.

3. A **methodology** is the model made specific, with your team’s rules, roles and habits. So Scrum at your company is a methodology, while “iterative” is the model behind it.

EXAM ANGLE

Prediction Q1

Waterfall, iterative and spiral are examples of:

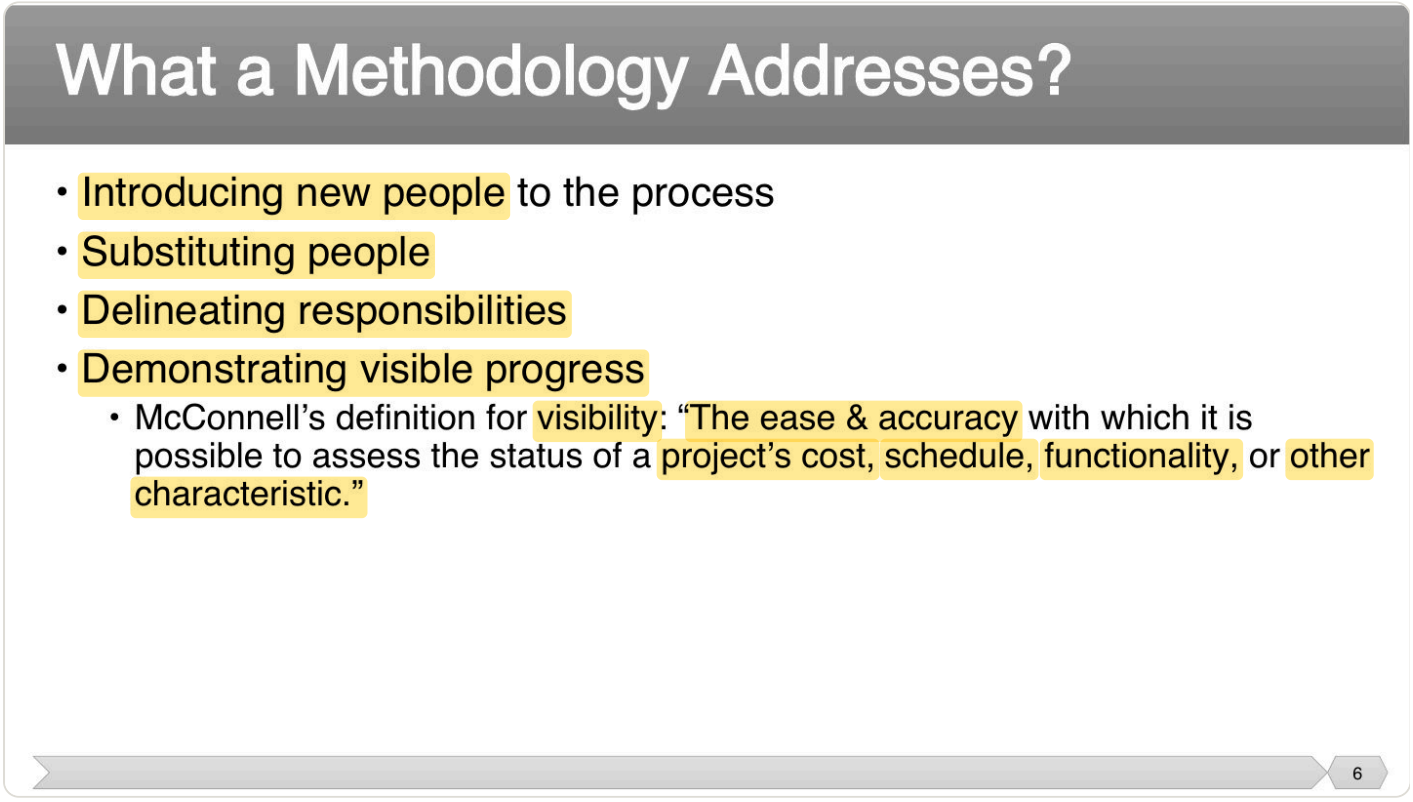
A Tools

B Milestones

C Methodologies

D **Process models (life cycles)**

SLIDE 6 / 98 **What a Methodology Addresses?**



NOTES

1. If a new hire can’t find out how things work, they’re slow for months. A methodology writes down how work is done, so **new people** ramp up faster, and a replacement can step in.

2. Managers need to know how a project is doing without guessing. **Visibility** means status is easy to see and accurate, so problems get noticed while they can still be fixed.

EXAM ANGLE

Prediction Q1

A new developer learns how the team works from its documented method. What does this help with?

A Late risk resolution

That's a waterfall symptom.

B Scope creep

That's uncontrolled growth in scope.

C Team storming

That's a stage of team development.

D Introducing new people to the process

It's the first item on the slide.

Prediction Q2

What does McConnell mean by visibility?

A How many people work in open offices

That's not the meaning.

B How well marketing promotes the product

That's not the meaning.

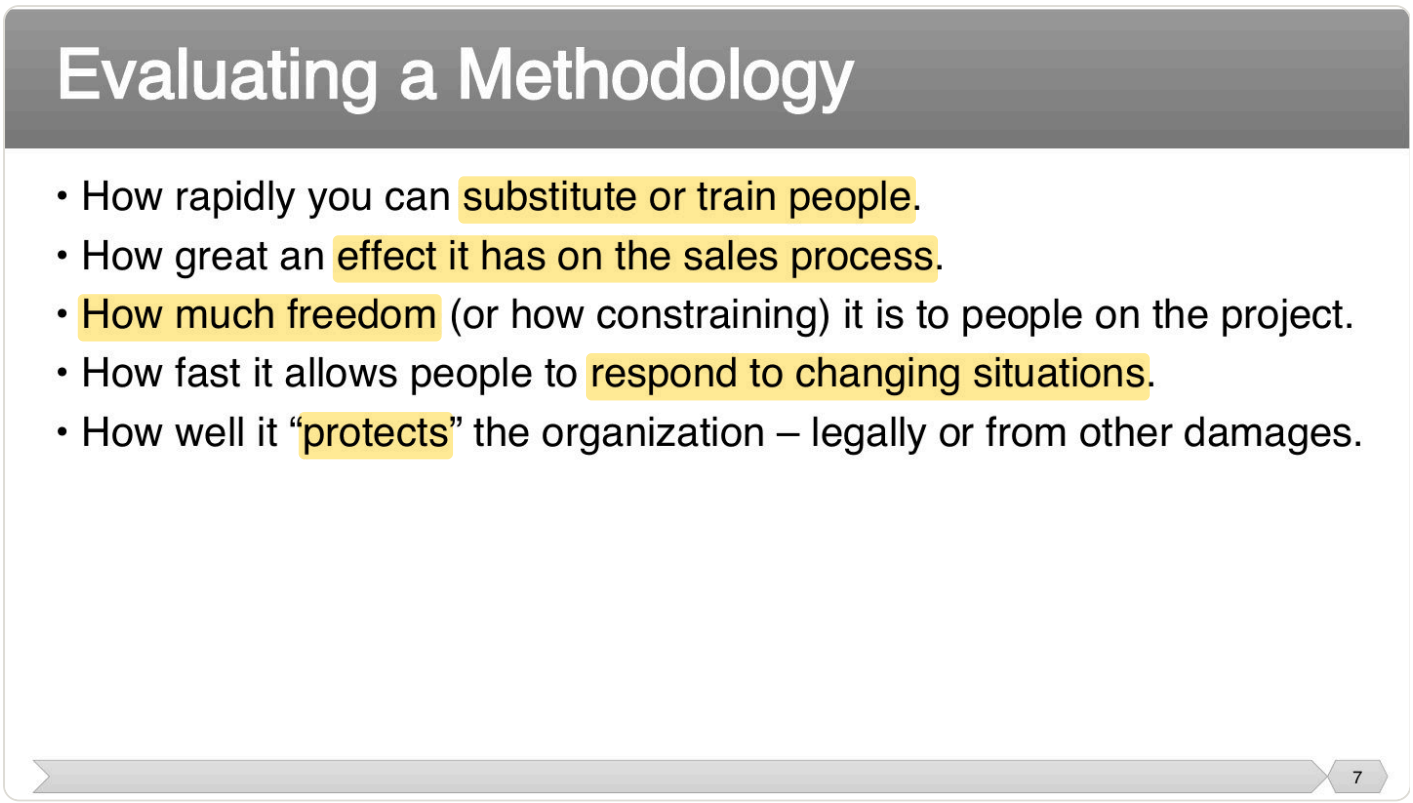
C How large the dashboard is

Display size isn't the point.

D How easily and accurately you can assess a project's status

That's his definition.

 Evaluating a Methodology



NOTES

1. If one person leaves and work stops, the method is failing you. A good one makes roles clear enough that you can **substitute** someone without losing weeks...
2. ...its **effect on the sales process** (can you show clients a credible way of working?)...
3. Too many rules slow people down and frustrate them, while too few cause chaos. This criterion asks you to judge that balance, or how much **freedom** people have...
4. If each change means a long approval process, you can't react when something shifts. How well you **respond to change** is one of the main differences between methods...
5. Records of who approved what help if a dispute goes to court. A method's paperwork can be annoying day to day and valuable when it **protects** the organization.

EXAM ANGLE

Prediction Q1

Documented approvals later help a company in a legal dispute. Which criterion is this?

A How much freedom it gives

That's a different criterion.

B How quickly you can train people

That's a different criterion.

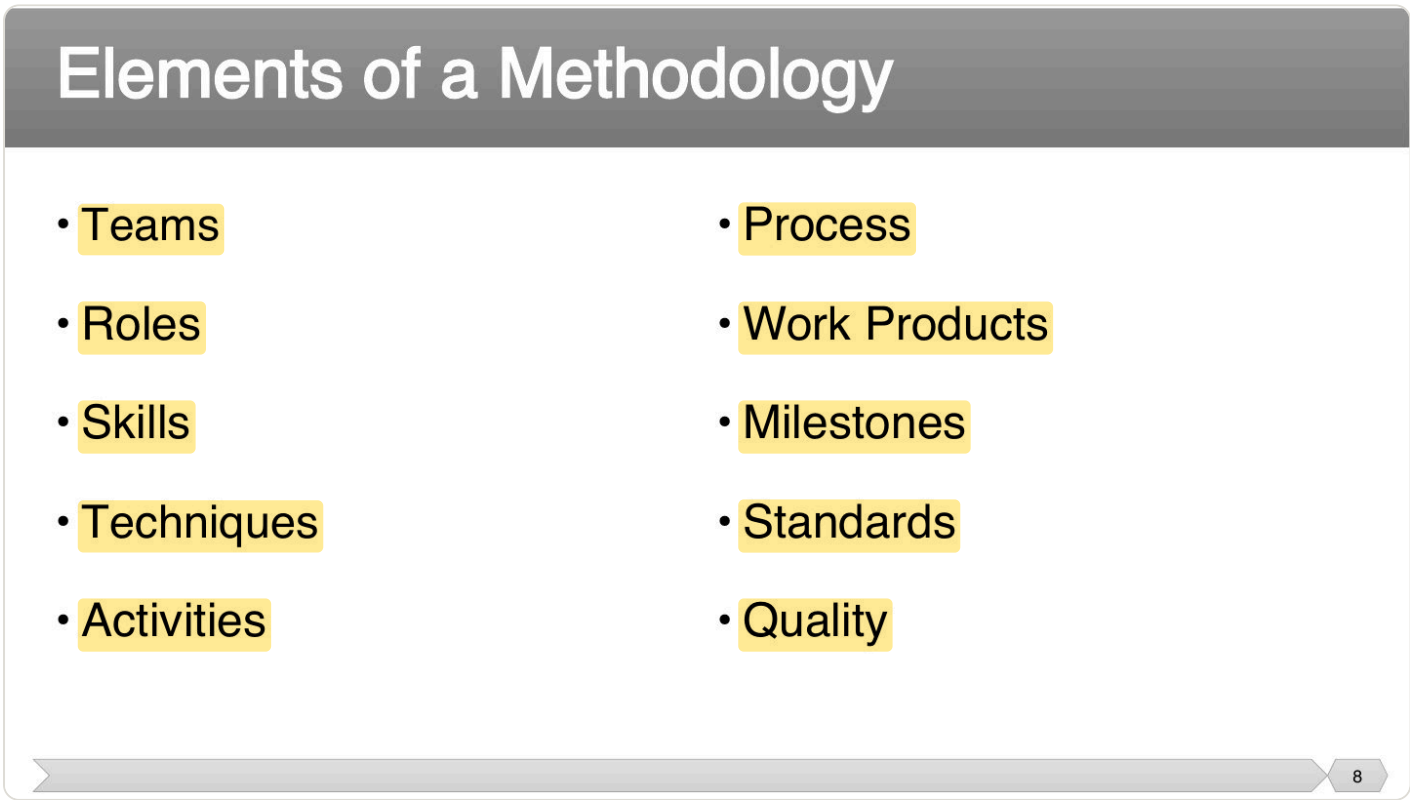
C **How well it protects the organization**

It's the slide's last criterion.

D Its effect on the sales process

That's a different criterion.

SLIDE 8 / 98 **Elements of a Methodology**



NOTES

1 You don't need to memorize these as one list. The left set is about people: who does the work and what they know. The **ten elements** are easier to recall in those groups.

2 This set is about the work itself: what order it's done in, what comes out of it, and what counts as good enough. Slides 10–13 define each.

EXAM ANGLE

Prediction Q1

Which is NOT one of the elements of a methodology?

A Roles

> This is on the list.

B Standards

> This is on the list.

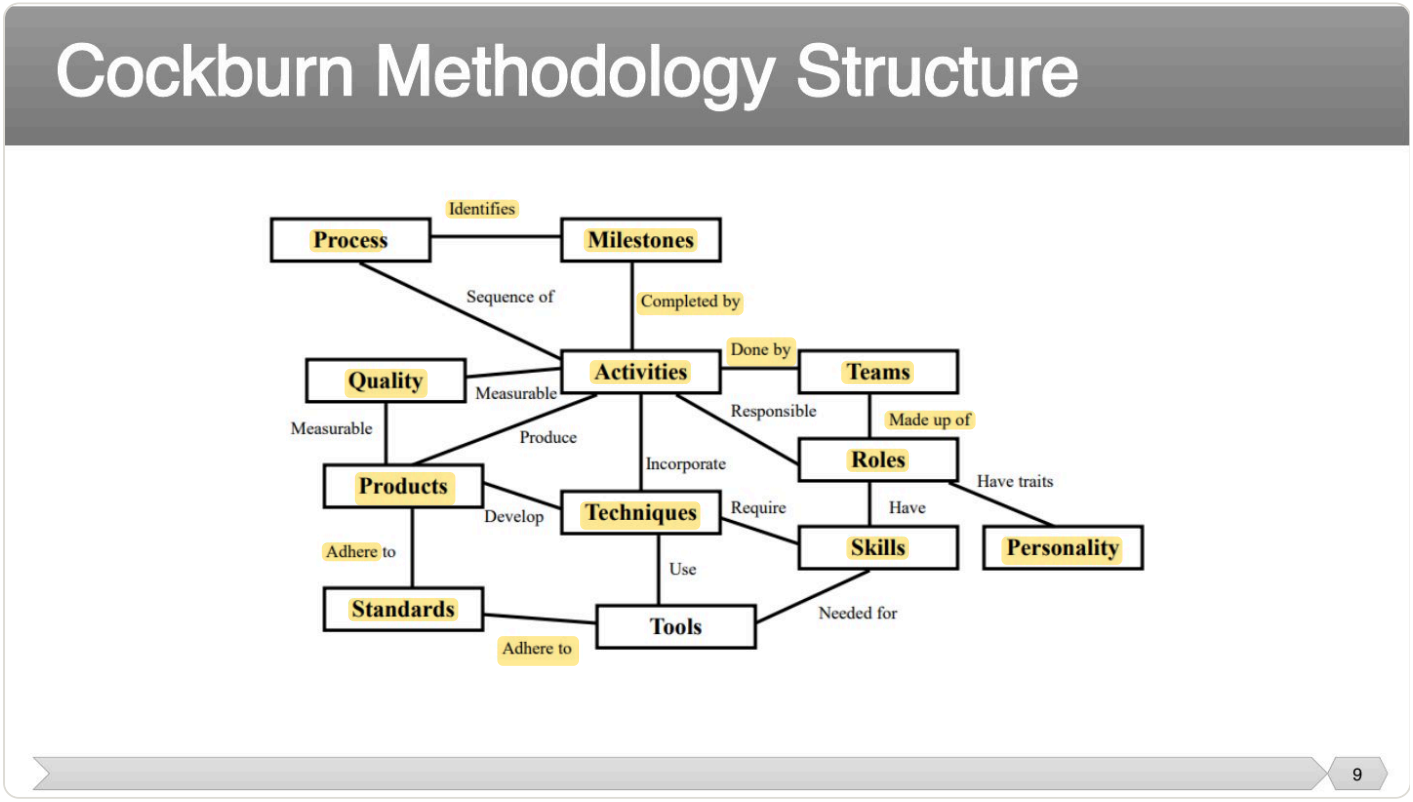
C **Dividends**

> That isn't on the list.

D Milestones

> This is on the list.

SLIDE 9 / 98 **Cockburn Methodology Structure**



NOTES

1. Read the diagram as sentences: the **process** is a sequence of **activities**, completed by **milestones**; activities produce **products** and incorporate **techniques**.
2. **Teams** are made of **roles**; roles have **skills** and are matched to **personalities**; techniques require skills and use **tools**.

EXAM ANGLE

Prediction

Q1

According to the Cockburn diagram, what do activities produce?

A Milestones only

> Milestones complete activities.

B **Products**

> The diagram reads: activities produce products.

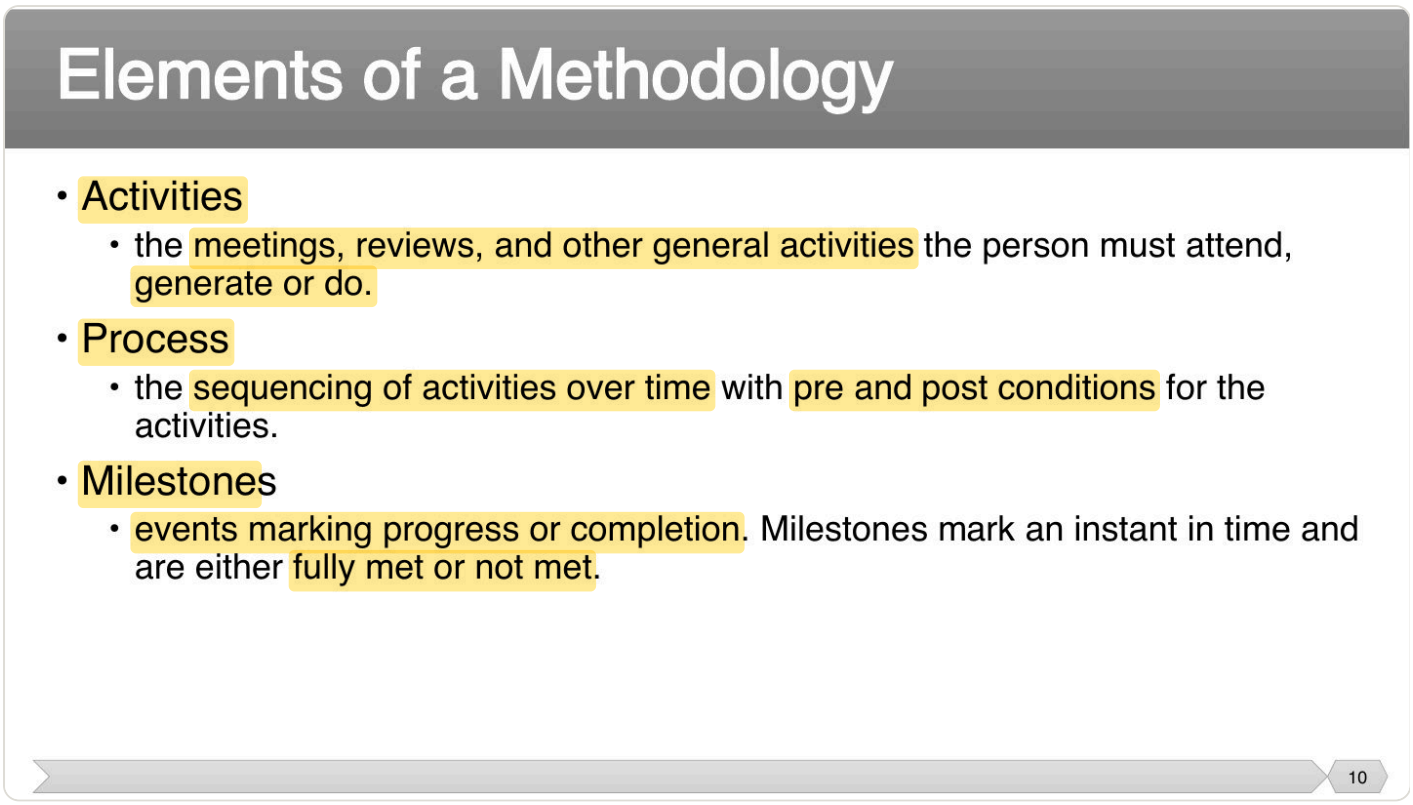
C Personalities

> Roles are matched to personalities.

D Standards

> Products adhere to standards.

 Elements of a Methodology



NOTES

1. Think of your calendar: reviews and meetings count as **activities** in the methodology too, since they take real time and need planning.
2. Activities alone don't tell you the order. **Process** says what must be finished before something can start, such as code review before merging.
3. A **milestone** isn't work that takes time. It's a moment you pass or don't. That's why "60% met" makes no sense: either the design was approved today or it wasn't.

TIPS & TRICKS

- **MAP**: Milestones, Activities, Process.

EXAM ANGLE

Past exam · Final 242 · Q1.2 Q1

Explain what is a Gantt chart and describe its significance in planning and tracking project activities. In your answer, define the terms milestone and dependency, and provide one example of each as they would appear in a Gantt chart.

Answer: A Gantt chart is a bar chart with activities down the side and a time scale across, where each bar shows an activity's start, finish and duration; it makes the schedule easy to plan and to track, because actual progress is shaded against the planned bars. A milestone is a significant event with zero duration, either fully met or not, shown as a diamond, e.g. "Requirements signed off". A dependency is a logical relationship where one activity can't start (or finish) until another does, shown as an arrow, e.g. "Coding" can't start until "Design approved" finishes.

Prediction Q2

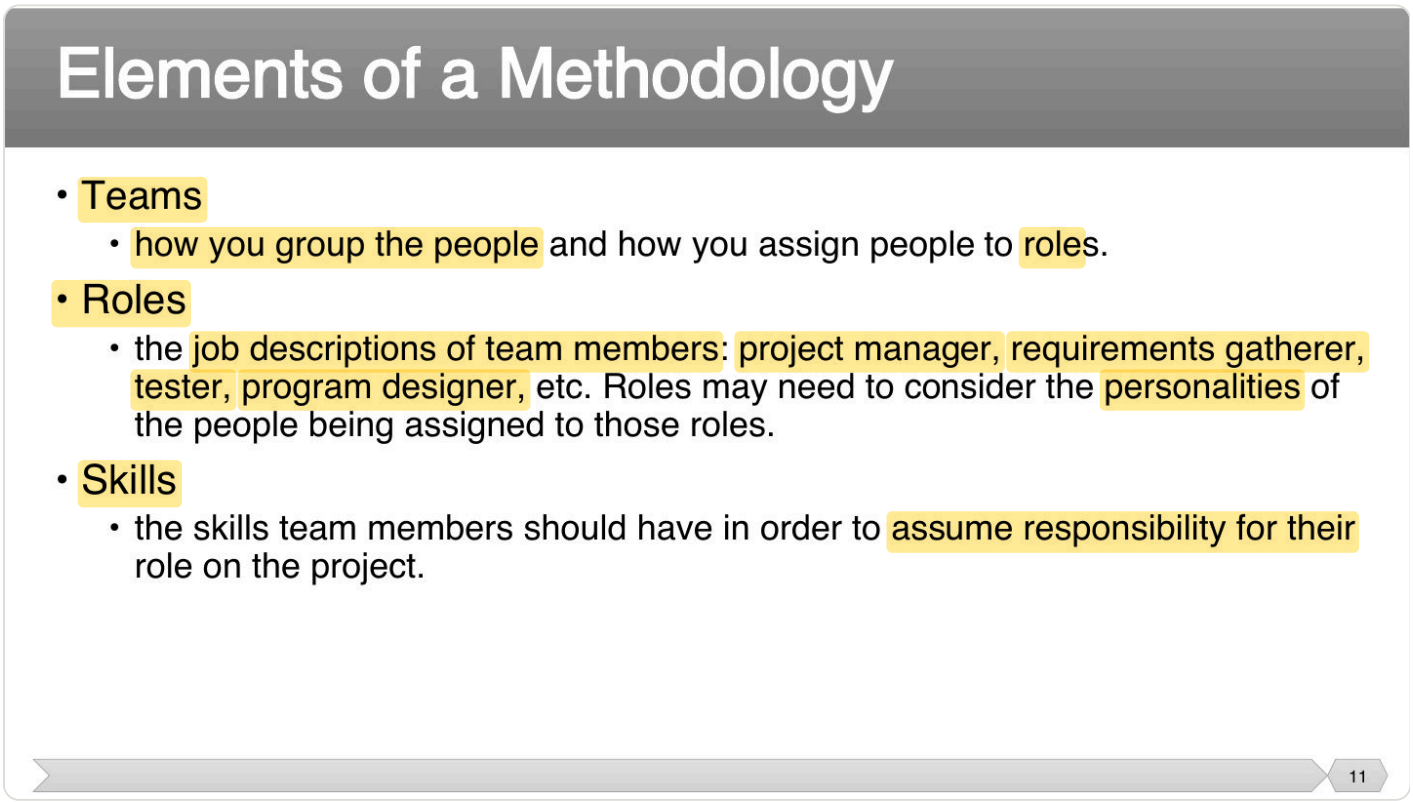
A milestone can be 60% met.

T True

F **False**

> Milestones mark an instant in time and are either fully met or not met.

SLIDE 11 / 98 **Elements of a Methodology**



NOTES

1. Who works with whom affects speed and quality. Putting testers with developers gives different results from keeping them apart, so **teams** are a real design choice.
2. A **role** is a job to be done, not a person, and one person may have several. Matching roles to **personalities** matters because someone who hates conflict may struggle as the negotiator.
3. Giving someone a role they can't do helps nobody. Listing **skills** per role tells you who needs training and who to hire.

TIPS & TRICKS

- **STaR**: Skills, Teams, Roles.

EXAM ANGLE

Prediction Q1

"How you group people and assign them to roles" defines:

A Skills

> Skills are what people need for a role.

B Process

> That's the order of activities.

C Standards

> That's what's permitted in a product.

D Teams

> That's the slide's definition.

Prediction Q2

Role assignment may need to consider people's personalities.

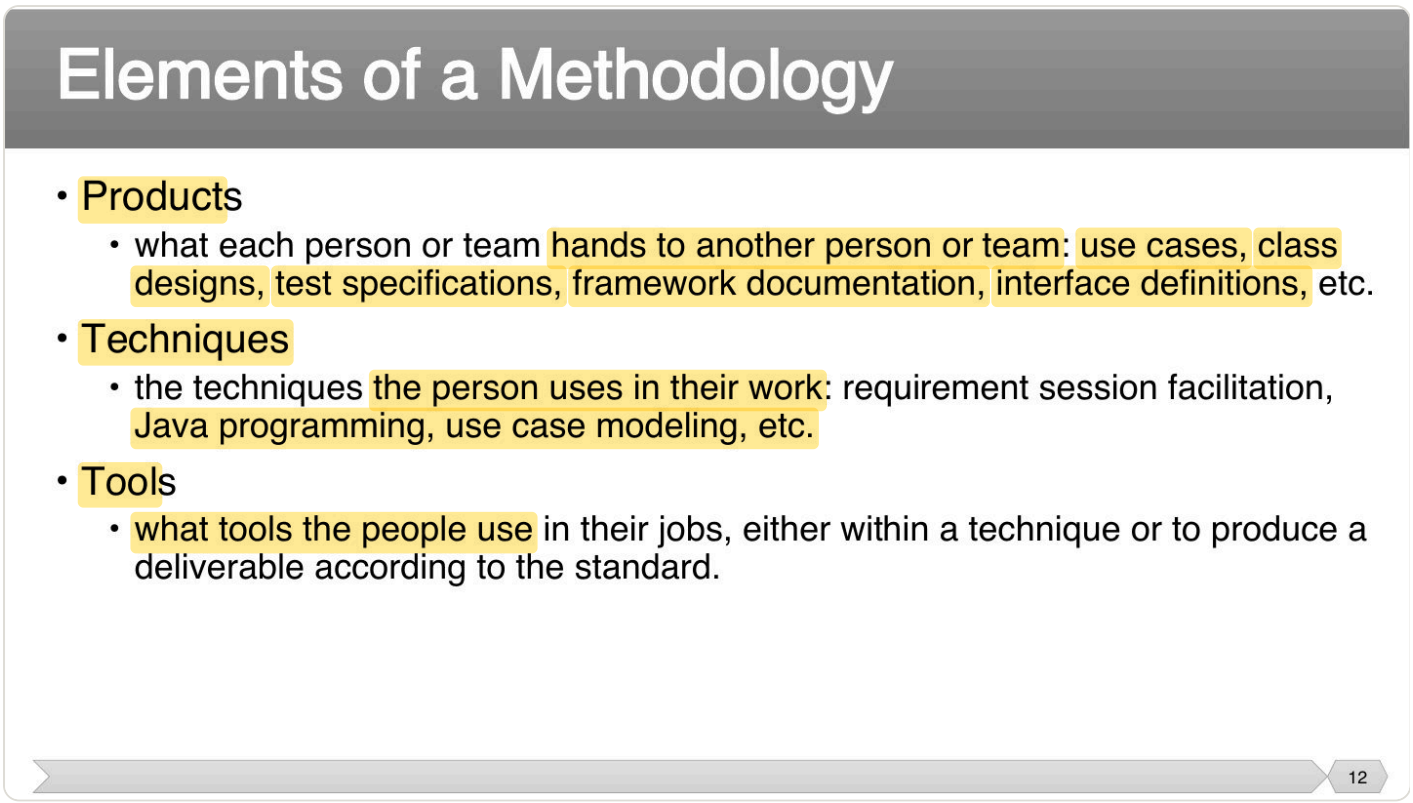
T True

> The slide says roles may need to consider this.

F False

> The slide mentions it directly.

 Elements of a Methodology



NOTES

1. Ask “what does the next person need from me?”. The tester needs test specifications, and that hand-over is a **product**. If it’s poor, the next person’s work suffers.
2. Two people can make the same product differently. **Techniques** are how the work is done, like modeling use cases, so a methodology can prescribe skills rather than only results.
3. A technique is the method, and a **tool** is what helps you apply it. Use case modeling is a technique, while the diagram software is a tool.

TIPS & TRICKS

- **PuTT**: Products, Techniques, Tools.

EXAM ANGLE

Prediction Q1

Use cases handed from an analyst to a designer are an example of:

A Techniques

> A technique is how the work is done.

B Tools

> Tools are what people use to do the work.

C Products

> A product is what one person hands to another.

D Roles

> A role is a job description.

Prediction Q2

Use case modeling is an example of a:

A Role

> A role is a job description.

B Milestone

> A milestone is an instant in time.

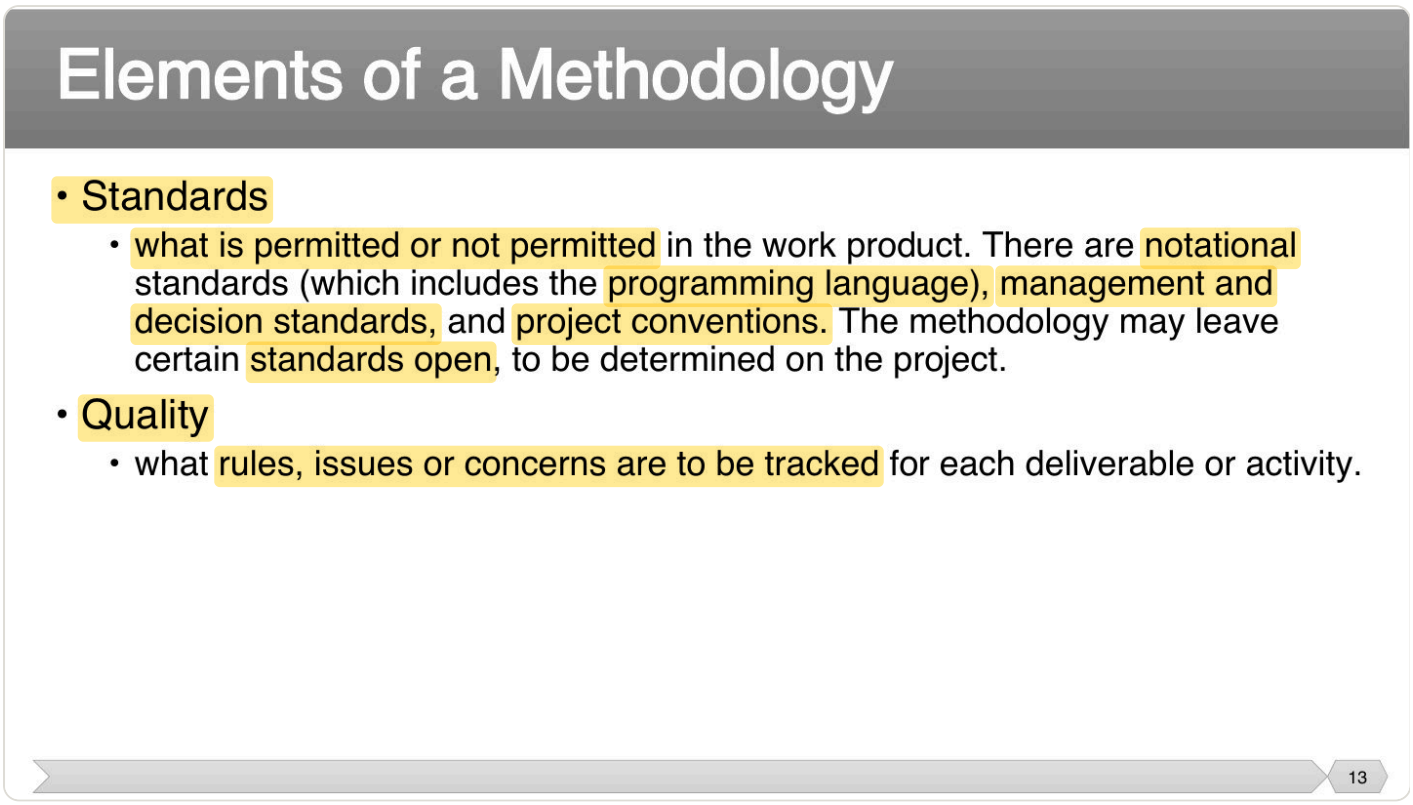
C Tool

> The software used for it would be the tool.

D Technique

> It’s how the person does the work.

 Elements of a Methodology



NOTES

1. Why have rules on the work product? Everybody can read and combine each other's work more easily. **Notational** standards are about how things are written, like which language or diagram style. Leaving some **open** allows projects to decide for themselves.

2. **Quality**: which rules, issues or concerns are tracked for each deliverable or activity (e.g. code review required, test coverage).

EXAM ANGLE

Prediction Q1

A team requires all code to be in Java. Which type of standard is this?

A Notational

> Notational standards include the programming language.

B Quality

> That's what gets tracked per deliverable.

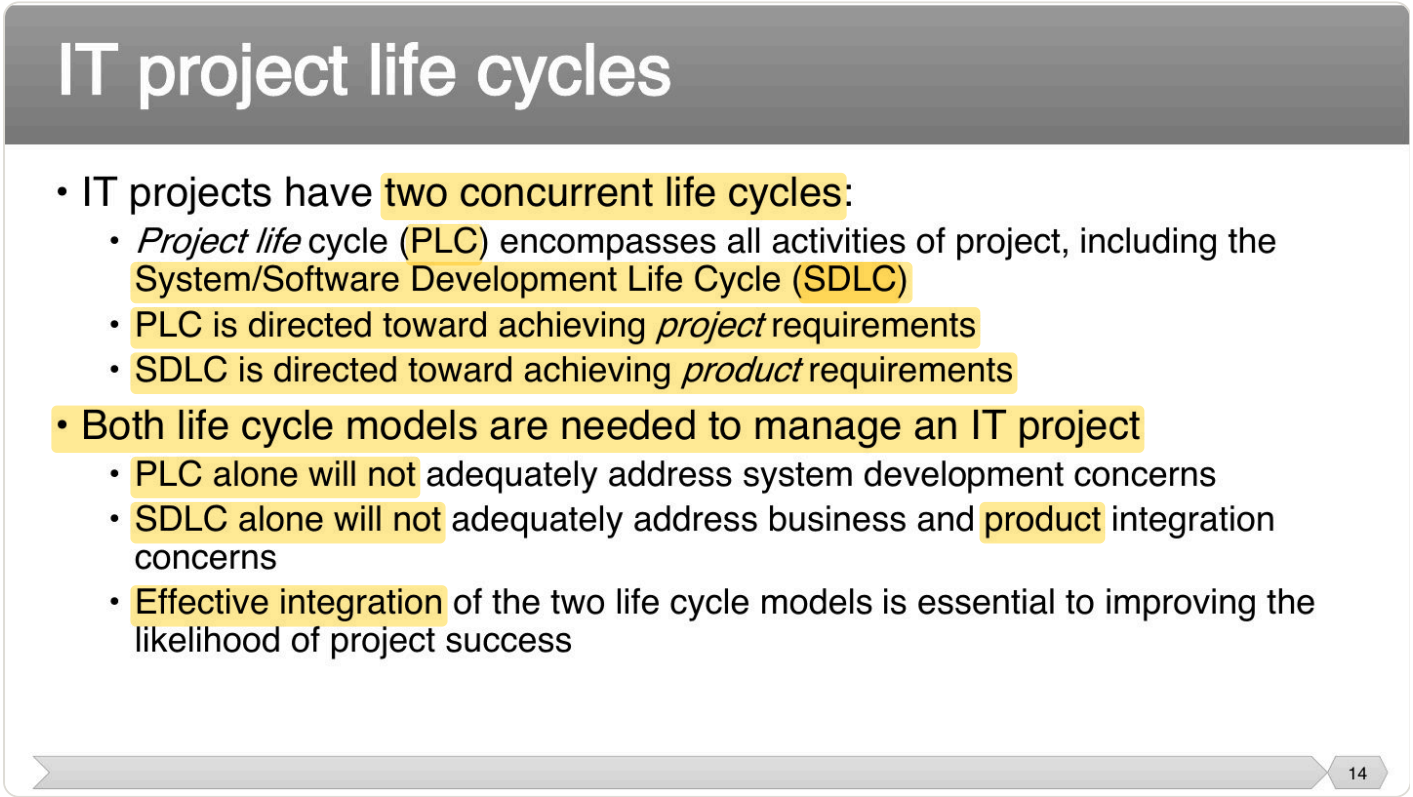
C Skill

> That's what a person needs.

D Milestone

> That's an event marking progress.

SLIDE 14 / 98 **IT project life cycles**



NOTES

1 A software project is more than coding: someone also handles contracts, budgets and approvals. So the **PLC** contains the development work, the **SDLC**, plus everything around it.

2. The **PLC** targets **project** requirements (scope, time, cost, approvals); the **SDLC** targets **product** requirements (features, quality). A common trap: don't swap them.

3. A perfect product can still fail if nobody arranged the budget, and a well-run project can still build broken software. So you need both, because **neither is enough alone**.

4. The two must line up. A PM who plans a release date without knowing how development works will set dates that can't be met, which is why **integrating** them matters.

EXAM ANGLE

Prediction Q1

The SDLC is directed toward achieving:

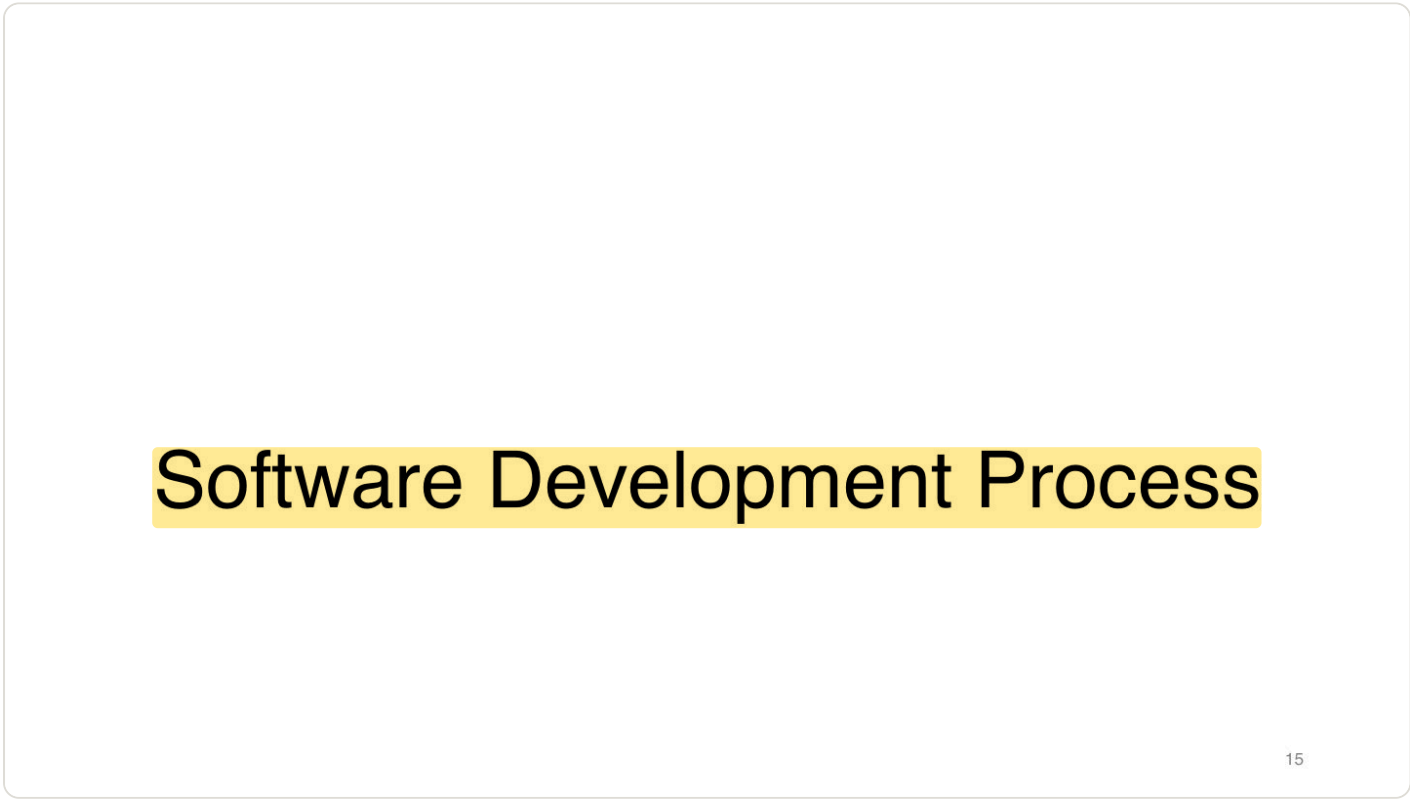
A Procurement

B Project requirements

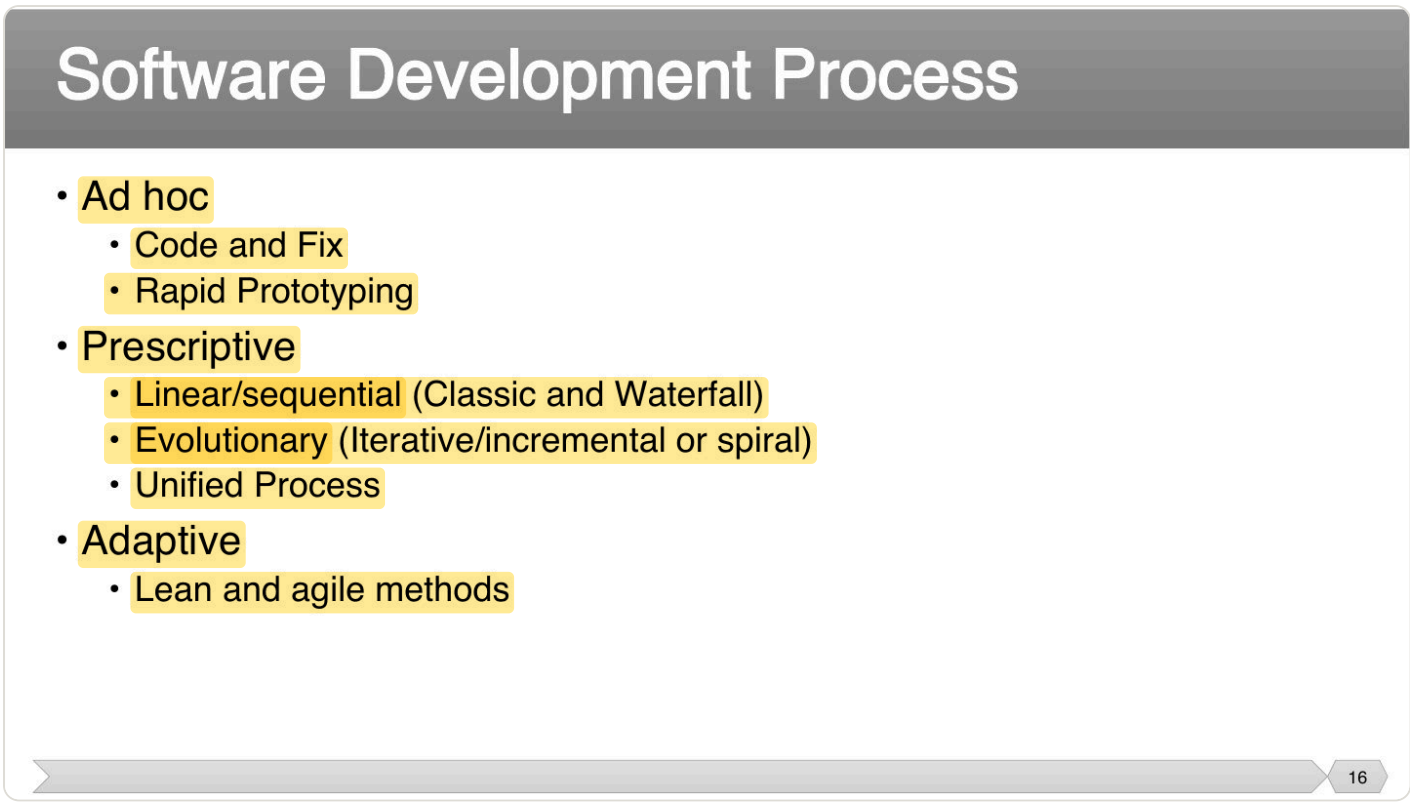
C **Product requirements**

D Business case approval

SLIDE 15 / 98 **Slide 15**



 Software Development Process



NOTES

1. "We just start coding and fix things" is how many small projects begin. **Ad hoc** can work for tiny tools, but it falls apart as projects grow, since nothing is defined.
2. These set out in advance what to do and when. **Prescriptive** doesn't have to mean waterfall: evolutionary approaches are prescriptive too, and it's the defined process that matters.
3. Rather than a detailed plan up front, these expect to change course. **Adaptive** methods like lean and agile treat what you learn as the main input for planning.

EXAM ANGLE

Prediction Q1

Code and fix belongs to which family?

A Ad hoc

It has no defined process.

B Prescriptive

That has a defined process.

C Unified Process

That’s prescriptive.

D Adaptive

Adaptive covers lean and agile.

Prediction Q2

Spiral belongs to which family?

A Neither

It’s listed.

B Ad hoc

Spiral is a defined process.

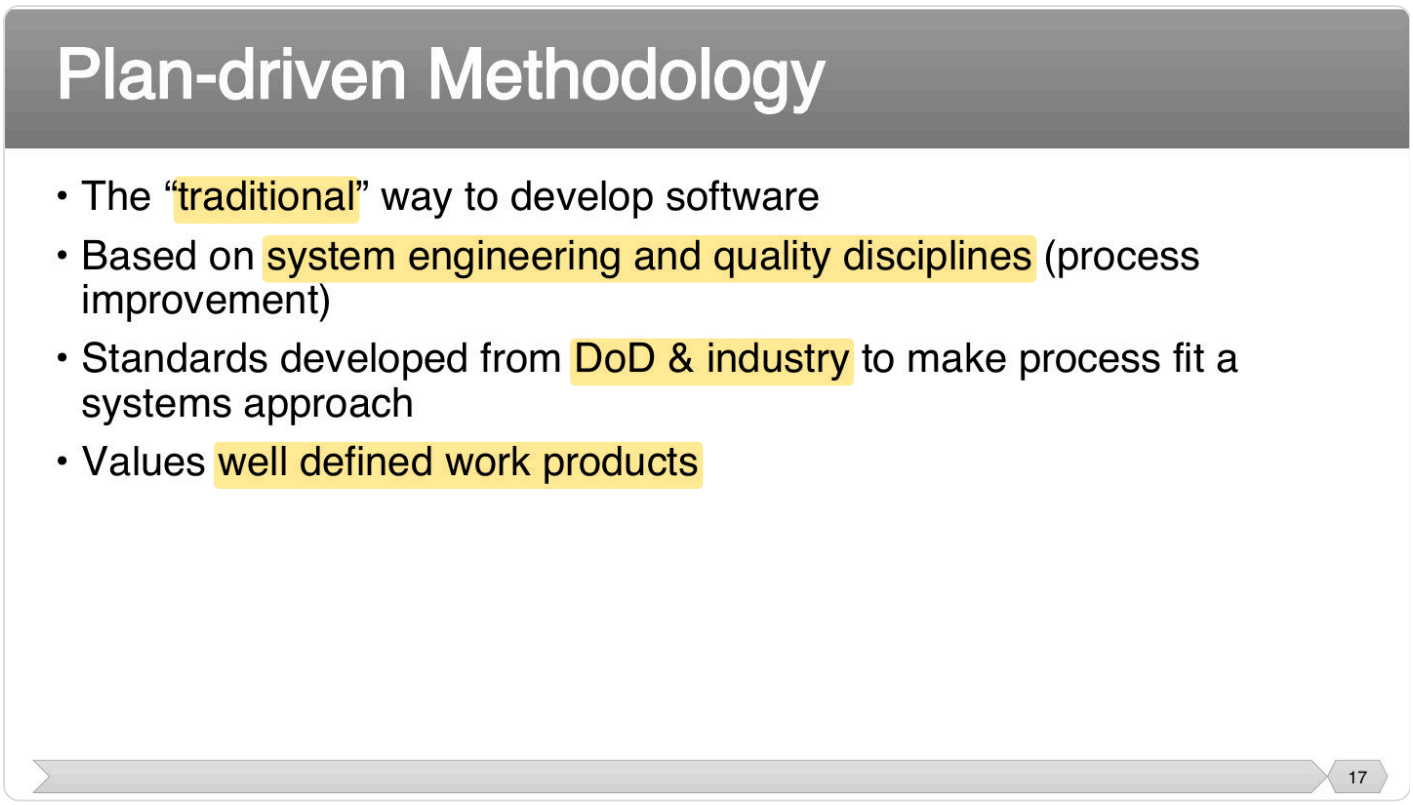
C Adaptive

Adaptive is lean and agile here.

D Prescriptive, as an evolutionary process

That’s where the slide places it.

 Plan-driven Methodology



NOTES

1. Traditional doesn’t mean bad. **Plan-driven** comes from fields where you plan carefully because re-work is very expensive...
2. ...such as building aircraft. It borrows from those fields’ way of thinking, which is why it’s **rigorous**: define it, measure it and improve it...
3. ...and when you’re a big customer, like the US **Department of Defense**, you want every contractor working the same way, so you can compare and control them...
4. ...which is why it **values documents**. They let different teams, even different companies, hand work to each other, and they leave a record to check.

EXAM ANGLE

Prediction Q1

What does plan-driven development value?

A Avoiding all documents

> That's the opposite.

B Well-defined work products

> Documents and specifications let teams hand work on.

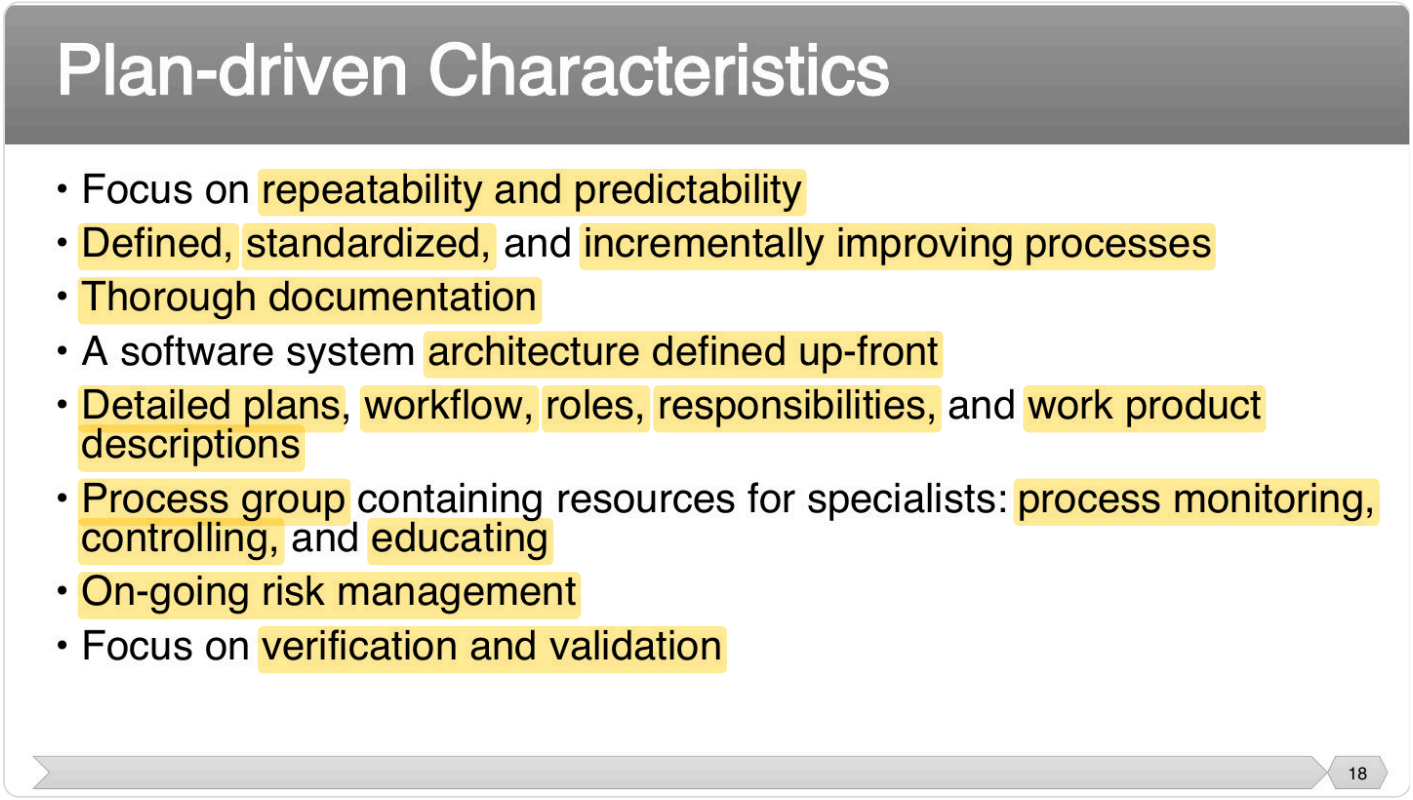
C No standards

> It's standards-based.

D Working without any plan

> That's the opposite.

SLIDE 18 / 98 **Plan-driven Characteristics**



NOTES

1 If you run the same process twice, you want similar results. That's what **repeatability** is for: being able to predict cost and date, which is exactly what big contracts demand.

2. Writing everything down lets others take over and lets you check against it. An **architecture up front** avoids expensive redesign, provided requirements don't change much.

3. Everyone knows exactly what's expected, which is valuable on very large teams, though it's heavy for small ones. A **process group** exists just to keep that machine running.

4. Because you've committed early, surprises are costly, so risk gets watched throughout. **Verification** asks "did we build it right?" and **validation** asks "did we build the right thing?".

EXAM ANGLE

Prediction Q1

What is the main aim of plan-driven processes?

A **Repeatability and predictability**

> That's what the first point says.

B Maximum improvisation

> That's the opposite.

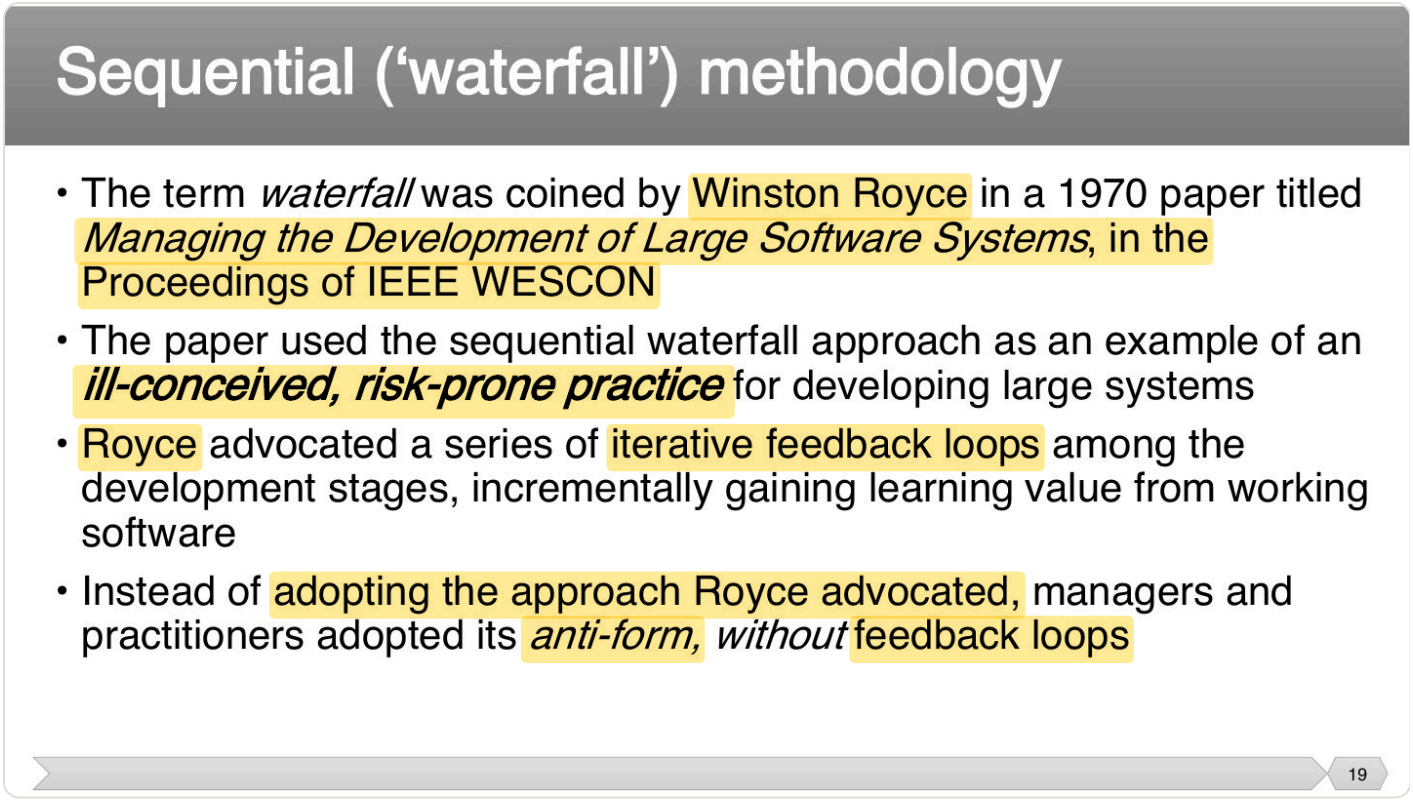
C Skipping verification

> It focuses on verification and validation.

D Avoiding risk management

> Risk management is ongoing.

 Sequential (‘waterfall’) methodology



NOTES

1. It’s an irony worth remembering: the name most people use comes from a paper that warned against it. **Royce** is the only historical name you need here.
2. He didn’t think testing only at the end was sensible for big systems. By then, fixing a design flaw is very expensive. So he called the pure version **risk-prone**.
3. His idea was to go back and improve earlier stages using what you learn. So **feedback loops** are the part of his paper people forgot.
4. Industry adopted its **anti-form**, the version without feedback loops. A common trap: “Royce recommended the pure waterfall” is false.

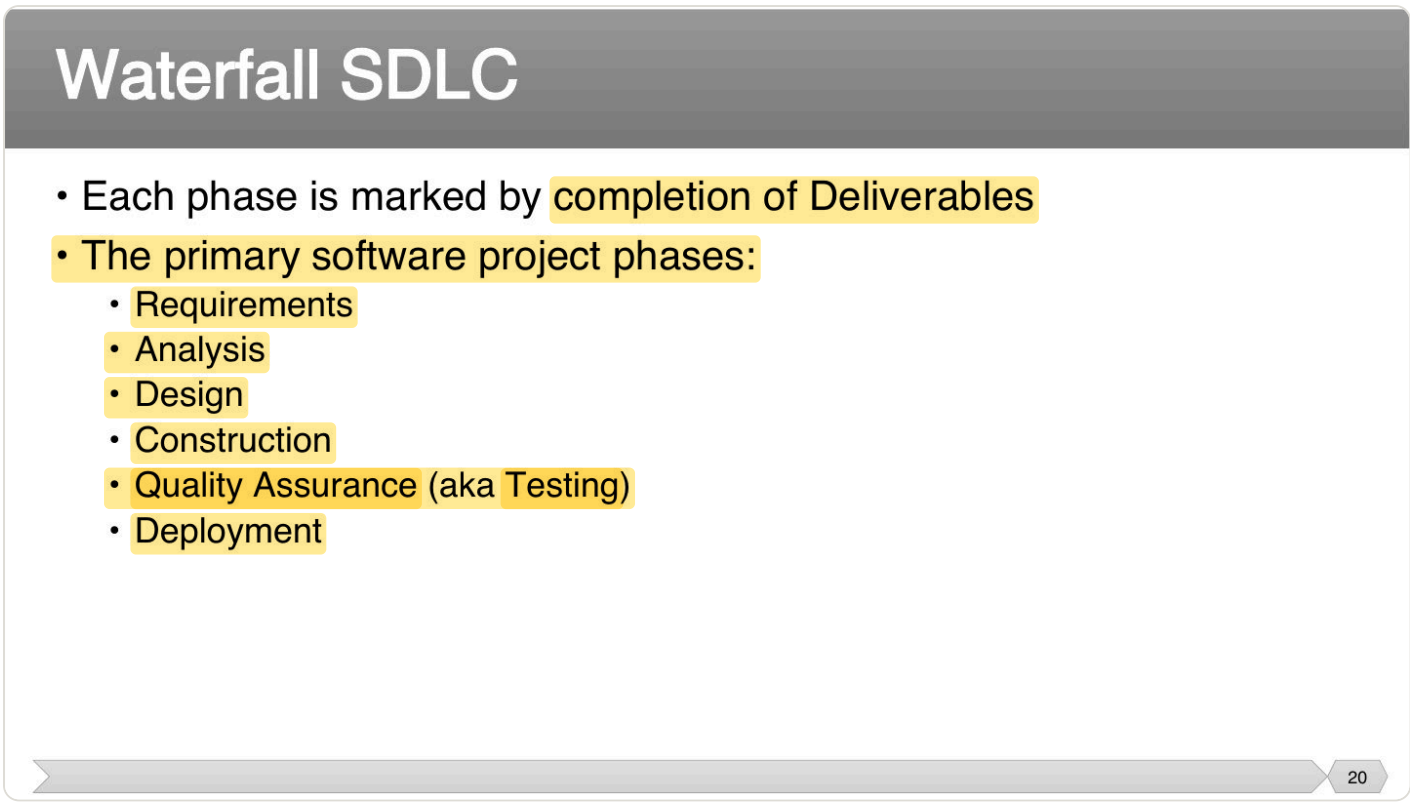
EXAM ANGLE

Prediction Q1

Winston Royce’s 1970 paper presented the pure sequential waterfall as an ill-conceived, risk-prone practice.

- T **True**
- F False

 Waterfall SDLC



NOTES

1. The idea is that you don't start design until requirements are signed off. Finished **deliverables** keep control tight, but they also make later changes expensive.
2. Each stage depends on the last: you can't design what hasn't been analyzed. Mind the order, and that **testing** comes near the end: that's the main weakness.

TIPS & TRICKS

- The order matters here: **R**eal **A**rchitects **D**raw, **C**onstruct, **Q**uality-check, **D**eploy. Requirements, Analysis, Design, Construction, Quality assurance, Deployment.

EXAM ANGLE

Prediction Q1

What marks the end of each waterfall phase?

A A customer demo every two weeks

That's iterative.

B **Completion of deliverables**

The next phase starts after that.

C A timebox ending

That's adaptive.

D A daily stand-up

That's Scrum.

Prediction Q2

Which phase comes right after Design in the slide's waterfall?

A Quality assurance

That follows construction.

B **Construction**

The order is requirements, analysis, design, construction.

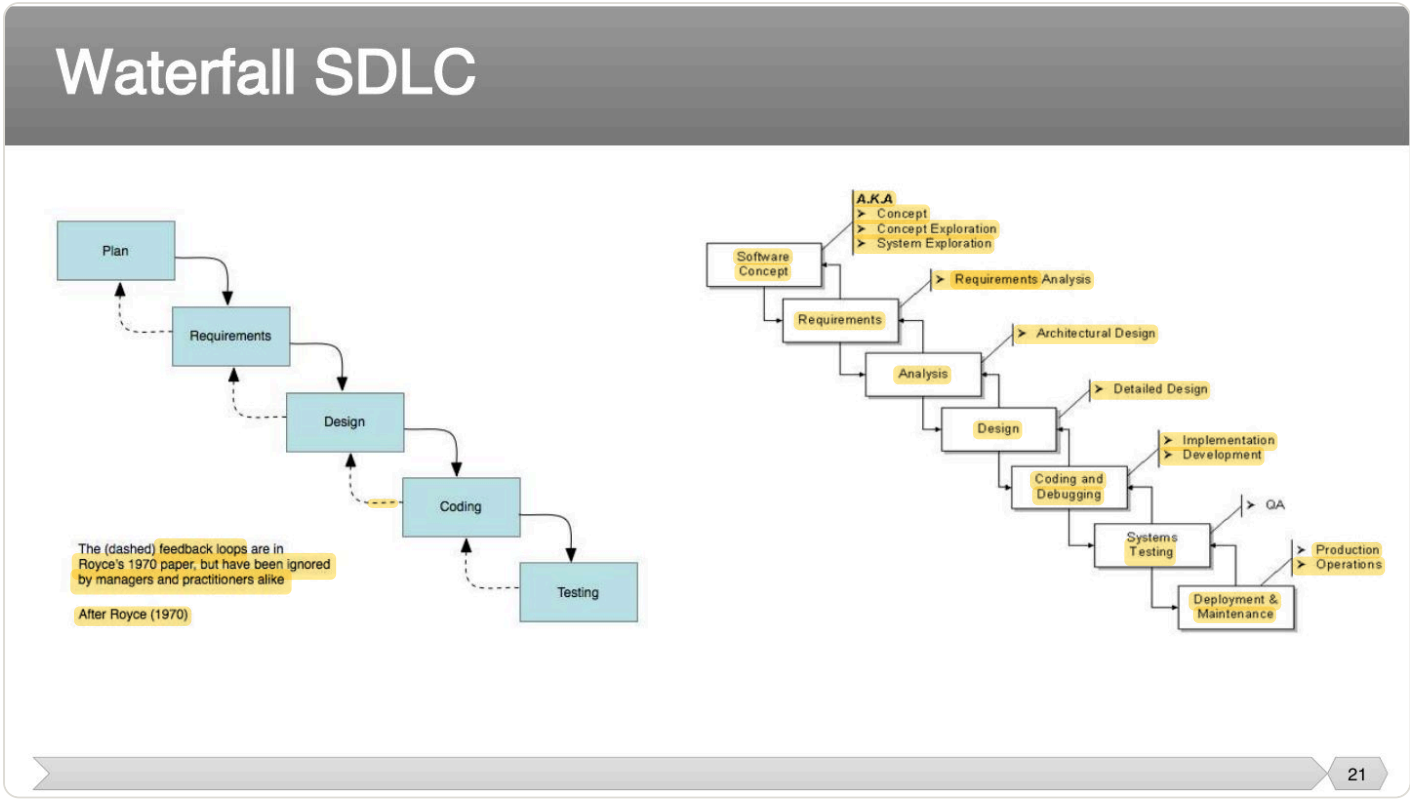
C Requirements

That comes first.

D Deployment

That comes last.

 Waterfall SDLC



NOTES

1. Royce's waterfall: plan → **requirements** → design → coding → testing, with dashed **feedback loops** back to the previous step. The note on the left says those loops were in Royce's paper but have been ignored by managers and practitioners alike.

2. On the right, a more detailed version (concept → requirements → architecture → detailed design → coding → **testing** → deployment). Either way, testing comes last, which is why problems are found late.

EXAM ANGLE

Prediction Q1

What did Royce's diagram include that people came to ignore?

A Feedback loops

> The note says so.

B Testing

> That's still used.

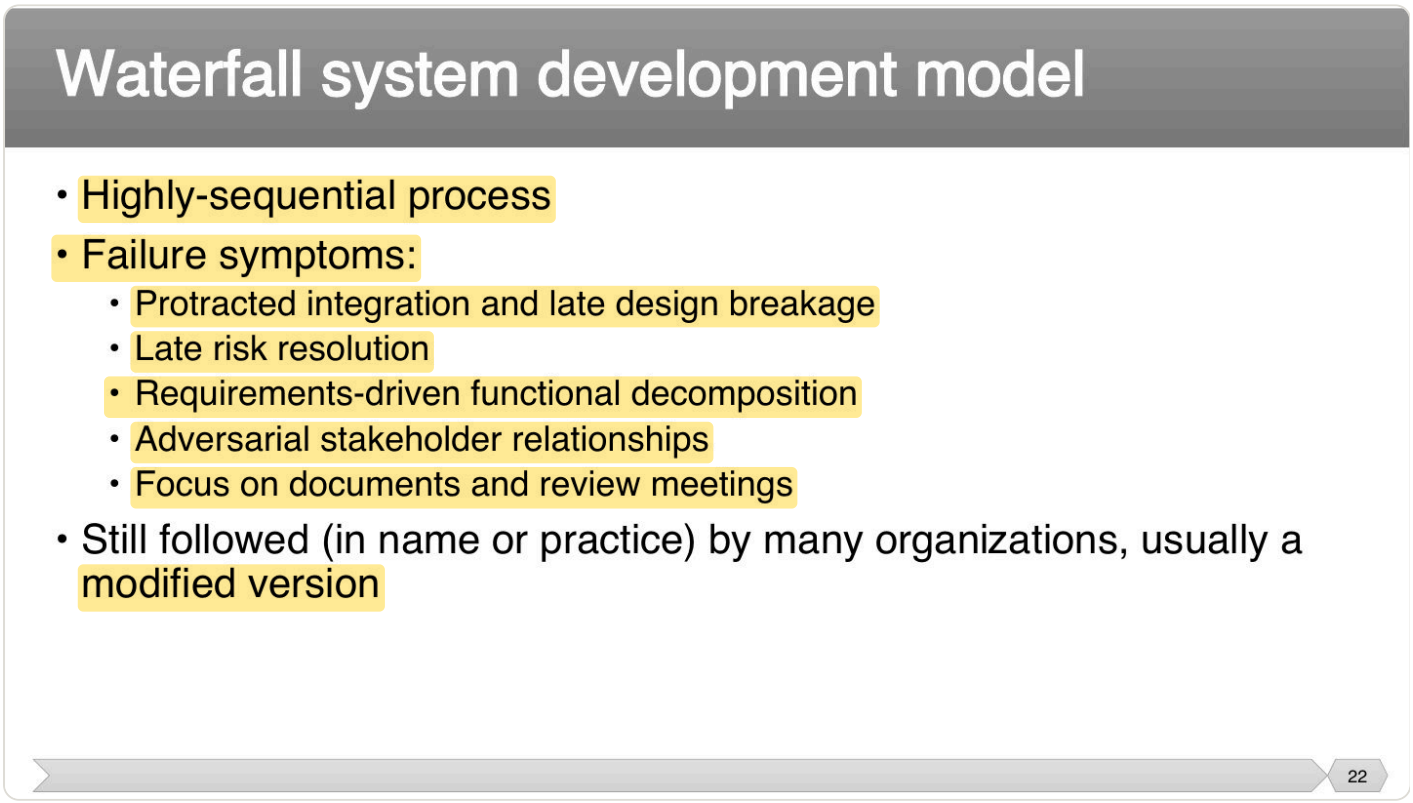
C Coding

> That's still used.

D Requirements

> Those are still used.

SLIDE 22 / 98 **Waterfall system development model**



NOTES

1 Nothing is combined until late, so the first real proof the parts fit comes at the end. That's when design flaws appear and are most expensive, hence **late design breakage** and **late risk resolution**.

2. **Adversarial stakeholder relationships** (arguing over signed documents) and a **focus on documents and reviews** instead of working software.

3. Why does it survive? It's easy to explain and to contract for, and some projects suit it. Most organizations use a **modified version**, adding some iteration, rather than the pure form.

EXAM ANGLE

Prediction Q1

Why does waterfall suffer from late design breakage?

A It integrates continuously

> That's the opposite.

B It only uses agile tools

> That has nothing to do with it.

C It has no design phase

> It has one.

D **Everything is integrated late, when design flaws first appear**

> That's the symptom.

Prediction Q2

Why might stakeholders become adversarial in waterfall?

A They have no documents

> There are plenty of documents.

B **They end up arguing over signed documents**

> Relationships center on what was promised on paper.

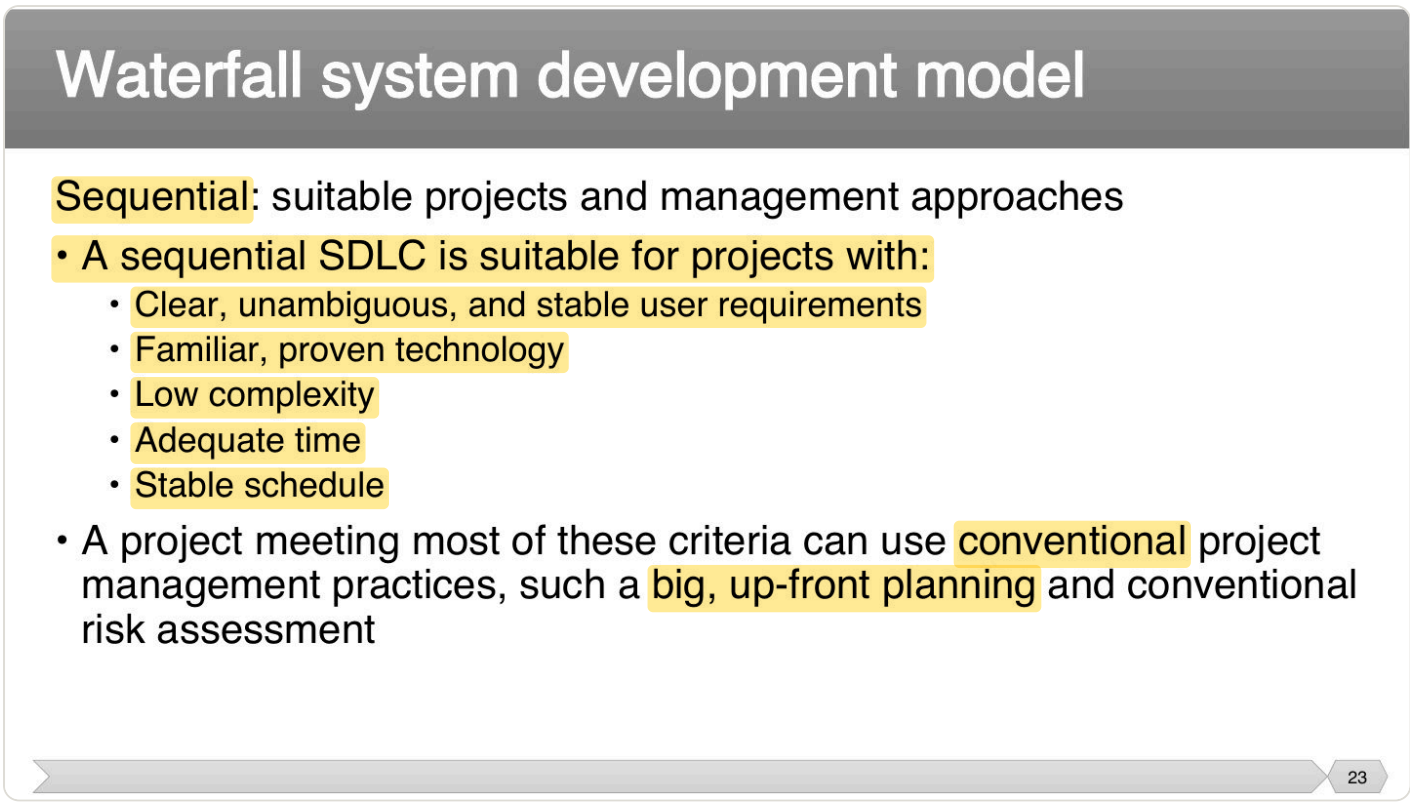
C They see working software each week

> That builds trust.

D They're involved in all iterations

> That's agile.

 Waterfall system development model



NOTES

1. Every item is a form of "we already know what we're doing". If requirements and technology are both familiar, there's little to discover, so a fixed plan works. If one is missing, **sequential** gets riskier.
2. When there's little to learn along the way, planning it all at the start is efficient. **Conventional** methods are the right call here, so don't assume agile is always better.

TIPS & TRICKS

- **FALConS**: Familiar technology, Adequate time, Low complexity, Clear requirements, Stable schedule.

EXAM ANGLE

Past exam · Midterm 251 · V1 · Case Q2 (also V2) Q1

Given the evolving requirements (student usability, advisor requests, IT integration rules), which development approach (Agile, Waterfall, or Hybrid) would you recommend, and why would it be the best fit for this project?

Answer: Agile (Scrum) (0.5), because it handles the evolving requirements from students, advisors and IT, delivers iteratively so each sprint gets feedback, and integrates with DevOps pipelines for frequent tested releases (0.5). Justify it against the slide criteria: the requirements are reasonably clear but not stable, so waterfall's "clear, unambiguous, stable requirements" condition fails.

Prediction Q2

Which project suits a sequential (waterfall) SDLC?

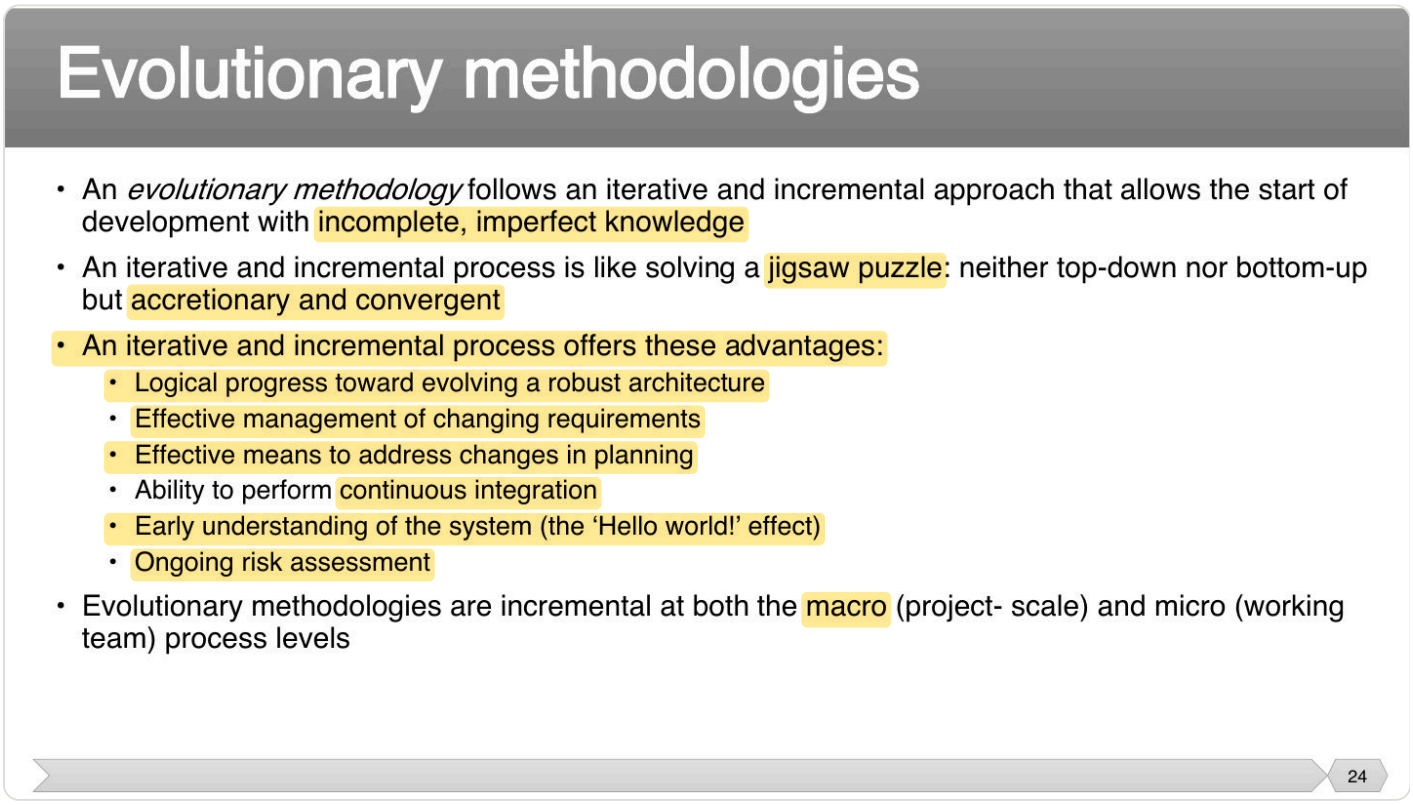
A Unclear requirements and new technology

B High complexity with a very short schedule

C **Stable requirements, familiar technology, low complexity**

D Requirements expected to change weekly

SLIDE 24 / 98 **Evolutionary methodologies**



NOTES

1. Waterfall needs you to know a lot before starting. **Evolutionary** approaches accept you don't, so you begin with what you know and learn the rest by building.
2. It is like a **jigsaw puzzle**: neither top-down nor bottom-up, but **accretionary and convergent** (pieces accumulate until the picture emerges).
3. Each benefit comes from working in small pieces. Because you integrate often, you spot problems fast, and early working results, even a "Hello world!", show the team that things connect. That's **continuous integration** in practice.
4. The same pattern works at two scales. The whole project grows in stages (**macro**), and within each stage, teams build small parts bit by bit (micro).

EXAM ANGLE

Prediction

Q1

Why is iterative development compared to a jigsaw puzzle?

A **The picture emerges as pieces accumulate**

> It's accretionary and convergent.

B It's all luck

> It's organized.

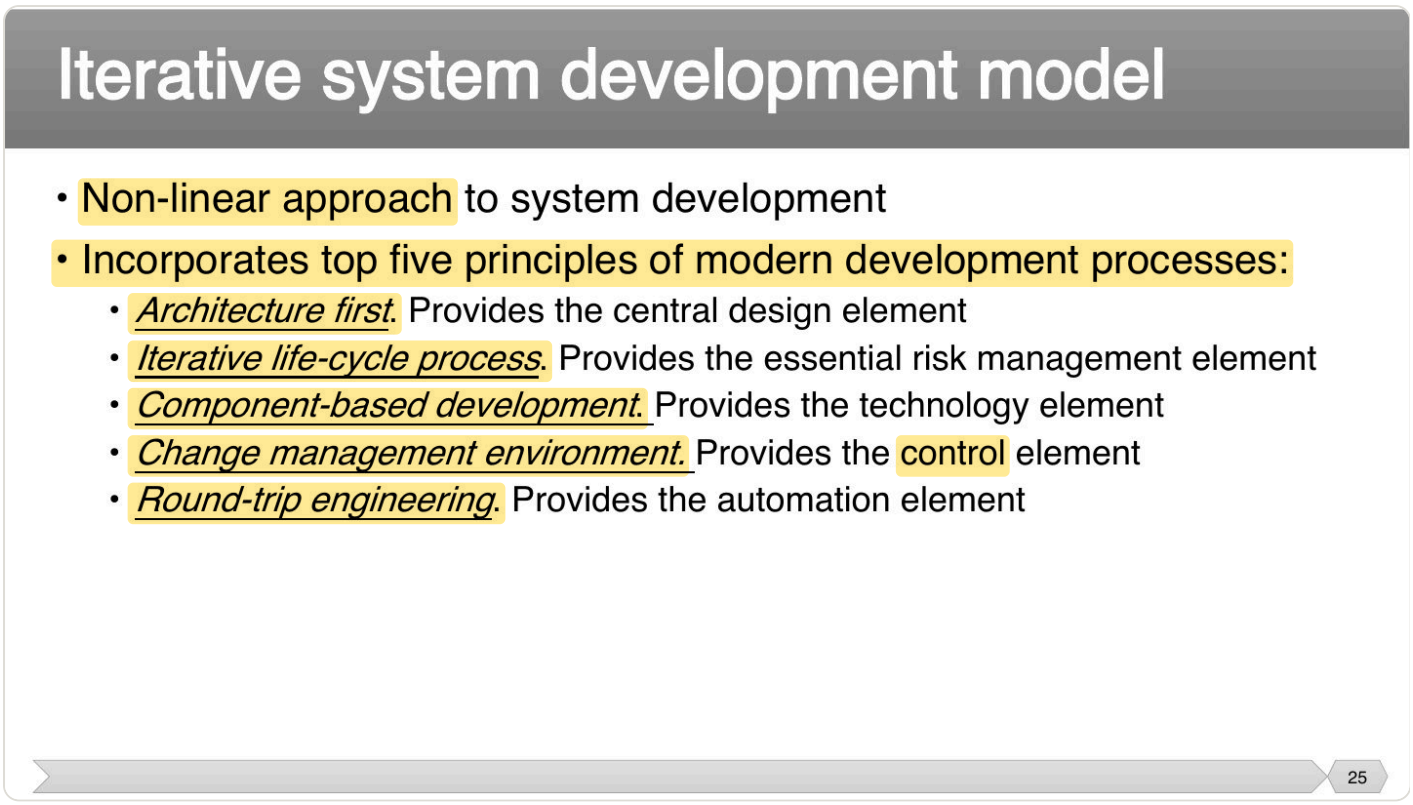
C It never finishes

> It converges.

D You must plan everything top-down

> It's neither top-down nor bottom-up.

 Iterative system development model



NOTES

1. You don't pass through each stage once and leave it behind. **Non-linear** means you can return to earlier work as you learn, which fixes waterfall's main weakness.
2. Why **architecture first**? It's the riskiest and hardest thing to change later, so get it stable early. And working in iterations surfaces **risks** sooner, which is why it's called the risk-management element.
3. Reusing components saves building everything from scratch. If changes happen often, you need a system to **control** them, and automation keeps models and code matching, so nobody works from outdated designs.

TIPS & TRICKS

- **CIRCA**: Component-based development, Iterative life cycle, Round-trip engineering, Change management environment, Architecture first.

EXAM ANGLE

Prediction Q1

Which principle provides the essential risk-management element?

A Architecture first

> That’s the central design element.

B **Iterative life-cycle process**

> That’s what the slide says.

C Component-based development

> That’s the technology element.

D Round-trip engineering

> That’s automation.

Prediction Q2

Which principle provides the control element?

A Architecture first

> That’s the central design element.

B **Change management environment**

> That’s what the slide says.

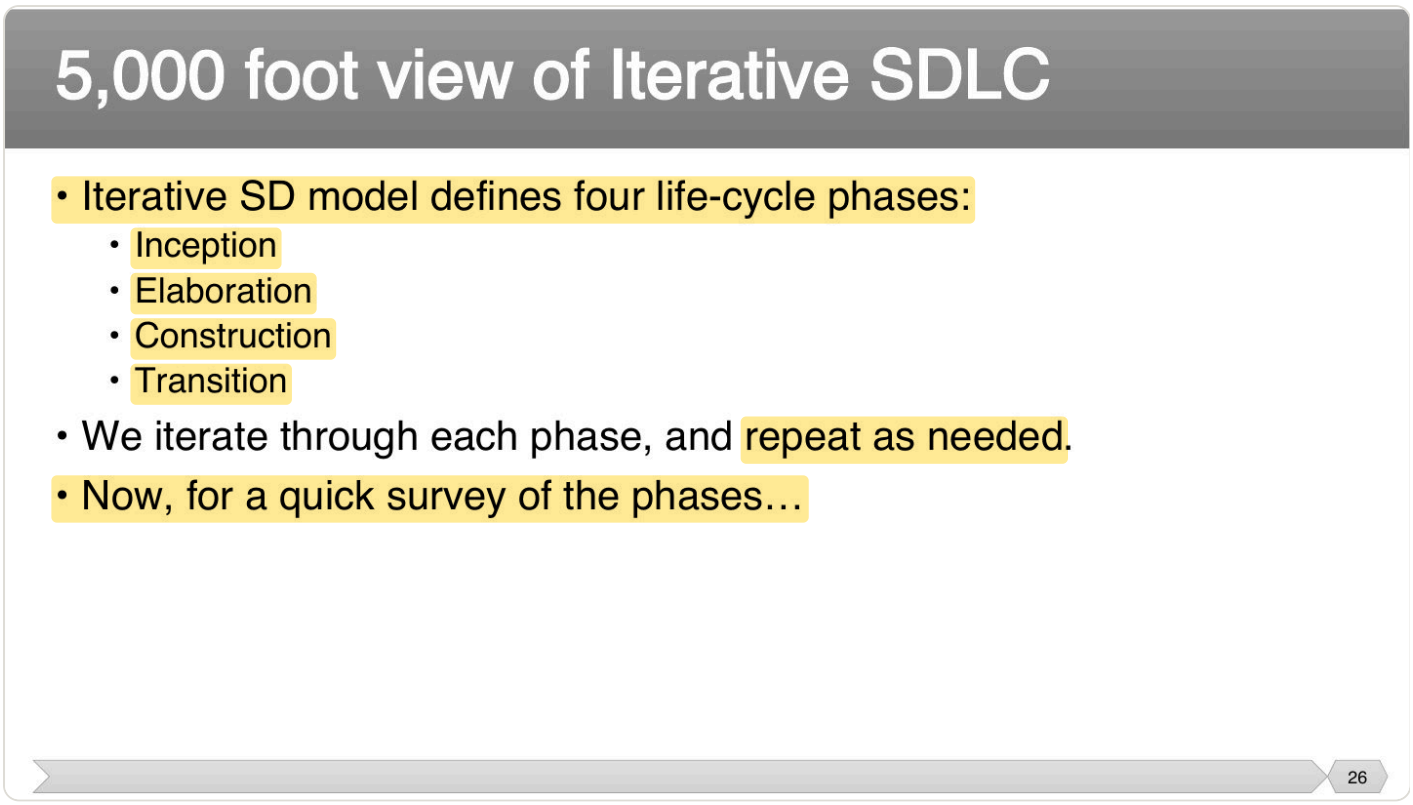
C Round-trip engineering

> That’s automation.

D Iterative life-cycle process

> That’s risk management.

 5,000 foot view of Iterative SDLC



NOTES

1. Each phase answers a different question. Should we build it? Can we build it? Build it. Can they use it? That's a good way to recall **inception, elaboration, construction, transition**.
2. A phase isn't one pass, because you may need several rounds before the risk is low enough to move on. That's the point of **repeated as needed**.

TIPS & TRICKS

- **InfECT**, already in order: Inception, Elaboration, Construction, Transition.

EXAM ANGLE

Prediction Q1

Which is the correct order of the iterative SDLC phases?

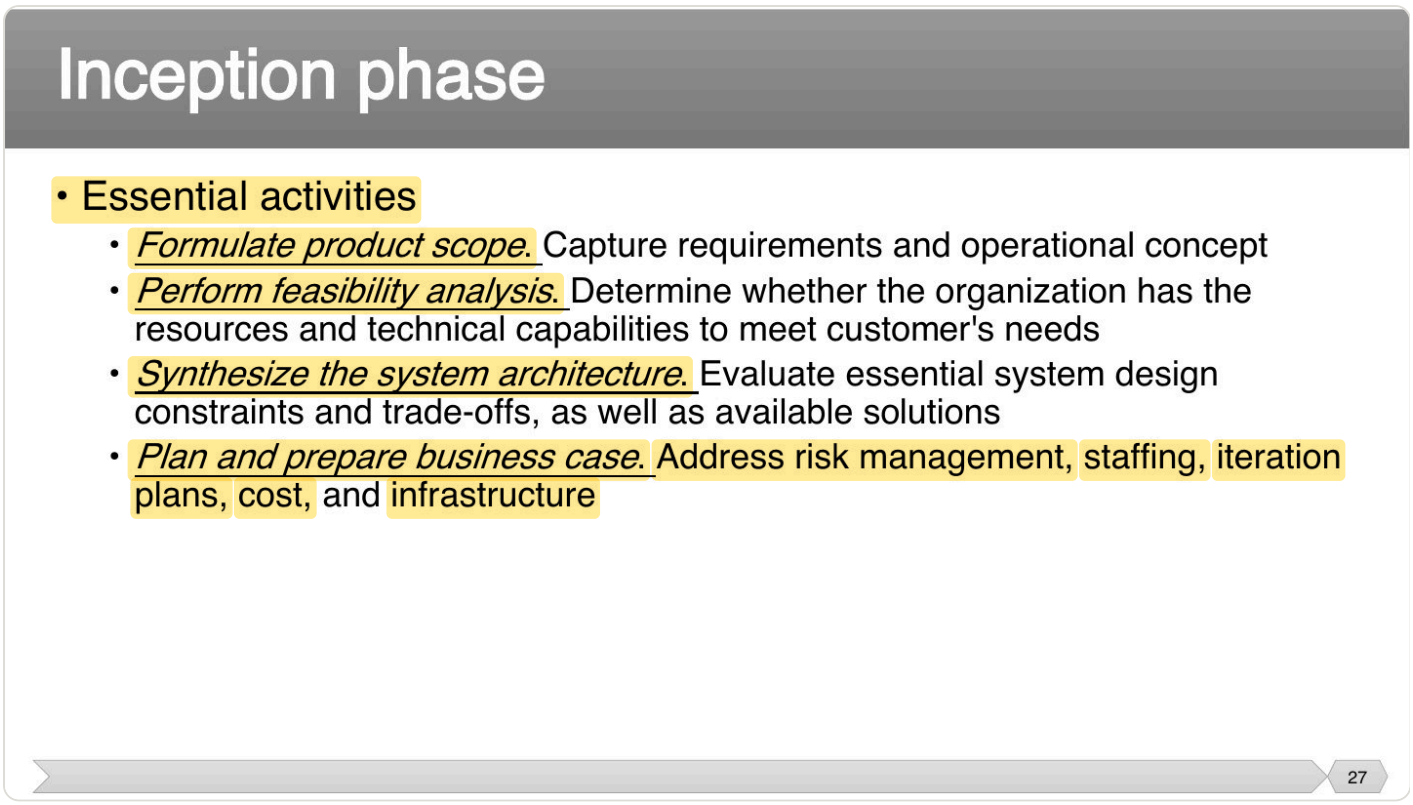
A Inception, construction, elaboration, transition

B Planning, design, build, deploy

C Elaboration, inception, construction, transition

D **Inception, elaboration, construction, transition**

 Inception phase



NOTES

1. Before spending much, work out what you're actually building and for whom. **Product scope** at this stage is a rough outline, not every detail...
2. ...and check it's possible with what you have. **Feasibility** can stop a bad project at the cheapest moment...
3. ...and sketch how it could work. You're not designing it all yet, only checking that one workable **architecture** exists...
4. ...and finally justify it. The **business case** is what decision-makers use to agree whether to go on.

EXAM ANGLE

Prediction Q1

Whether the organization has the resources and technical capability is checked in:

A Closing

> That's not one of the four phases.

B Transition

> That's deployment.

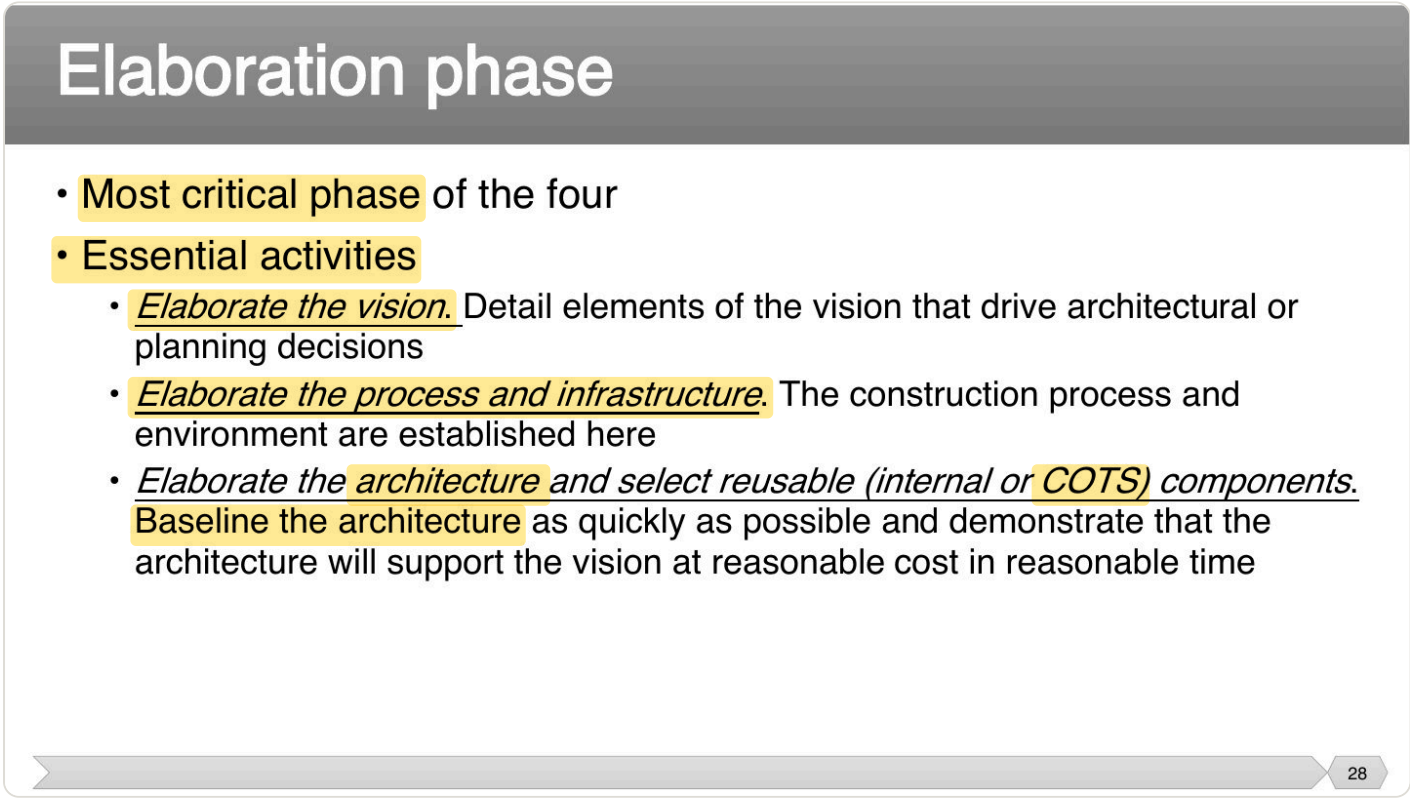
C Construction

> That's building.

D Inception, by feasibility analysis

> That's an inception activity.

SLIDE 28 / 98 **Elaboration phase**



NOTES

1 **Elaboration** is the **most critical** of the four phases: it removes the biggest risks before heavy spending starts.

2. You can't detail everything yet, so focus on what shapes big decisions. If the vision says "millions of users", that detail changes the **architecture**, and the color of a button doesn't.

3. Everything construction needs is set up now: tools, environments and ways of working. Getting the **infrastructure** right here avoids slowing down a much larger team later.

4. To **baseline** means to fix it as the reference, and only after showing that it works. Reusing **COTS** (off-the-shelf) parts saves effort, as long as they fit the architecture.

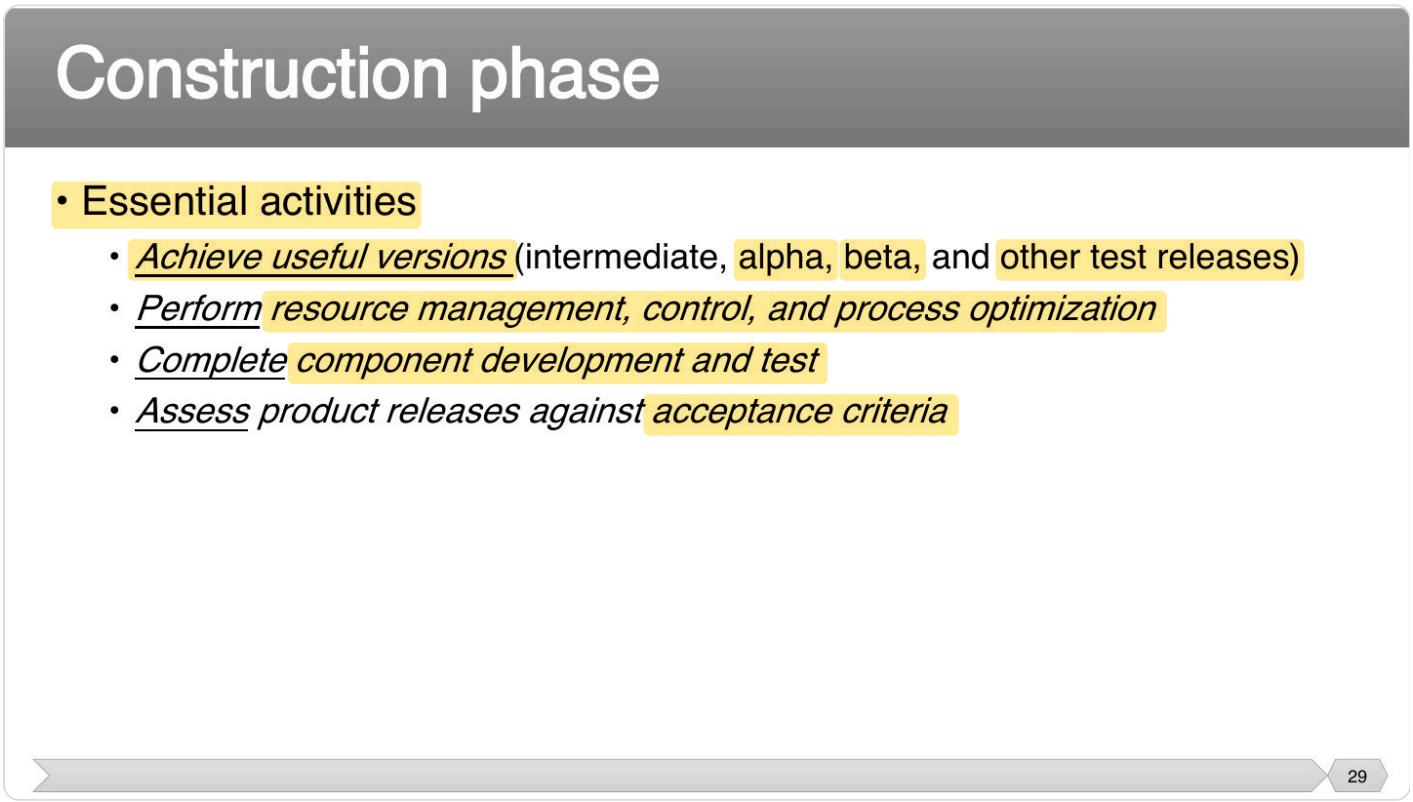
EXAM ANGLE

Prediction Q1

Which iterative phase is described as the most critical?

- **A** **Elaboration**
- **B** Transition
- **C** Construction
- **D** Inception

SLIDE 29 / 98 **Construction phase**



NOTES

1. Unlike waterfall, you don't wait until the end to see something. Each **useful version** can be tried, and an alpha or beta gets real feedback early...

2. ...this is when most people and money are in use. So the PM's work shifts to **resource management** and keeping the flow efficient...
3. ...it's also the phase where the bulk of **development and testing** gets done, with the hard design risks already handled...
4. ...and "done" isn't just an opinion. Each release gets checked against **acceptance criteria**, agreed in advance, so that everybody knows when it's good enough.

EXAM ANGLE

Prediction Q1

Alpha and beta releases are part of which phase?

A Inception

> That sets scope.

B Analysis

> That isn't one of the four.

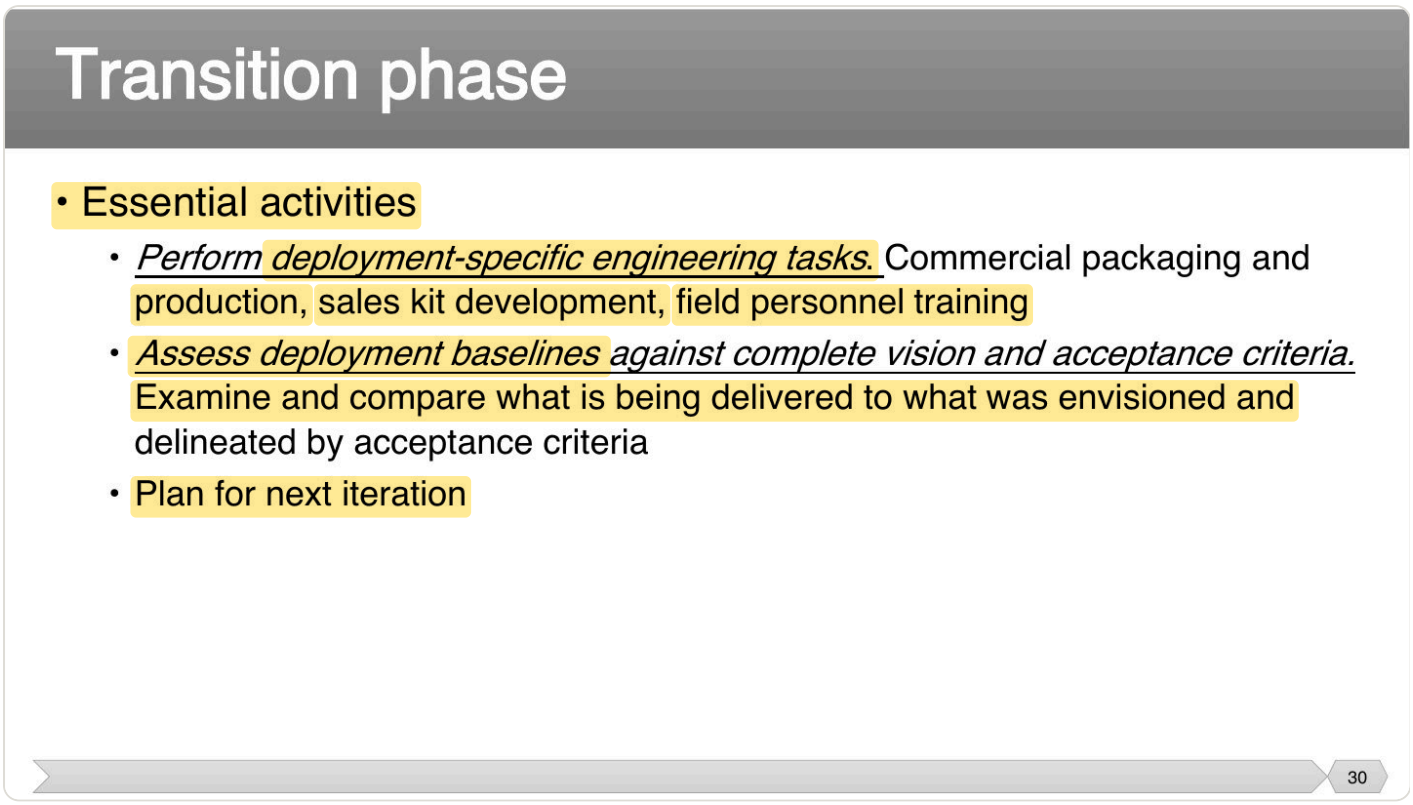
C **Construction**

> It produces useful versions.

D Elaboration

> That baselines the architecture.

 Transition phase



NOTES

1. Working software isn't yet a usable product for customers. Someone still has to package it, and train users and support staff, which is what **deployment-specific** tasks mean.
2. This is the final reality check: did we get what we set out to get? **Comparing** the result with the original vision reveals gaps before you call it finished.
3. Most products carry on after a release, so the last phase feeds into the next. **Planning the next iteration** is why the model is drawn as a repeating cycle.

EXAM ANGLE

Prediction Q1

Training field staff and preparing sales kits belong to:

A Elaboration

> That's architecture.

B Inception

> That sets scope.

C Requirements

> That isn't one of the four.

D Transition

> These are deployment-specific tasks.

SLIDE 31 / 98 **Comparative expenditure profiles**

Comparative expenditure profiles

Waterfall		Iterative	
Activity	Cost	Cost	Activity
Management	5%	10%	Management
Requirements	5%	10%	Requirements
Design	10%	15%	Design
Code & Unit Testing	30%	25%	Implementation
Integration & Test	40%	25%	Assessment
Deployment	5%	5%	Deployment
Environment	5%	10%	Environment
Total	100%	100%	Total

31

NOTES

1 **Waterfall** spends most of its budget late: 30% on code and unit test and **40% on integration and test**, because problems surface only when everything comes together.

2 It costs more up front, as more is spent on **management** and design. But you pay less for late surprises, since testing and integration are spread throughout. Iterative moves the spending earlier, when mistakes are cheaper.

EXAM ANGLE

Prediction Q1

In the waterfall expenditure profile, which activity takes the largest share of cost?

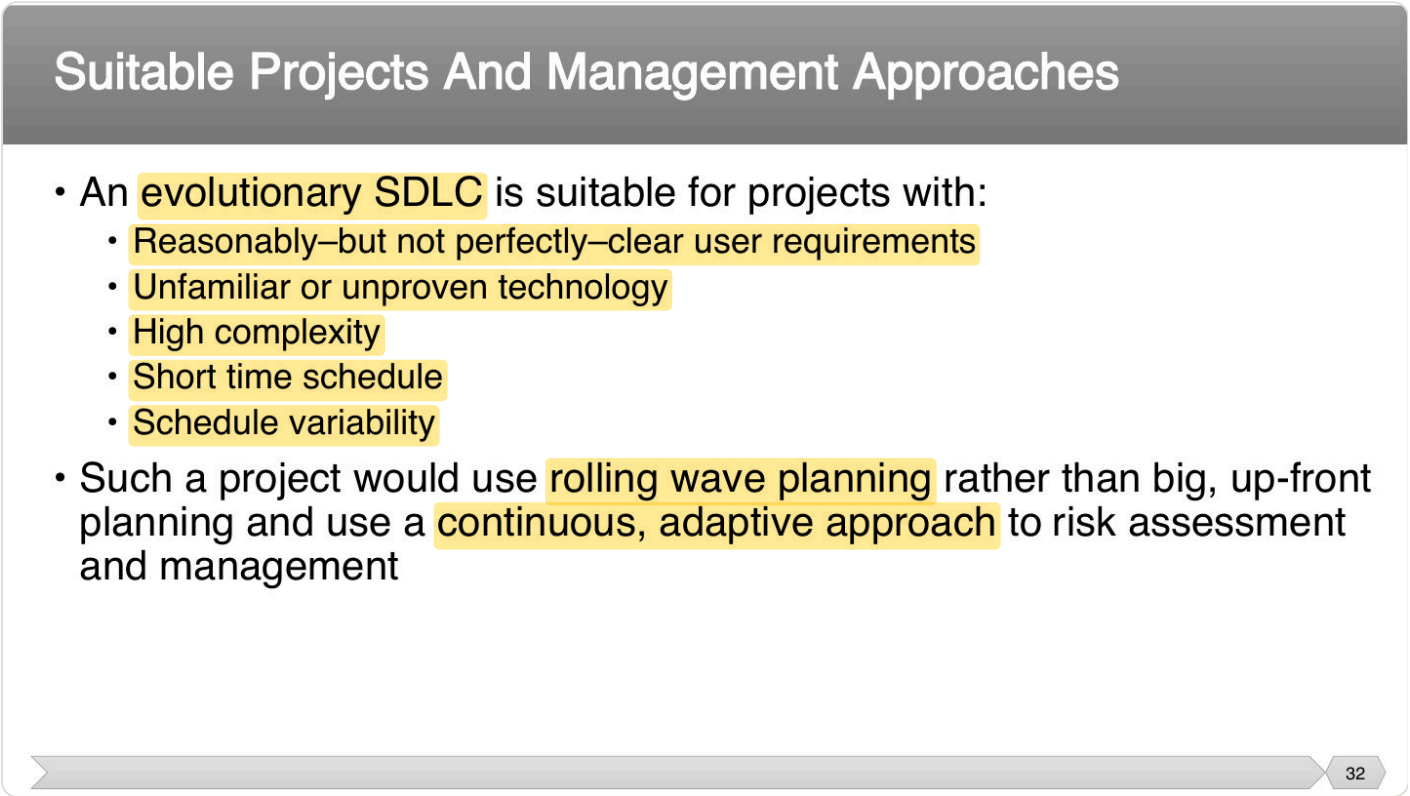
A **Integration and test (40%)**

B Requirements (5%)

C Design (10%)

D Management (5%)

SLIDE 32 / 98 **Suitable Projects And Management Approaches**



NOTES

1 Compare with slide 23: everything flips. You don't fully know what's wanted or how to do it, and time is tight. So an **evolutionary SDLC** lets you learn and deliver as you go. Remember **RUSHeS**.

2 Such a project uses **rolling wave planning** (detailed plans for the near term, rough plans for later) instead of big up-front planning, and a **continuous, adaptive** approach to risk.

TIPS & TRICKS

- **RUSHeS**, the opposite of slide 23: Reasonably clear requirements, Unfamiliar technology, Short time, High complexity, Schedule variability.

EXAM ANGLE

Past exam · Midterm 251 · V1 · MCQ 6 (also V2) Q1

In which type of project would an iterative development approach be most beneficial?

A **A project where requirements are unclear and expected to evolve over time.**

> Iterative development refines ideas through feedback, which suits unclear or evolving requirements.

B A project with well-defined, unchanging requirements from the start.

C A project that must deliver the entire product only at final completion.

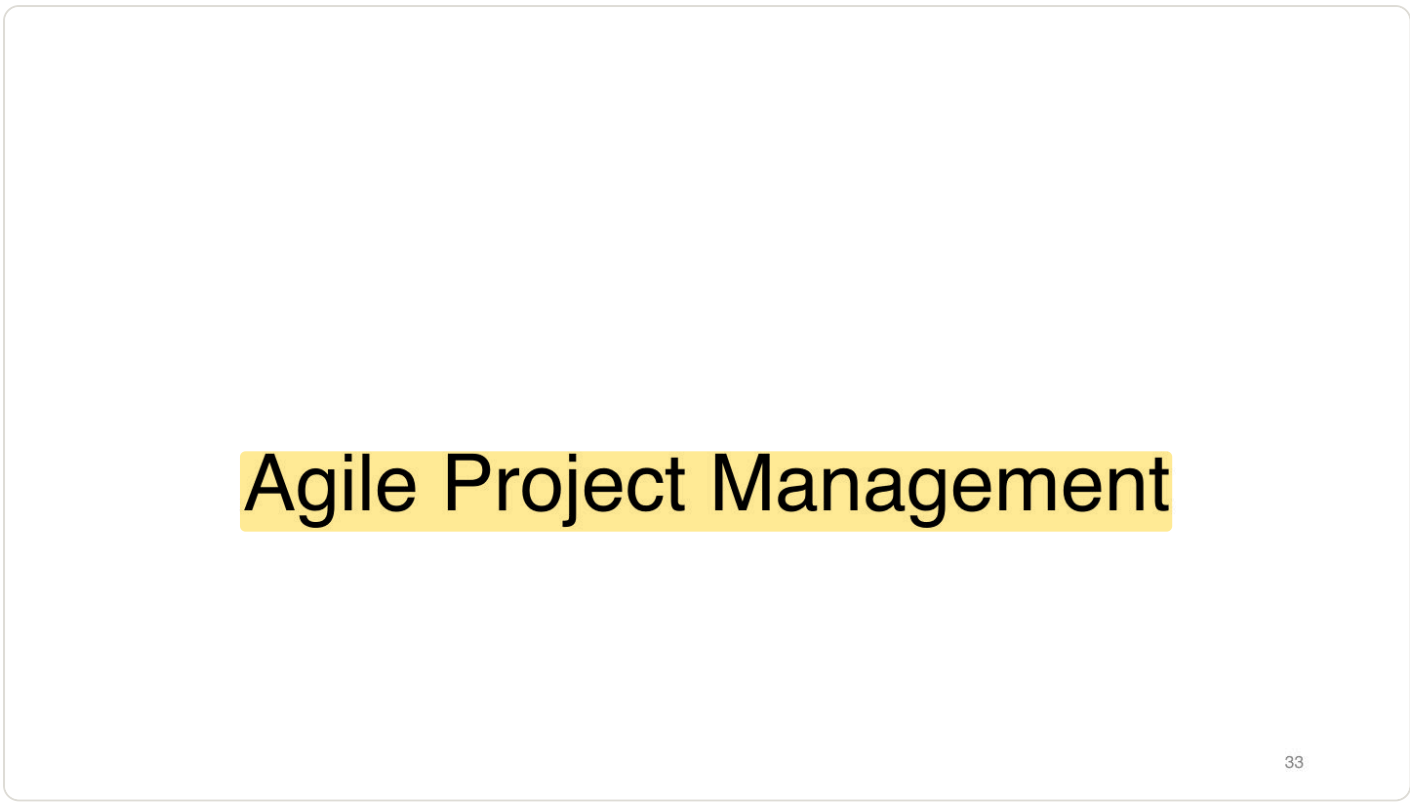
D A project that cannot accommodate stakeholder feedback during execution.

Past exam · Midterm 251 · V1 · Case Q2 (also V2) Q2

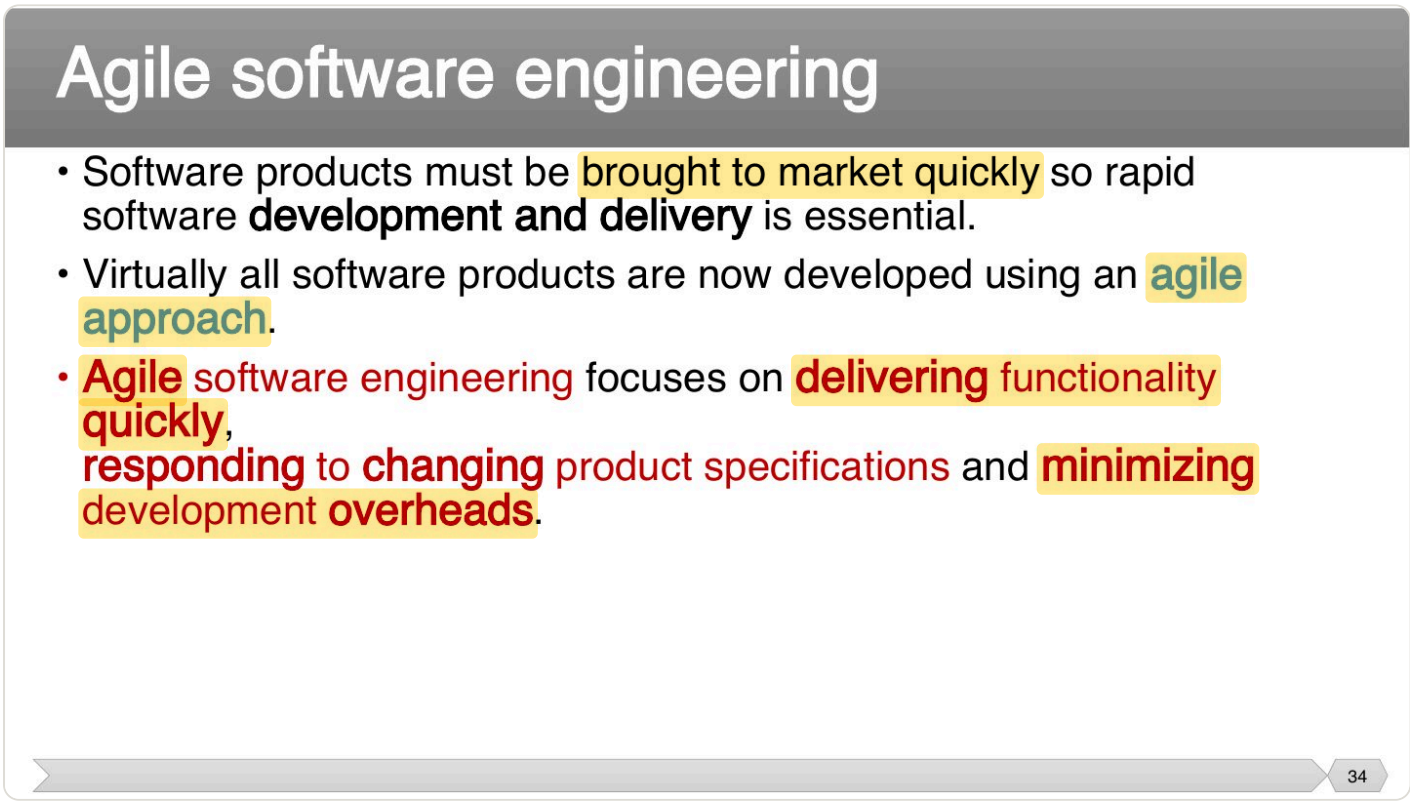
Given the evolving requirements (student usability, advisor requests, IT integration rules), which development approach (Agile, Waterfall, or Hybrid) would you recommend, and why would it be the best fit for this project?

Answer: Agile (Scrum) (0.5), because it handles the evolving requirements from students, advisors and IT, delivers iteratively so each sprint gets feedback, and integrates with DevOps pipelines for frequent tested releases (0.5). Justify it against the slide criteria: the requirements are reasonably clear but not stable, so waterfall's "clear, unambiguous, stable requirements" condition fails.

SLIDE 33 / 98 **Slide 33**



SLIDE 34 / 98 **Agile software engineering**



NOTES

1. Competitors may launch first, and a late product can miss its market. That's what makes speed so valuable, and why **quickly** comes up on this slide.
2. It's now the normal way to build most products. But "virtually all" is a broad claim: safety-critical systems may still favor heavier methods. Still, you should treat **agile** as the default for products.

3 Three goals with a common idea: avoid effort that doesn't reach the customer. Long documents that nobody reads are **overhead**, and they make change expensive.

EXAM ANGLE

Prediction Q1

What is agile software engineering focused on?

A **Delivering functionality quickly and minimizing overhead**

> The slide says so.

B Long release cycles

> It values speed.

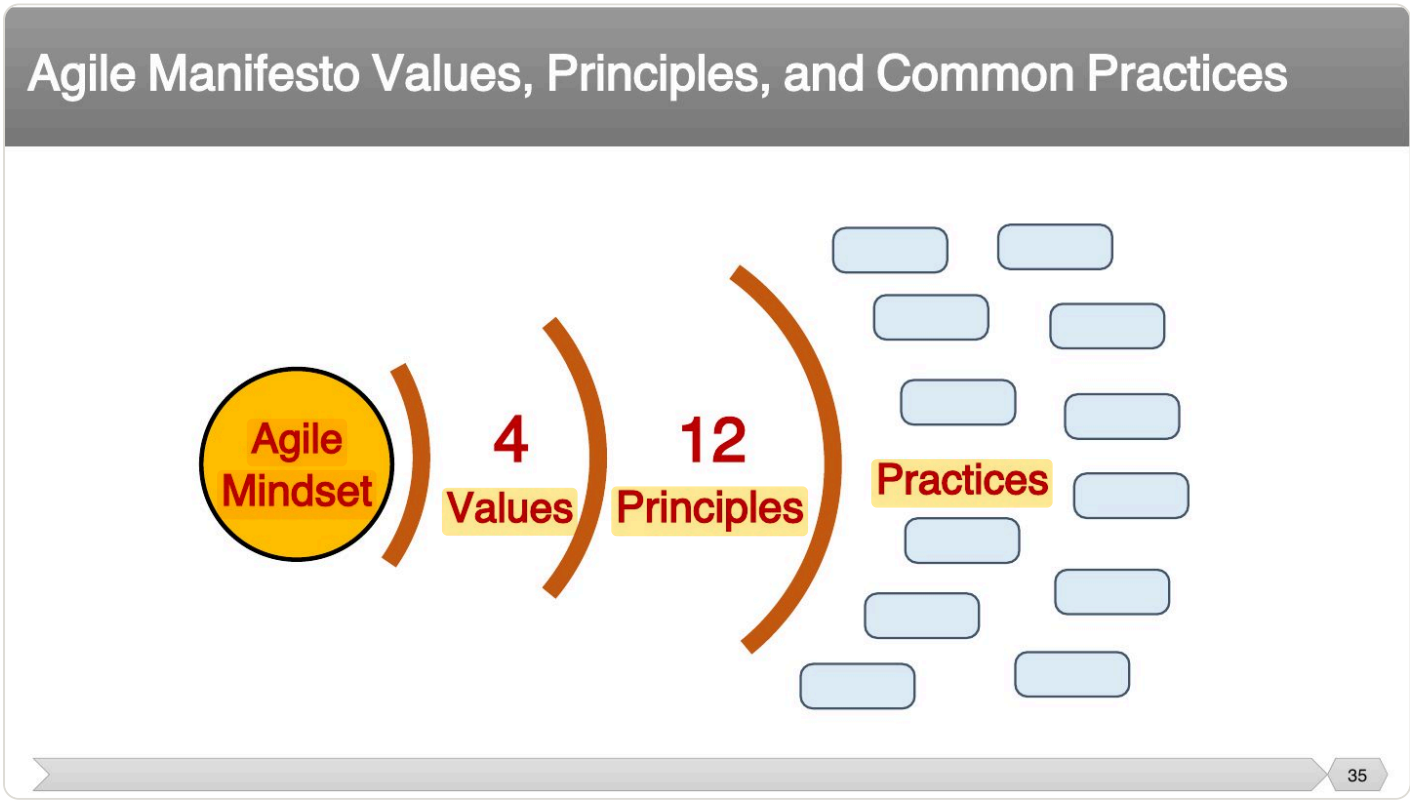
C Freezing specifications

> It responds to changing ones.

D Producing as much documentation as possible

> That's overhead.

SLIDE 35 / 98 **Agile Manifesto Values, Principles, and Common Practices**



NOTES

1. Agile is first a way of thinking. You can hold daily stand-ups and still not be agile, if you won't change the plan. The **4 values** say what matters most...
2. ...and the **12 principles** turn those values into more specific guidance, such as welcoming late changes...
3. ...and expressed through many **practices** (stand-ups, user stories, TDD), which a team chooses to fit its needs.

EXAM ANGLE

Prediction

Q1

How many values and principles define the Agile Manifesto?

A **4 values and 12 principles**

> That's the structure.

B 12 values and 4 principles

> That's reversed.

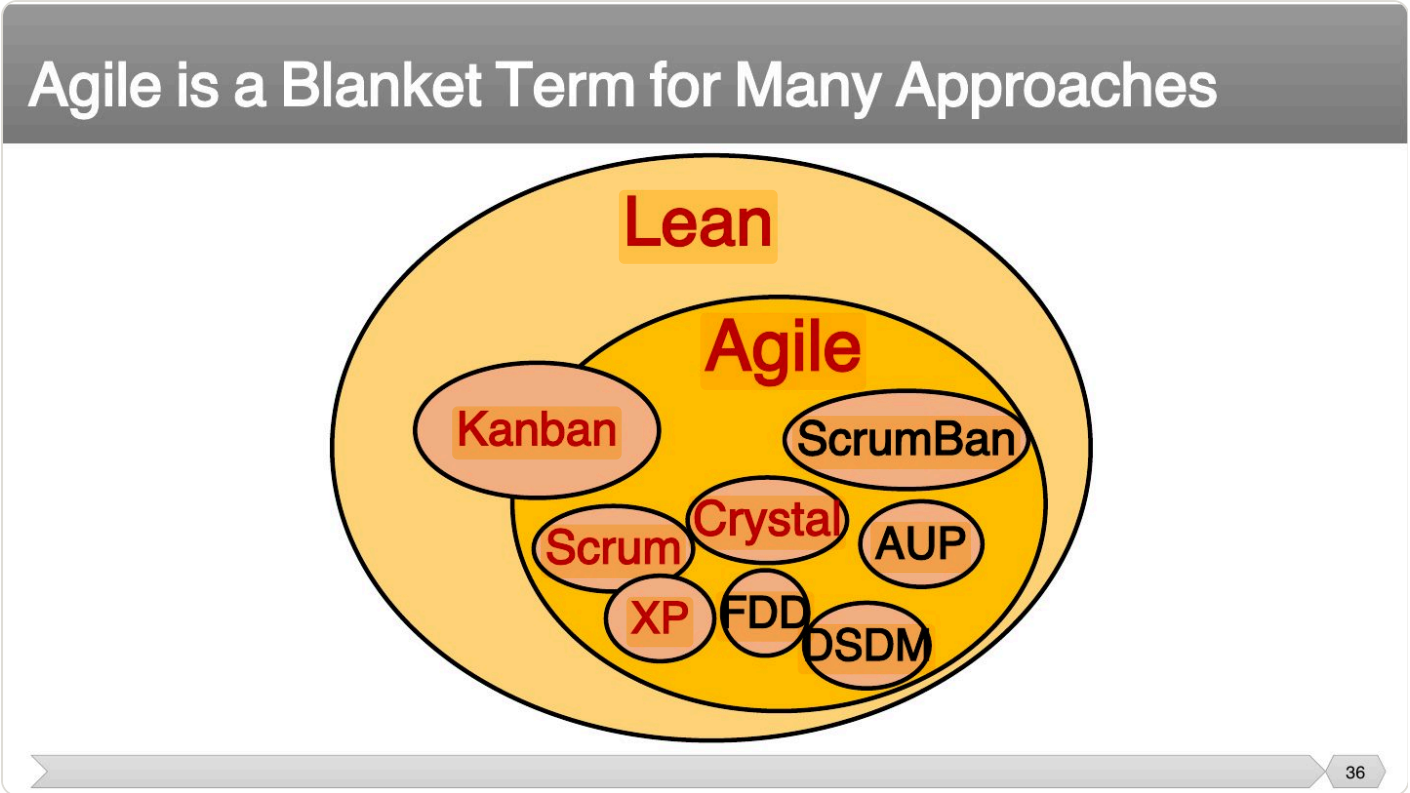
C 5 values and 10 principles

> That's not it.

D 8 values and 8 principles

> That's not it.

 Lean



NOTES

1. **Agile** isn't a single method, so "we use agile" says little by itself. Scrum and Kanban both count, and they work quite differently.
2. **Lean** is broader still, and agile sits inside it.

EXAM ANGLE

Prediction Q1

Scrum and Kanban are both covered by the term "agile".

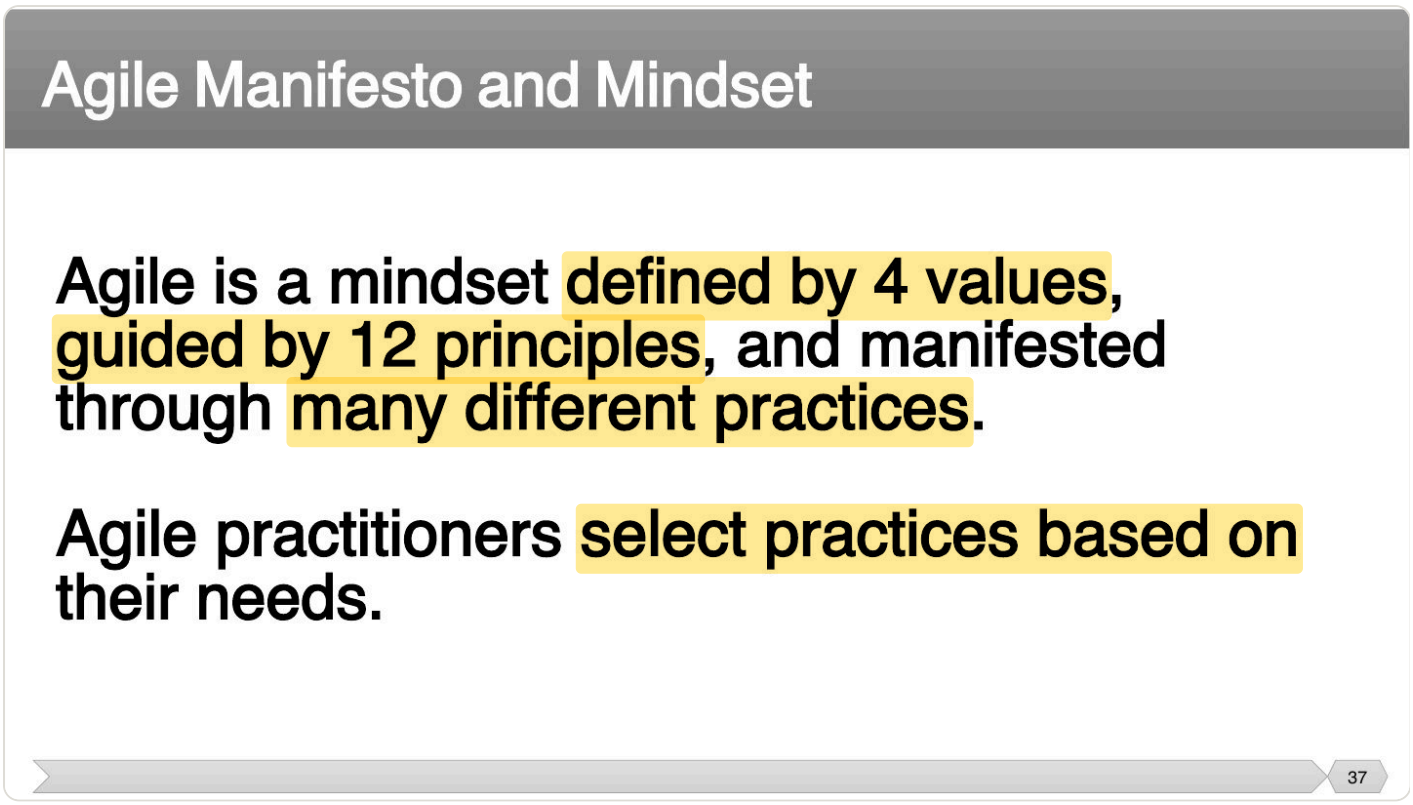
T **True**

> Agile is a blanket term for many approaches.

F False

> The slide shows both inside it.

 Agile Manifesto and Mindset



NOTES

1. Three layers, from general to specific. The **values** rarely change, while teams choose practices as they see fit. That's why two agile teams can look very different.
2. Practitioners **select practices** to fit their needs; there is no single "agile process".

EXAM ANGLE

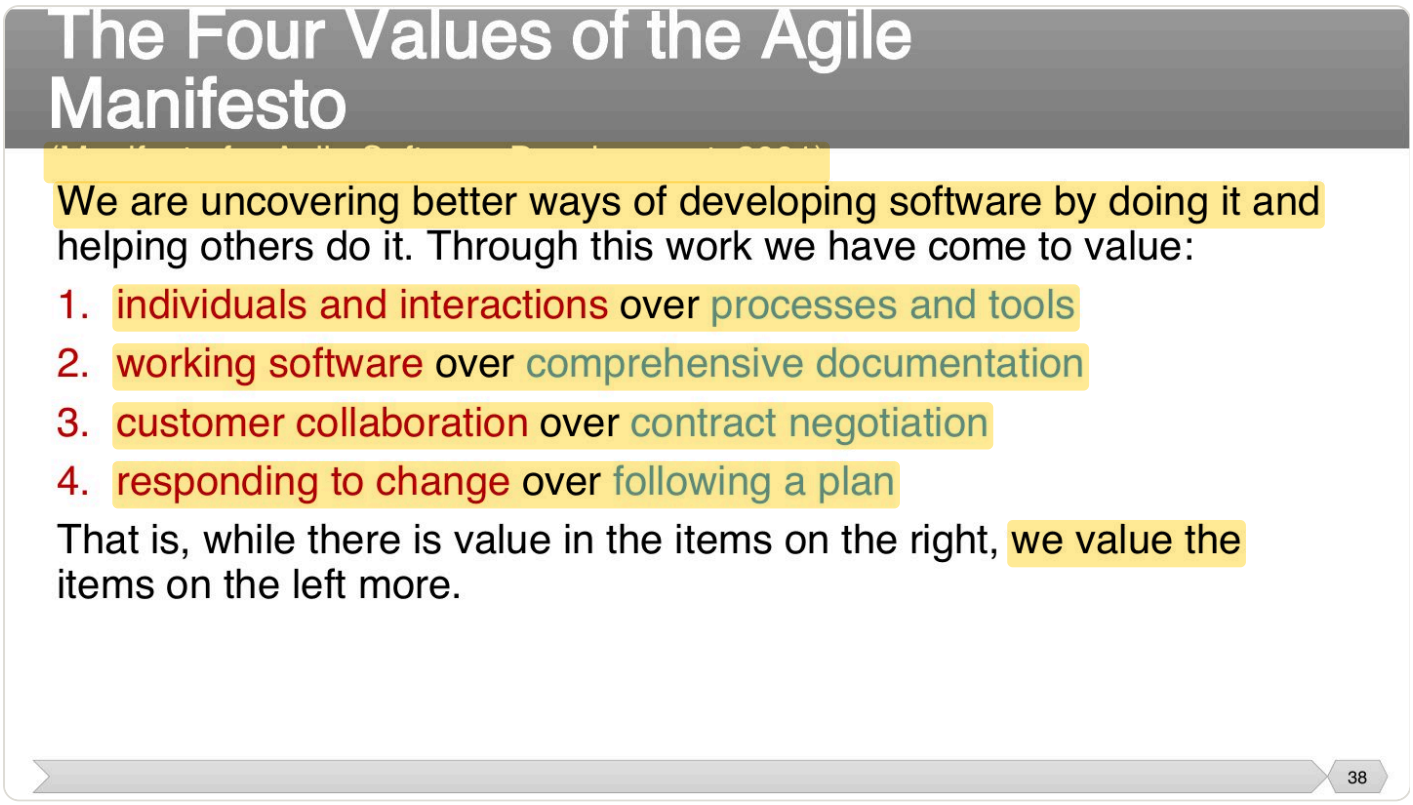
Prediction

Q1

Agile is best described as:

- A A documentation standard
- B A type of contract
- C **A mindset defined by 4 values and 12 principles, expressed through many practices**
- D A single fixed method

 Manifesto



NOTES

1. A team that talks can solve most problems quickly, while one that only follows a process may not notice it's doing the wrong thing. So the value says prefer **individuals and interactions**.
2. A 200-page plan can't show whether anything works, but running software can. That's why **working software** is the real test of progress.
3. A contract can't foresee everything. If you treat the customer as a partner, you solve surprises together, rather than arguing over the wording, so **collaboration** comes first.
4. A plan is your best guess at the time. If the market changes, following it perfectly means delivering something no longer wanted, so **responding to change** beats following the plan.
5. The items on the right still have value; the **left is valued more**. A common trap: agile does not say "no documentation".

TIPS & TRICKS

- The four values, in order: **I W**ill **C**hange **R**eadily. Individuals and interactions, Working software, Customer collaboration, Responding to change.

EXAM ANGLE

Prediction Q1

Which is one of the four values of the Agile Manifesto?

- **A Customer collaboration over contract negotiation**
- B Following a plan over responding to change
- C Processes and tools over individuals and interactions
- D Comprehensive documentation over working software

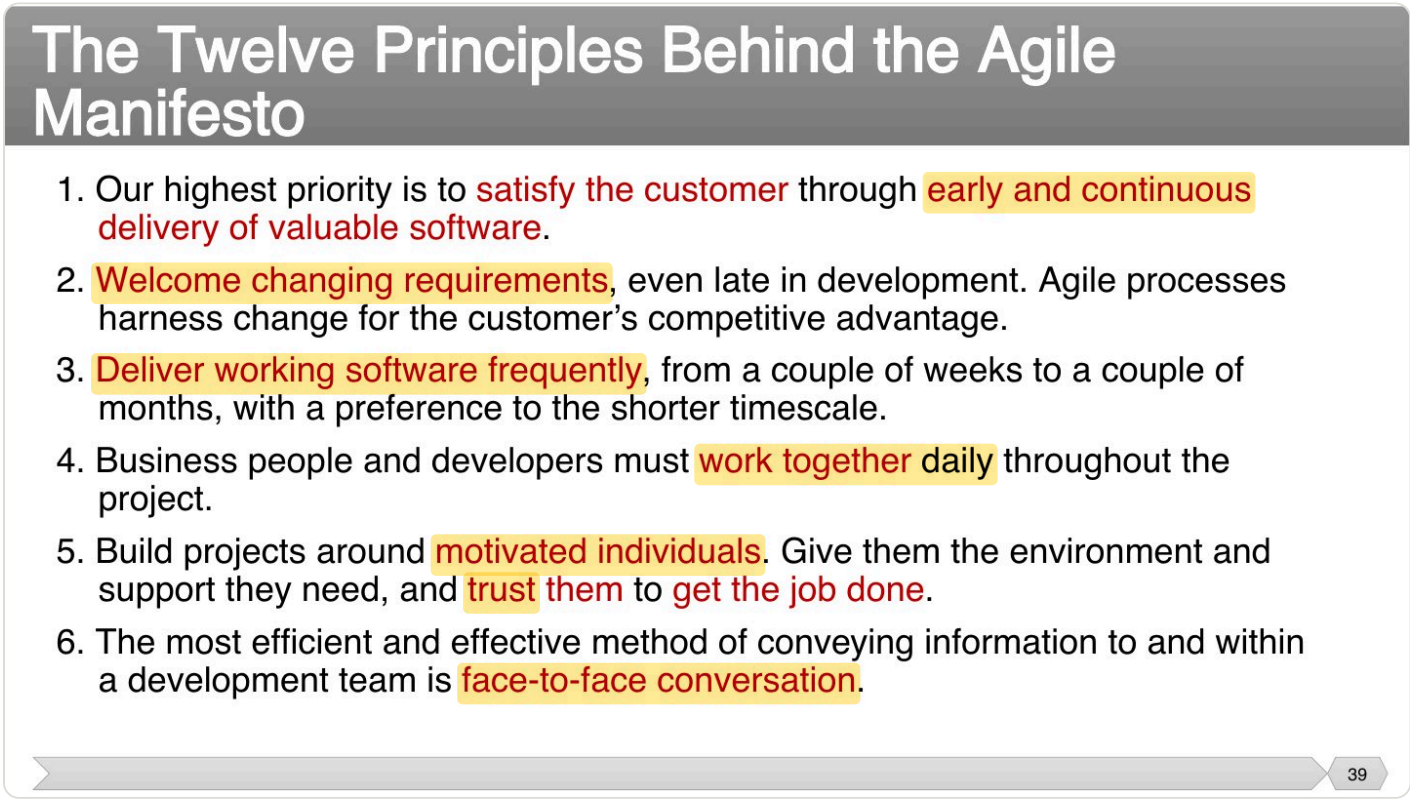
Prediction Q2

The Agile Manifesto says documentation has no value.

- T True
- **F False**

> There is value in the items on the right; the items on the left are valued more.

SLIDE 39 / 98 **Manifesto**



NOTES

1. Early delivery gets value to users sooner and gets real feedback, not guesses. That's why **early and continuous delivery** is the top priority.
2. This is a big shift, as traditional projects treat late change as a failure. Agile sees it as a sign you've learned something, so it **welcomes changing requirements** if they make the customer more competitive.
3. The shorter the cycle, the sooner you find a mistake, and the less you've wasted on it. So **frequent delivery** reduces risk as well as adding value.
4. If developers only hear from the business occasionally, they have to guess. **Working together daily** replaces guesswork with a quick question.
5. You can't control people into caring. Hire good people, give them what they need, and **trust** them. Micromanaging skilled people wastes their ability.
6. A quick talk settles what could take days of emails, because you can ask and clarify on the spot. That's why **face-to-face** is named as the most effective.

EXAM ANGLE

Prediction Q1

Late in the project, a customer asks for a change. What does the agile principle say?

A Hide the request

> That helps nobody.

B Welcome changing requirements, even late

> They can bring a competitive advantage.

C Refuse all late changes

> That's the opposite.

D Restart the whole project

> That isn't suggested.

Prediction Q2

What is the highest priority under the principles?

A Following the plan

> That's valued less.

B Writing documentation

> That's valued less.

C Reducing meetings

> That isn't principle 1.

D Satisfying the customer through early and continuous delivery

> That's principle 1.

 Manifesto

The Twelve Principles Behind the Agile Manifesto

7. Working software is the primary measure of progress.
8. Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely.
9. Continuous attention to technical excellence and good design enhances agility.
10. Simplicity—the art of maximizing the amount of work not done—is essential.
11. The best architectures, requirements, and designs emerge from self-organizing teams.
12. At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly.

NOTES

1. Hours spent and documents written don't tell you whether it works. Only **working software** does, so "90% done" with nothing running means little.
2. Overtime can win a week, but tired people make more mistakes and leave. **Sustainable** means a pace you could keep up forever, without crunches.
3. Messy code gets slower to change over time, so skipping design makes you less agile later. **Technical excellence** is therefore what keeps change cheap.
4. The best feature might be one you never had to build. **Simplicity** means asking "do we really need this?", because work not done has no bugs and no maintenance.
5. People doing the work see problems first and care more about what they designed themselves. So **self-organizing teams** tend to produce better designs than plans handed down.
6. A process that's never reviewed never improves. The team stops regularly to **reflect** on how it's working, and agrees on something to change. That's a retrospective.

EXAM ANGLE

Prediction Q1

According to the agile principles, the primary measure of progress is:

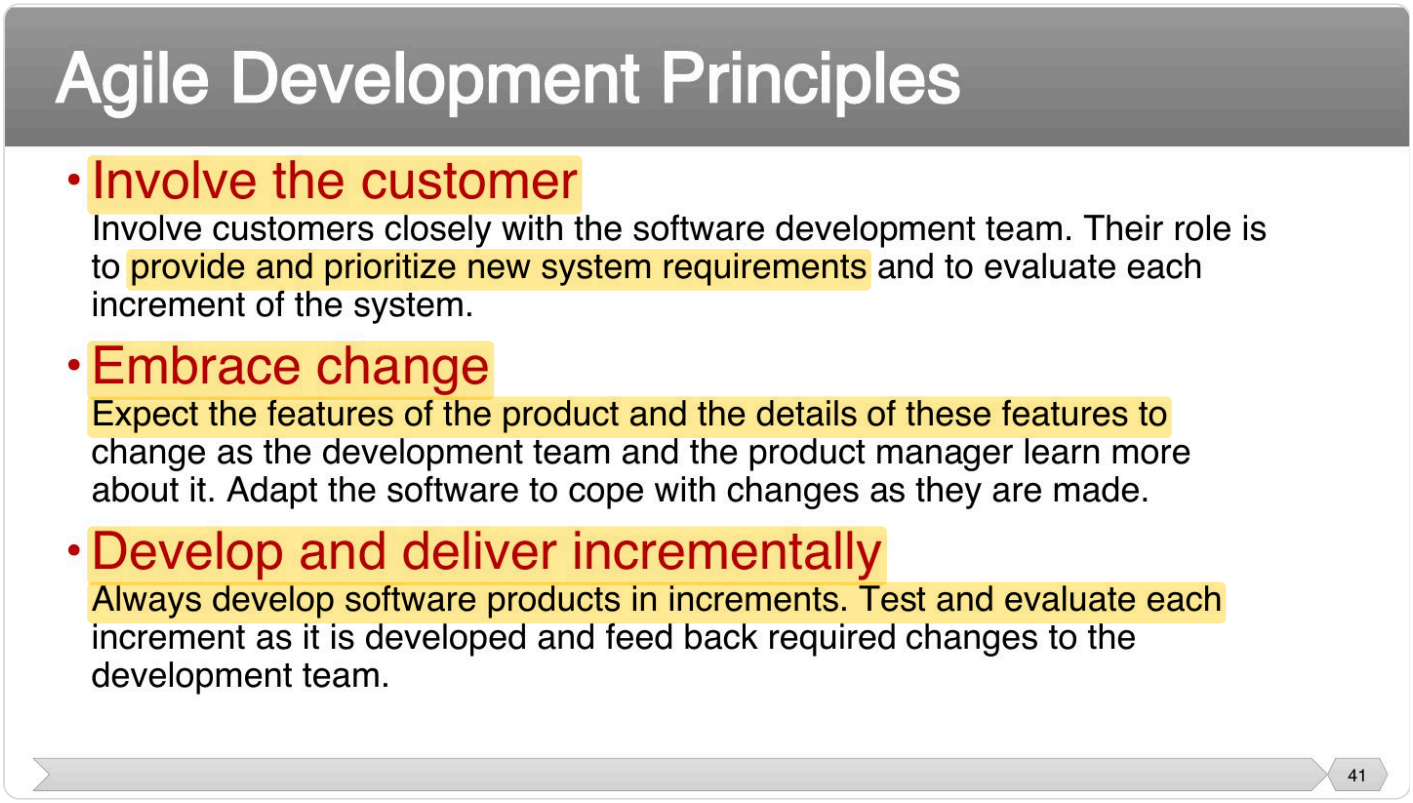
A Hours worked

B **Working software**

C Documents completed

D Lines of code

SLIDE 41 / 98 **Agile Development Principles**



NOTES

1. Nobody knows what customers want better than customers do. Having them **prioritize** and judge each increment prevents you building the wrong thing for months.
2. Don’t treat change as a failure of planning. You’ll learn more about what’s needed as you go, so **embrace change** and build software that’s easy to adjust.
3. Each piece is a chance to check you’re right. If you only **deliver incrementally**, a wrong turn costs one increment rather than the whole project.

EXAM ANGLE

Prediction

Q1

In agile, who provides and prioritizes new requirements?

A Nobody, as requirements don’t change

> They do change.

B The HR department

> That’s unrelated.

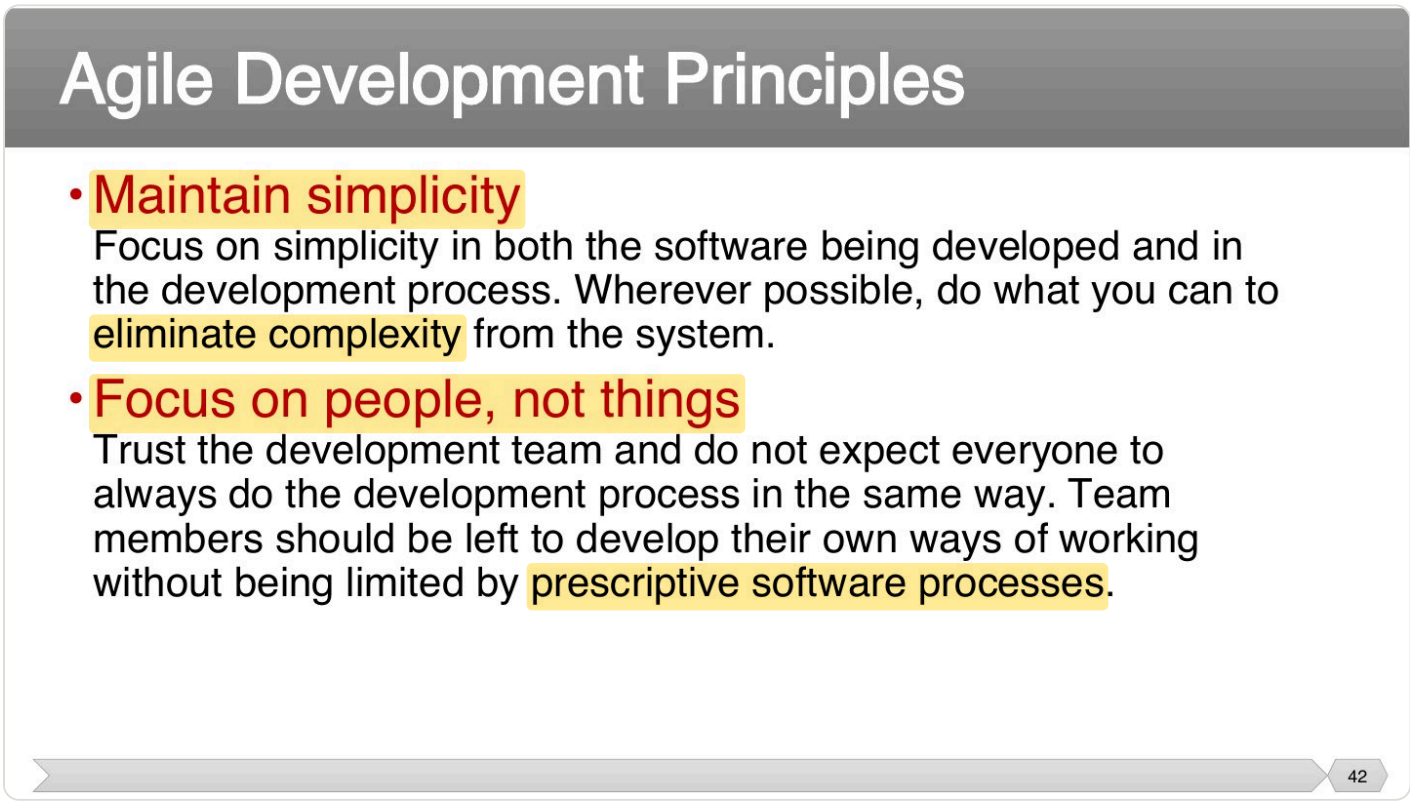
C Only the testers

> That’s not their role.

D **Customers involved closely with the team**

> That’s their role in the principle.

SLIDE 42 / 98 Agile Development Principles



NOTES

1. Complexity is what makes software hard to change later, and extra process does the same. Ask “can this be simpler?” of the code and of how you work: **maintain simplicity**.

2 People differ, and forcing everybody into one way of working wastes their strengths. **Trust the team** to find what works for them, rather than imposing a rigid process.

TIPS & TRICKS

- Slides 41 and 42 together, **MIdFiEID**: Maintain simplicity, Involve the customer, Focus on people, Embrace change, Develop incrementally.

EXAM ANGLE

Prediction Q1

A manager makes every developer follow one rigid routine. Which principle does this go against?

A Embrace change

> That's about changing features.

B Deliver incrementally

> That's about small releases.

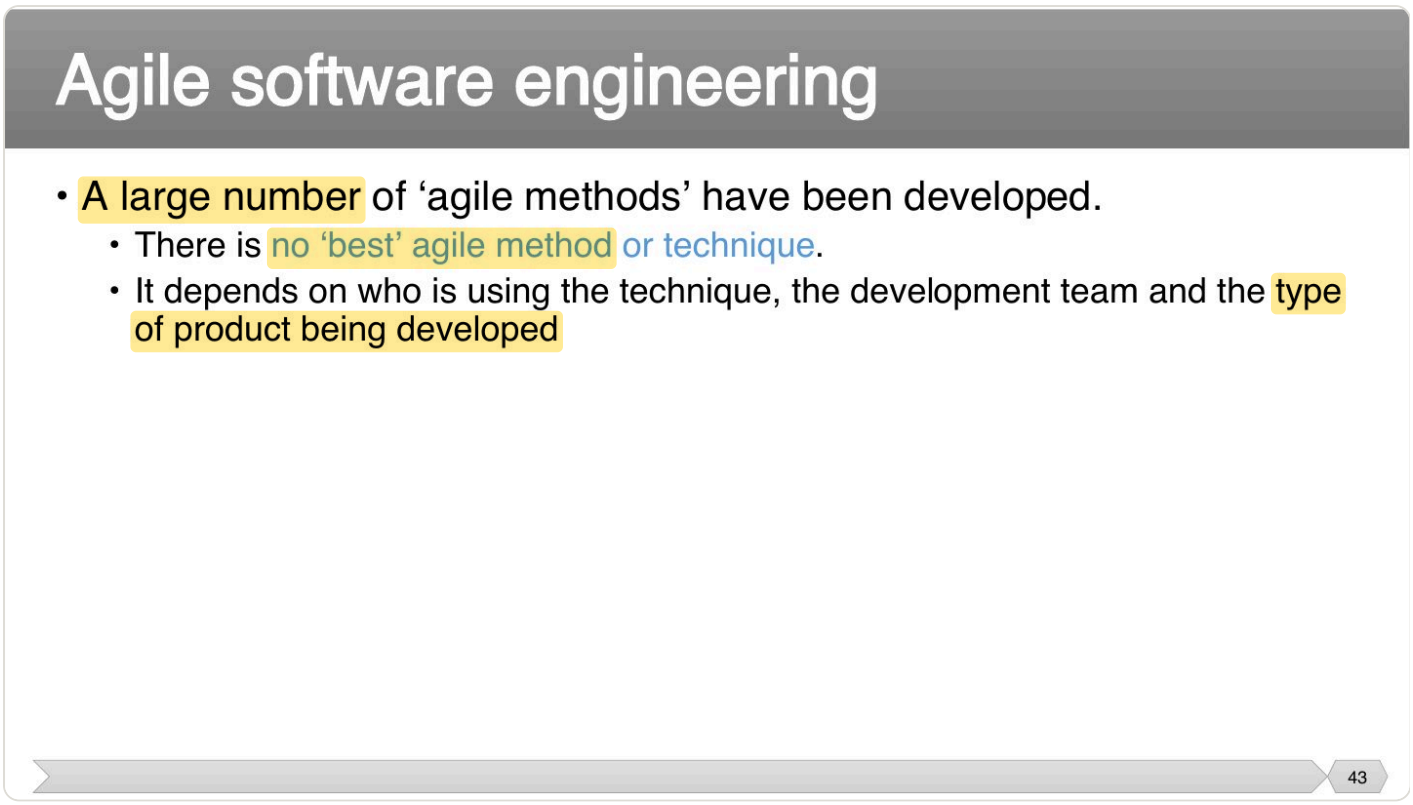
C **Focus on people, not things**

> Team members should develop their own ways of working.

D Maintain simplicity

> That's about reducing complexity.

 Agile software engineering



NOTES

1. Don’t be overwhelmed by the names, since they share the same core ideas. The **many** methods mostly differ in emphasis...
2. ...and there is **no single best one**...
3. A small startup and a bank may need different things. So asking “what’s the best agile method?” is the wrong question, and you should ask what suits this **type of product** and team.

EXAM ANGLE

Prediction Q1

There is one best agile method for all teams.

T True

> The slide says there’s no best method.

F False

> It depends on the team and the type of product.

 Agile methods

Agile methods

- Plan-driven development evolved to support the engineering of large, long-lifetime systems
  - This approach is based on controlled and rigorous software development processes that include detailed project planning, requirements specification and analysis and system modelling.
  - However, plan-driven development involves significant overheads and documentation and it does not support the rapid development and delivery of software.
- Agile methods were developed in the 1990s to address this problem.
  - These methods focus on the software rather than its documentation, develop software in a series of increments and aim to reduce process bureaucracy as much as possible.

NOTES

1. It isn't a bad approach, just built for a different problem. A system that must run for decades, built by hundreds, needs careful control, which is why it suits **large, long-lifetime** systems...
2. ...and that's what all the rigor is for. **Detailed planning** and system models let very large groups co-ordinate without constantly talking...
3. ...but for a small product in a fast market, all that **overhead** means you may arrive too late, as requirements can change before you've finished the documents.
4. Time saved on paperwork goes into building. That's the trade: less **bureaucracy** and documentation, which is fine for a small team that can just talk to each other.

EXAM ANGLE

Prediction Q1

What problem were agile methods developed to address?

A The overhead of plan-driven development, which didn't support rapid delivery

> That's what the slide says.

B A lack of documents

> They reduce documents.

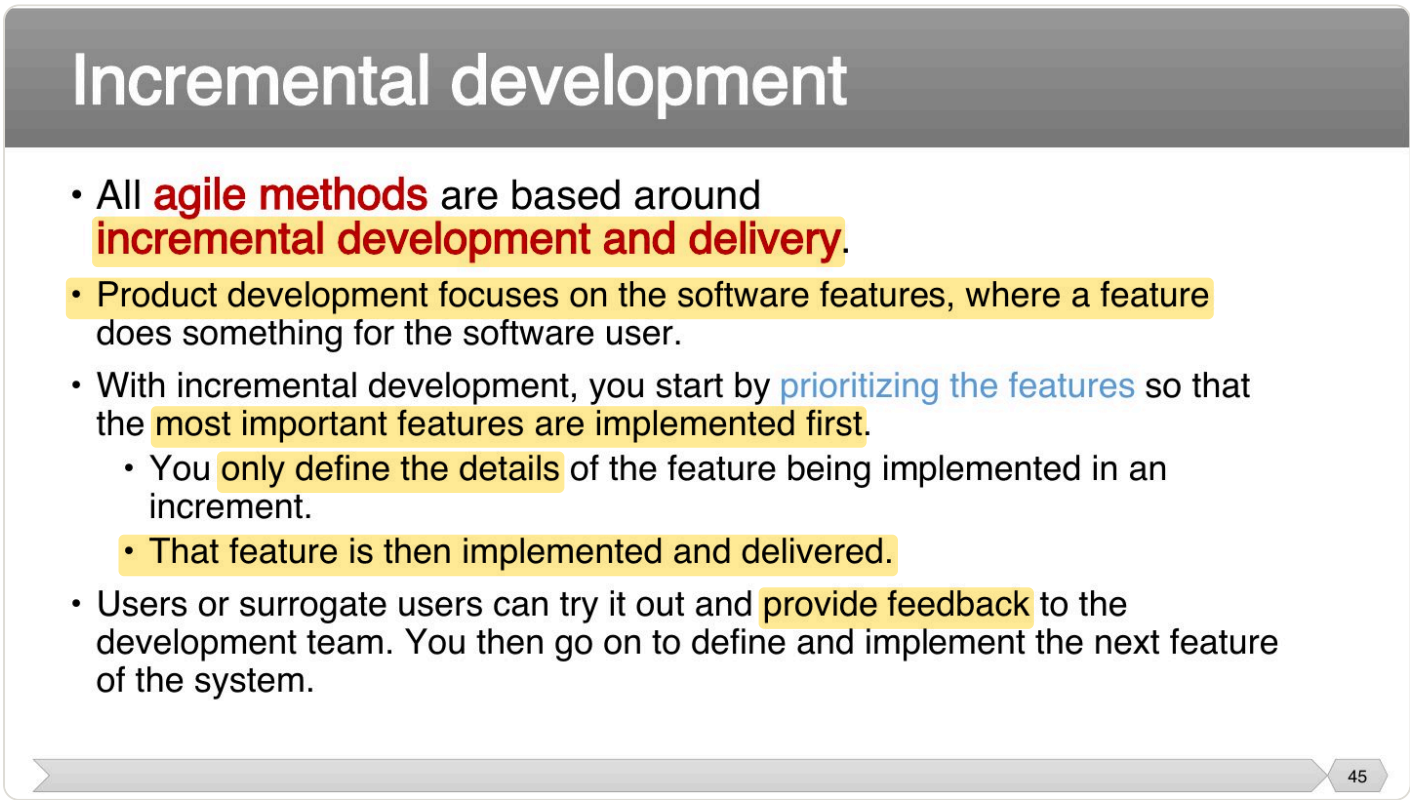
C Long-lifetime hardware

> Plan-driven was made for that.

D Too little bureaucracy

> They reduce it further.

SLIDE 45 / 98 **Incremental development**



NOTES

1 A feature is defined by what it does for the user, not how it's built. Organizing around features with **incremental delivery** keeps everybody focused on visible value.

2. If you run out of time or money, you want the most useful things already done. Doing **the most important first** means you lose only the least valuable work.
3. Only define the **details of the feature being built in this increment**; the rest stays rough until it is needed.
4. Real use reveals what discussions can't. Letting users **give feedback** after each feature means the next piece of work benefits from what you've just learned.

EXAM ANGLE

Prediction Q1

Which features get implemented first?

A The most important ones

> Features are prioritized.

B The easiest ones

> Priority comes from importance.

C Random ones

> They're prioritized.

D The most recent requests

> That isn't the rule.

Prediction Q2

When do you define a feature's details?

A After release

> That's too late.

B Never

> You do define them, later.

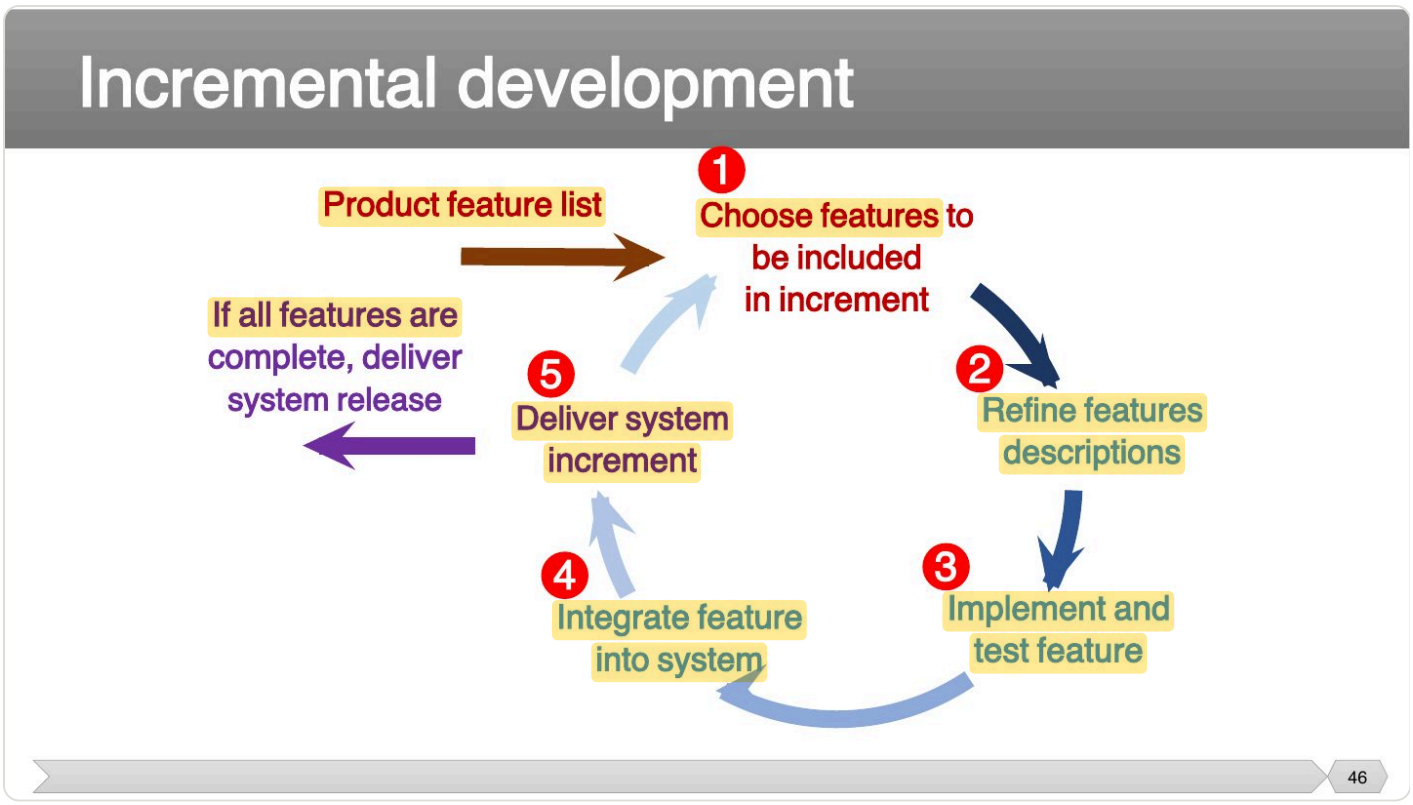
C Only when it's being implemented in the increment

> The rest stays rough.

D All at the very start

> That's plan-driven.

 Incremental development



NOTES

1. Start from the full list but pull in only what fits this round, so the team isn't **choosing** everything at once...
2. ...only now do you work out the details, since by this time you know most about each feature, so you **refine** only what's going to be built...
3. ...and build and check it together. If you **implement and test** at the same time, bugs get found while the code's still fresh...
4. ...then you check it works with everything else. **Integrating** each feature right away prevents the big painful merge at the end...
5. ...and put it in front of people. Each **delivery** is another chance to learn. Repeat until the list is empty.

EXAM ANGLE

Prediction Q1

What comes right after choosing features for the increment?

A Implement and test

> That comes after refining.

B **Refine their descriptions**

> That's the second step.

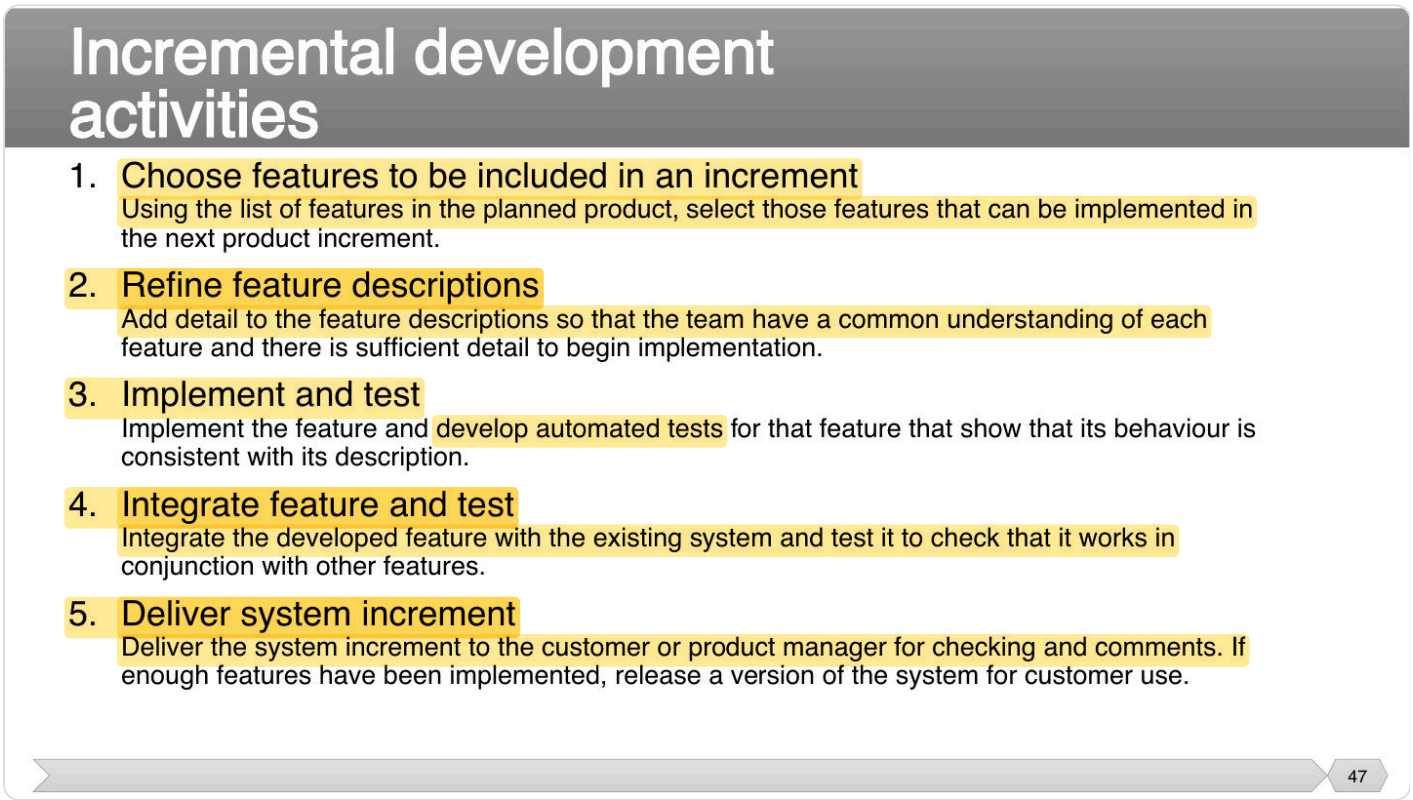
C Integrate the features

> That comes later.

D Deliver the system

> That's last.

SLIDE 47 / 98 **activities**



NOTES

1 Pick what can realistically be finished next. **Choosing** a small amount keeps each cycle short and its results visible.

2. Just enough detail for everyone to understand it the same way. **Refining** too early wastes effort if the feature changes, and too late means confusion.
3. Automated checks mean anyone can confirm that the feature still works, any time. **Automated tests** also stop later changes silently breaking it.
4. A feature can work alone and still fail with others. **Integrate and test** checks they work together, as the problems often live between the parts.
5. Someone has to judge it. Giving it to the customer or product manager brings comments that shape the next increment. A public release can wait until enough has been **delivered**.

TIPS & TRICKS

- The order matters here: **C**ooks **R**oast **I**n **I**ron **D**ishes. Choose features, Refine them, Implement and test, Integrate and test, Deliver.

EXAM ANGLE

Prediction Q1

What does “implement and test” include?

A Customer contracts

> That’s unrelated.

B Only manual checks at the end

> The slide says automated.

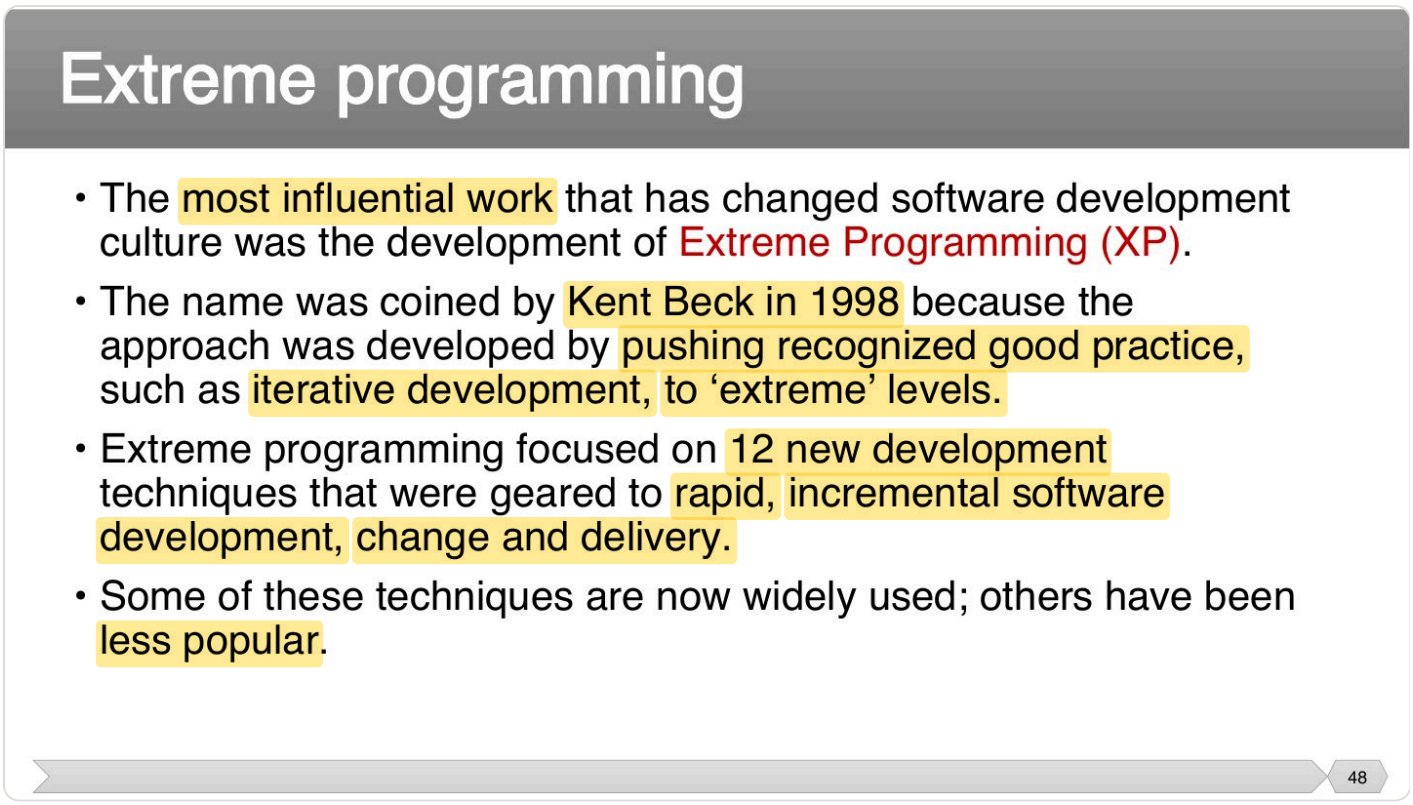
C No tests

> Tests are essential.

D **Automated tests showing the behavior is as described**

> The tests come with the feature.

 Extreme programming



NOTES

1. Many everyday practices came from here, so it's why it's called **most influential**, even for teams who've never used XP as a whole.
2. If code review is good, why not review constantly? That's pair programming. If testing is good, why not test all the time? That's the **extreme** idea: take what works and push it further.
3. You don't need to memorize all **12**, and the next slides cover the ones in wide use. Together, they're built to make frequent change safe...
4. ...and teams picked what suited them. Pair programming all day is **less popular**, as it's tiring and costly, while tests and continuous integration became the norm.

EXAM ANGLE

Prediction Q1

Why is it called “Extreme” Programming?

A **It pushes recognized good practice to extreme levels**

> That’s the origin of the name.

B It’s only for sports software

> That isn’t why.

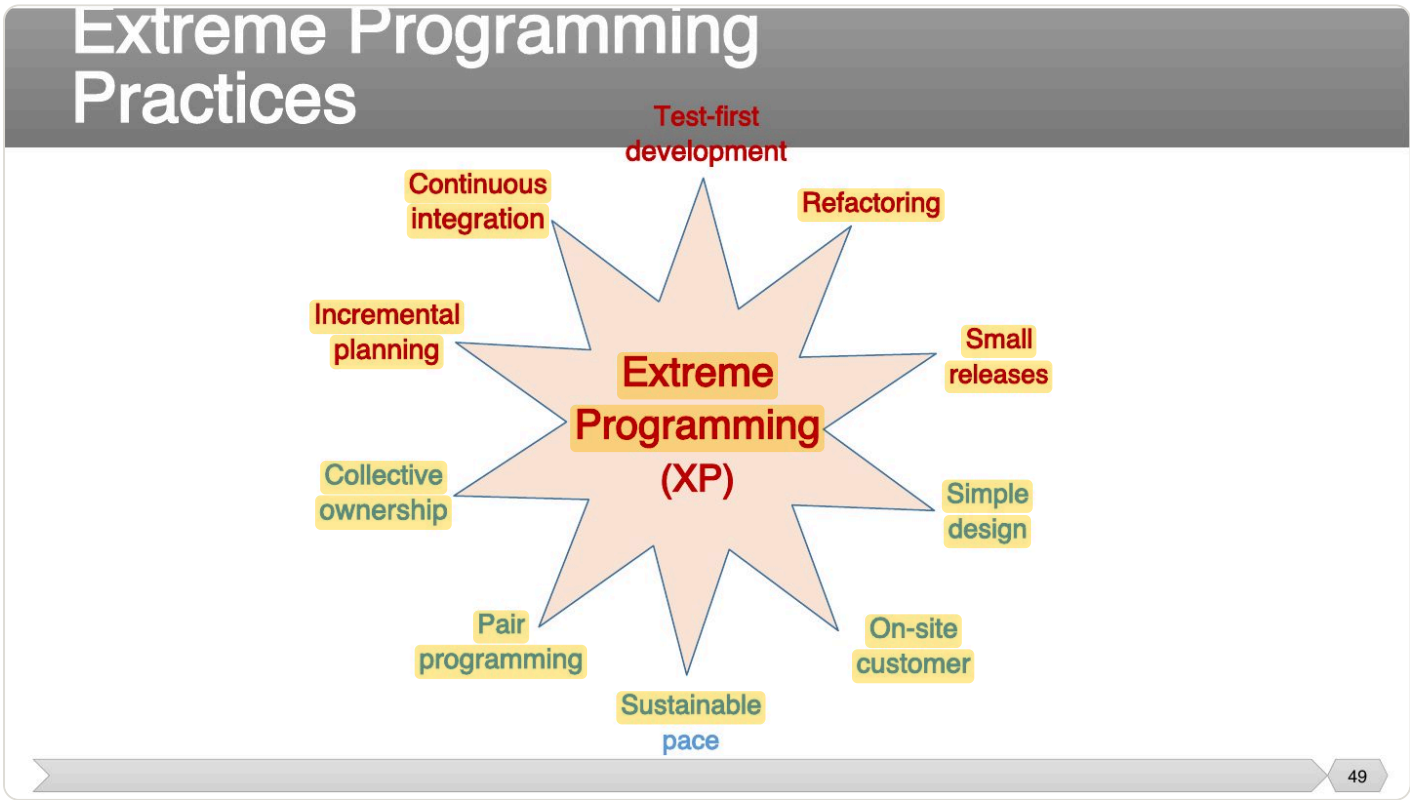
C It avoids testing

> It’s the opposite.

D It’s dangerous

> That isn’t why.

SLIDE 49 / 98 **Practices**



NOTES

1. These all make frequent change safe: tests catch mistakes, **refactoring** keeps code clean, and small releases limit the damage if something goes wrong.

2. These tackle knowledge. With **pair programming** and shared ownership, no one person is the only one who understands a module, and an **on-site customer** gives instant answers.
3. **Sustainable pace**: no long overtime. The next slides cover the widely adopted ones.

EXAM ANGLE

Prediction Q1

Which is an XP practice?

A Long overtime

> XP asks for a sustainable pace.

B Big up-front design

> XP prefers simple design.

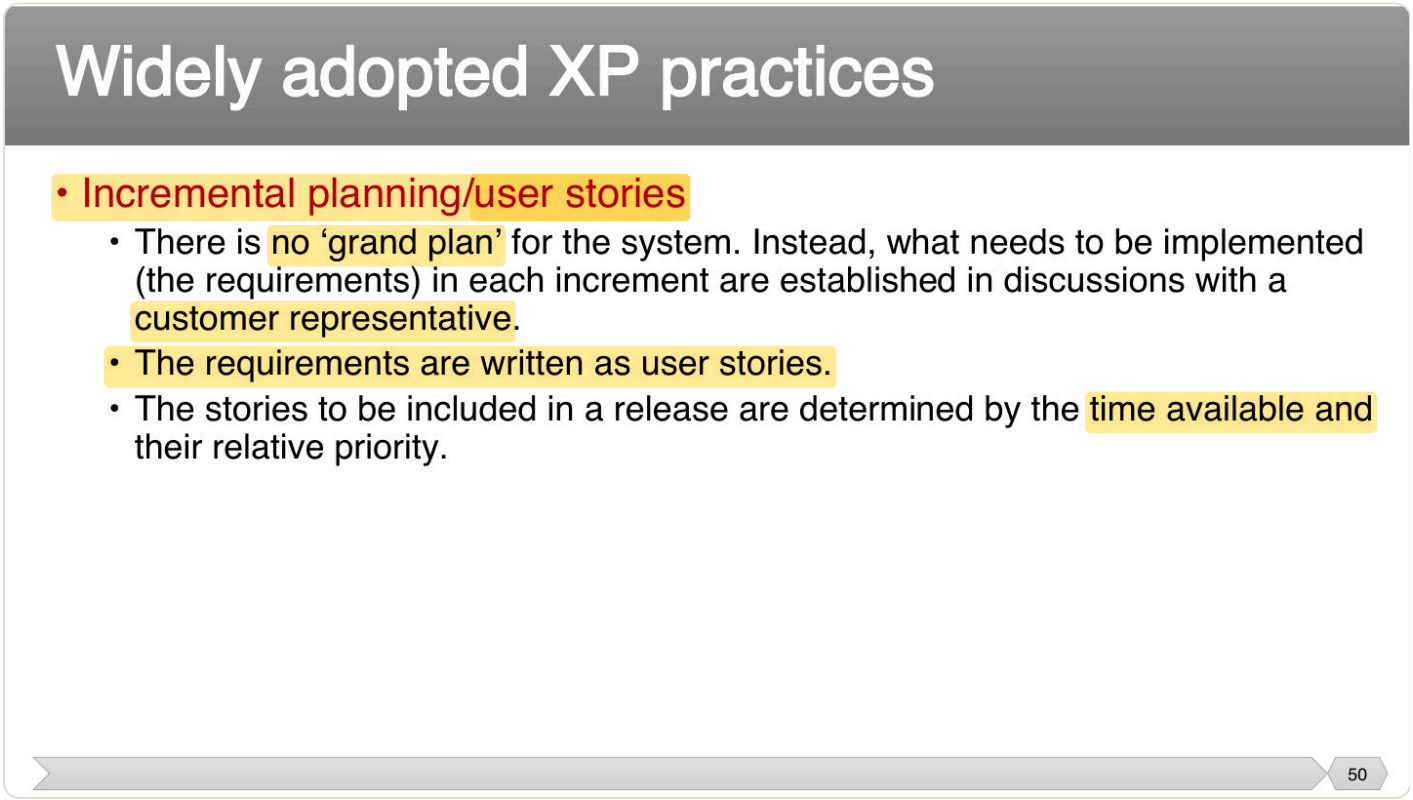
C Individual code ownership

> XP uses collective ownership.

D **Pair programming**

> It's on the star.

SLIDE 50 / 98 **Widely adopted XP practices**



NOTES

1. A big plan would be out of date before you got there. Deciding one increment at a time with a **customer representative** keeps the work tied to what's needed now.
2. A **user story** states who wants something and why, not how to build it. The "so that" matters, as it tells you whether a simpler solution would do.
3. You can't do everything by the deadline, so the two limits decide: what's most valuable, and what fits in the **time available**. That's how scope stays flexible.

TIPS & TRICKS

- Slides 50 to 54, **STRICt**: Small releases, Test-driven development, Refactoring, Incremental planning, Continuous integration.

EXAM ANGLE

Prediction Q1

How are requirements written in XP?

A As org charts

That's not it.

B As a 200-page specification

That's plan-driven.

C As contracts

That's not it.

D As user stories

That's on the slide.

Prediction Q2

What decides which stories go in a release?

A The time available and relative priority

Both limits apply.

B Only who wrote them

That doesn't matter.

C The size of the office

That doesn't matter.

D Alphabetical order

That doesn't matter.

Widely adopted XP practices

Widely adopted XP practices

- Small releases
  - The minimal useful set of functionality that provides business value is developed first.
  - Releases of the system are frequent and incrementally add functionality to the previous release.

NOTES

1. Ask “what’s the smallest thing that’s useful?”, and build that first. A **minimal useful set** starts earning value and feedback much sooner than a full system.
2. Each release is small, so it’s low risk, and it’s quick to fix if wrong. **Frequent** releases also keep customers seeing progress.

EXAM ANGLE

Prediction Q1

What gets developed first under small releases?

A Only the database

> It needs to be useful to the business.

B Only the documentation

> It wants working functionality.

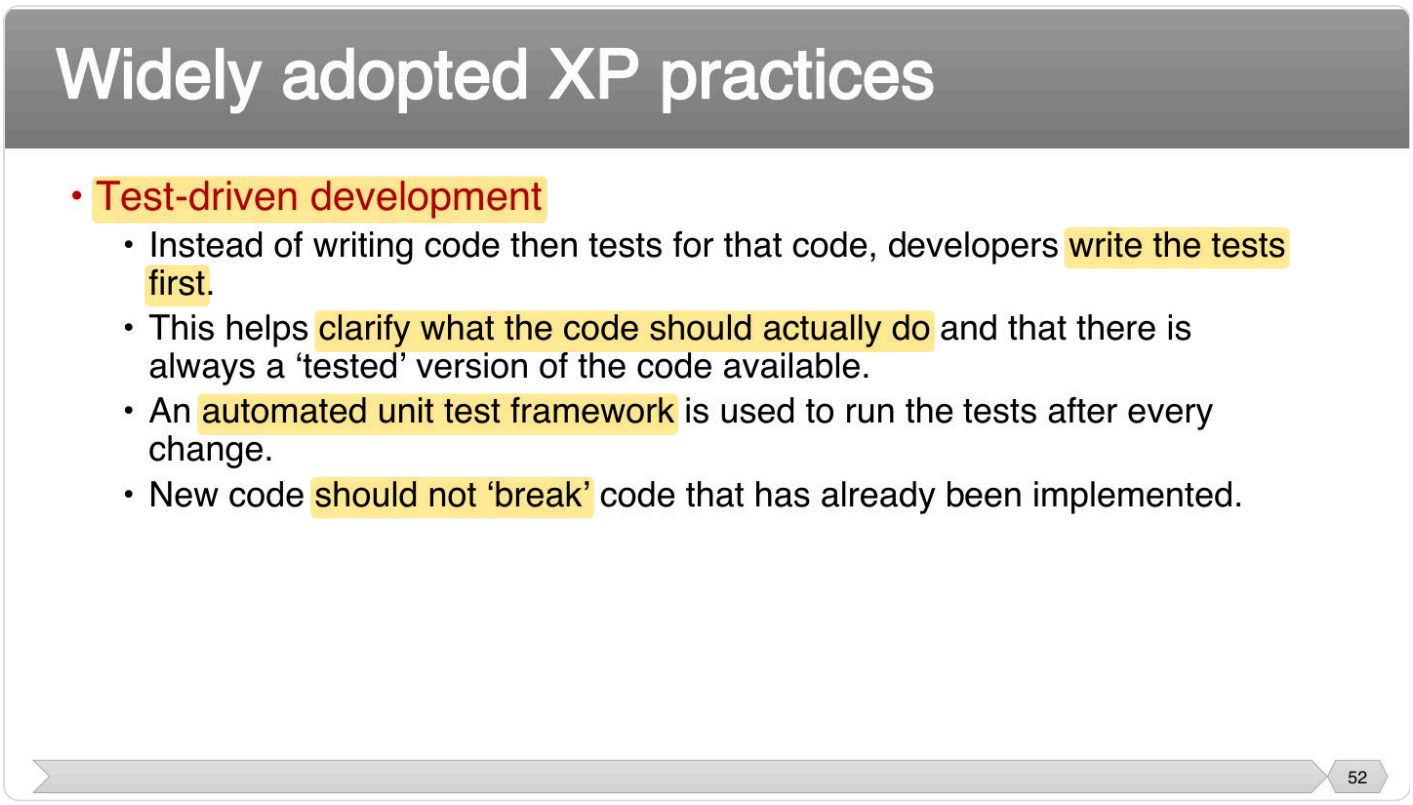
C The complete system

> That’s the opposite.

D The minimal useful set of functionality with business value

> That’s the idea.

 Widely adopted XP practices



NOTES

1. It sounds backwards, yet writing the test first forces you to define "working" before you start. That's the idea behind **test-driven development**.
2. If you can't write a test for it, you may not understand what it should do yet, so the test **clarifies** the job. You'll also always have a version known to pass.
3. Nobody has time to manually re-test everything after every change. **Automated** runs make that free, and that's what makes frequent change safe...
4. ...so if a change breaks an old feature, you find out within minutes, while it's easy to see what caused it, rather than weeks later.

EXAM ANGLE

Prediction Q1

In test-driven development, developers:

A Only test manually

B Write tests after release

C **Write tests before writing the code**

D Skip unit tests

Widely adopted XP practices

Widely adopted XP practices

- Continuous integration
  - As soon as the work on a task is complete, it is integrated into the whole system and a new version of the system is created.
  - All unit tests from all developers are run automatically and must be successful before the new version of the system is accepted.

NOTES

1. The longer code stays separate, the more painful the merge. **Integrating** each finished task right away keeps the differences tiny, and everyone works from the current system.
2. All unit tests from all developers run automatically and **must pass** before the new version is accepted, so integration problems appear in minutes instead of at the end.

EXAM ANGLE

Prediction Q1

When a task is done in continuous integration, what happens?

A **It’s integrated right away, and a new version is built**

> All tests must pass for it to be accepted.

B It’s emailed to the customer

> That isn’t what’s described.

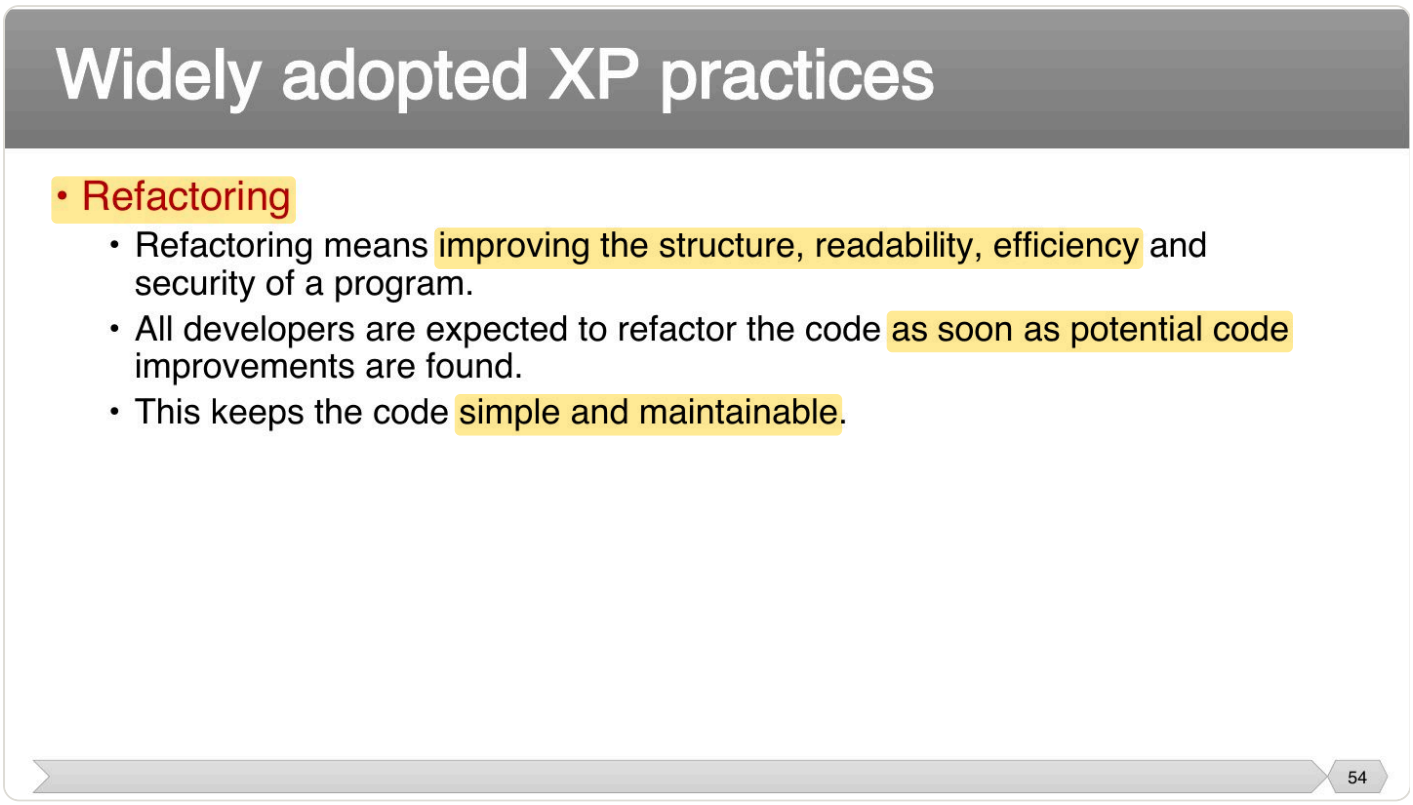
C It waits until the end of the project

> That’s the problem it solves.

D It’s deleted

> That isn’t what’s described.

 Widely adopted XP practices



NOTES

1. Think of tidying a kitchen while the same meals still come out. Users see no difference, but **improving the structure** makes the next change easier and safer. Tests prove nothing broke.
2. Waiting for a “cleanup phase” means it never happens. Fixing small problems **as soon as** you see them stops them from building up, and this only works thanks to automated tests...
3. ...the payoff comes later. **Maintainable** code stays cheap to change, which is what lets a team keep moving quickly over the years.

EXAM ANGLE

Prediction Q1

A developer tidies code without changing what it does. What is this?

A Adding a feature

> That changes what it does.

B Deployment

> That's releasing.

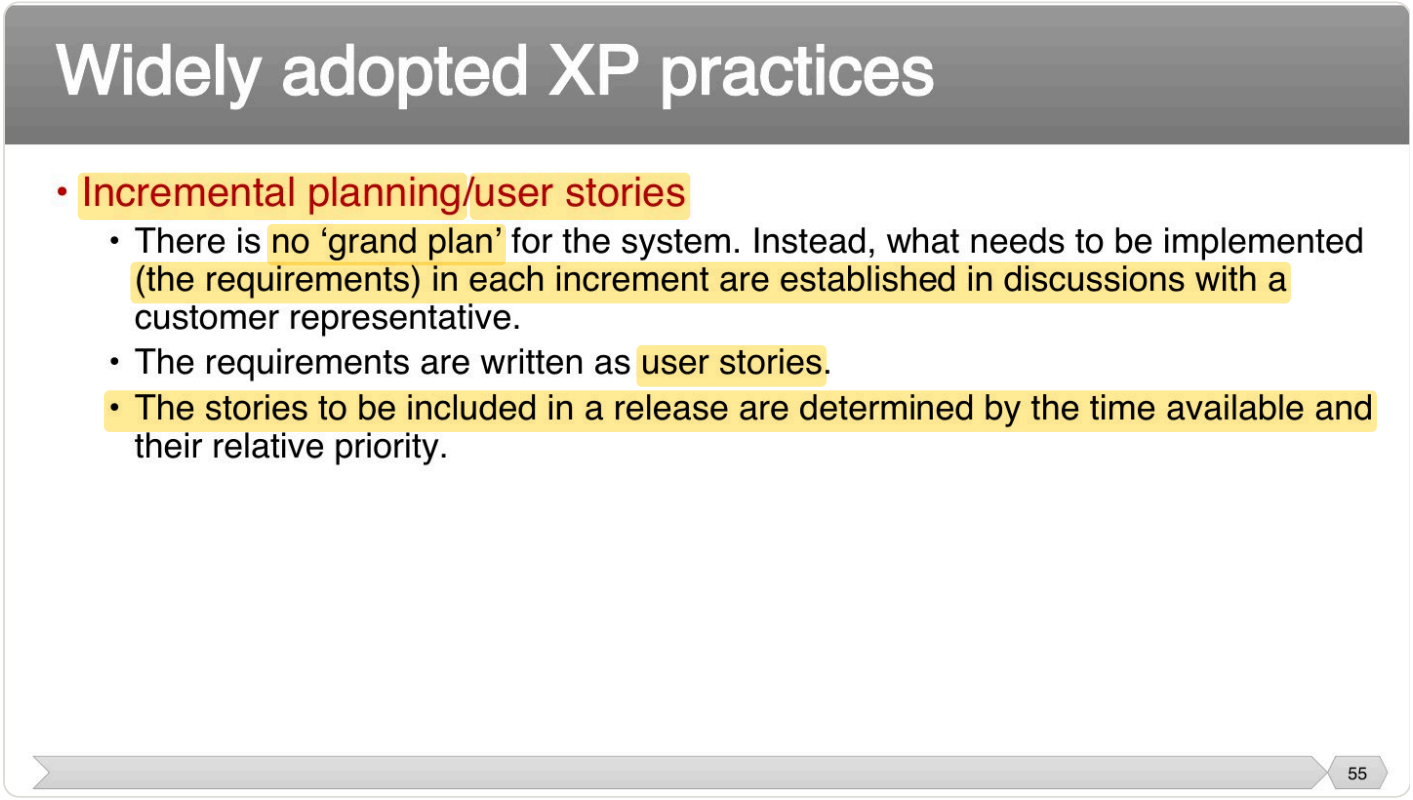
C **Refactoring**

> It improves the structure only.

D Scope creep

> No scope is added.

SLIDE 55 / 98 **Widely adopted XP practices**



NOTES

1 The same material appears here again: **incremental planning** with **user stories**. If it's on two slides, it's likely worth remembering.

EXAM ANGLE

Prediction Q1

What replaces a “grand plan” in XP?

A A fixed five-year plan

> That’s what it avoids.

B A phase gate

> That’s predictive.

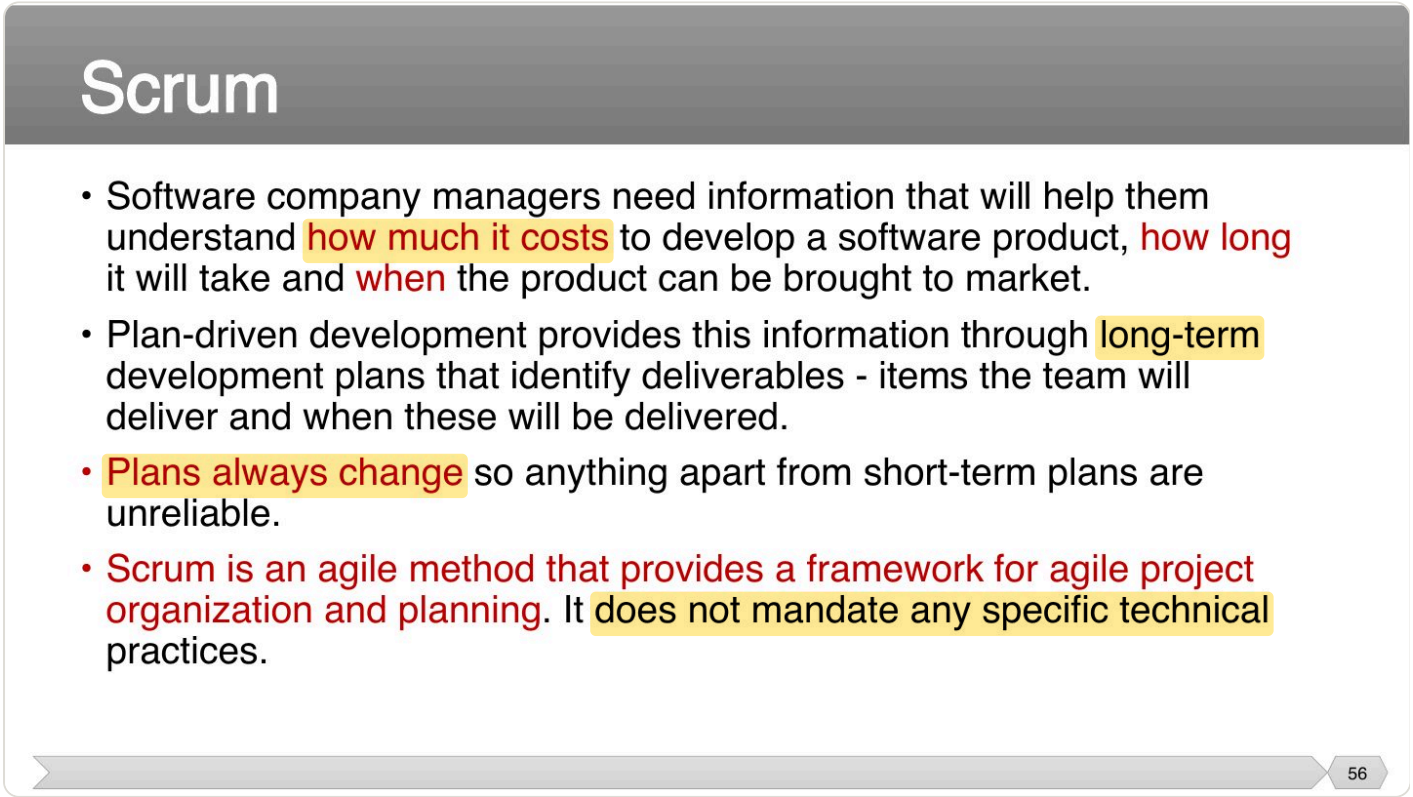
C **Incremental planning with user stories**

> This repeats slide 50.

D No planning at all

> It still plans.

SLIDE 56 / 98 **Scrum**



NOTES

1 A business can’t fund something without knowing roughly what it costs and when it arrives. Scrum doesn’t remove this need for **how much**...

2. Traditional planning's answer sounds reassuring: a full schedule with dates. But a **long-term plan** gives confidence, not necessarily accuracy...

3. The further ahead a plan looks, the more it's just guessing. So when **plans always change**, trusting a detailed plan for next year is a mistake. Short-term plans are the ones you can rely on.

4. Scrum is about how a team organizes and plans, not how it writes code. So it **mandates no technical practices**, which is why it's often combined with XP. A common trap.

EXAM ANGLE

Prediction Q1

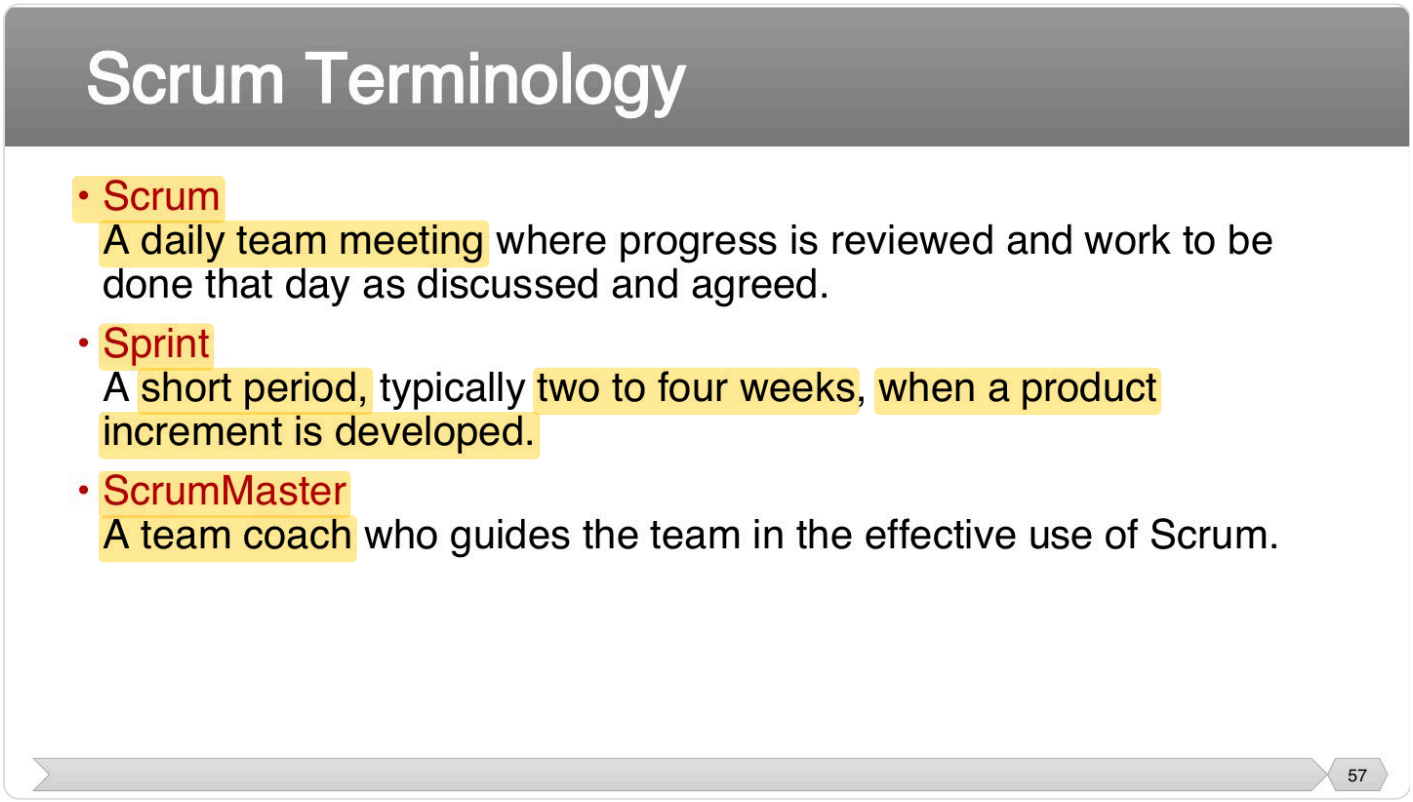
Scrum mandates specific technical practices such as pair programming.

T True

F **False**

> Scrum provides a framework for organization and planning only; XP defines the engineering practices.

SLIDE 57 / 98 **Scrum Terminology**



NOTES

1. It keeps everyone in sync without long meetings. If someone's stuck, it's out in the open within a day. Careful: in this slide "**Scrum**" means the meeting, as well as the method's name.

2. A short, fixed period forces real progress and limits how long you can go wrong. **Sprint** length usually stays the same, so the team gets a steady rhythm.

3 The **ScrumMaster** is not a boss. They help the team use the method well and remove obstacles, so don't assume it's just a new name for project manager.

EXAM ANGLE

Prediction Q1

In this slide, what does "Scrum" mean as a term?

A A two-year plan

> That's not it.

B A contract

> That's not it.

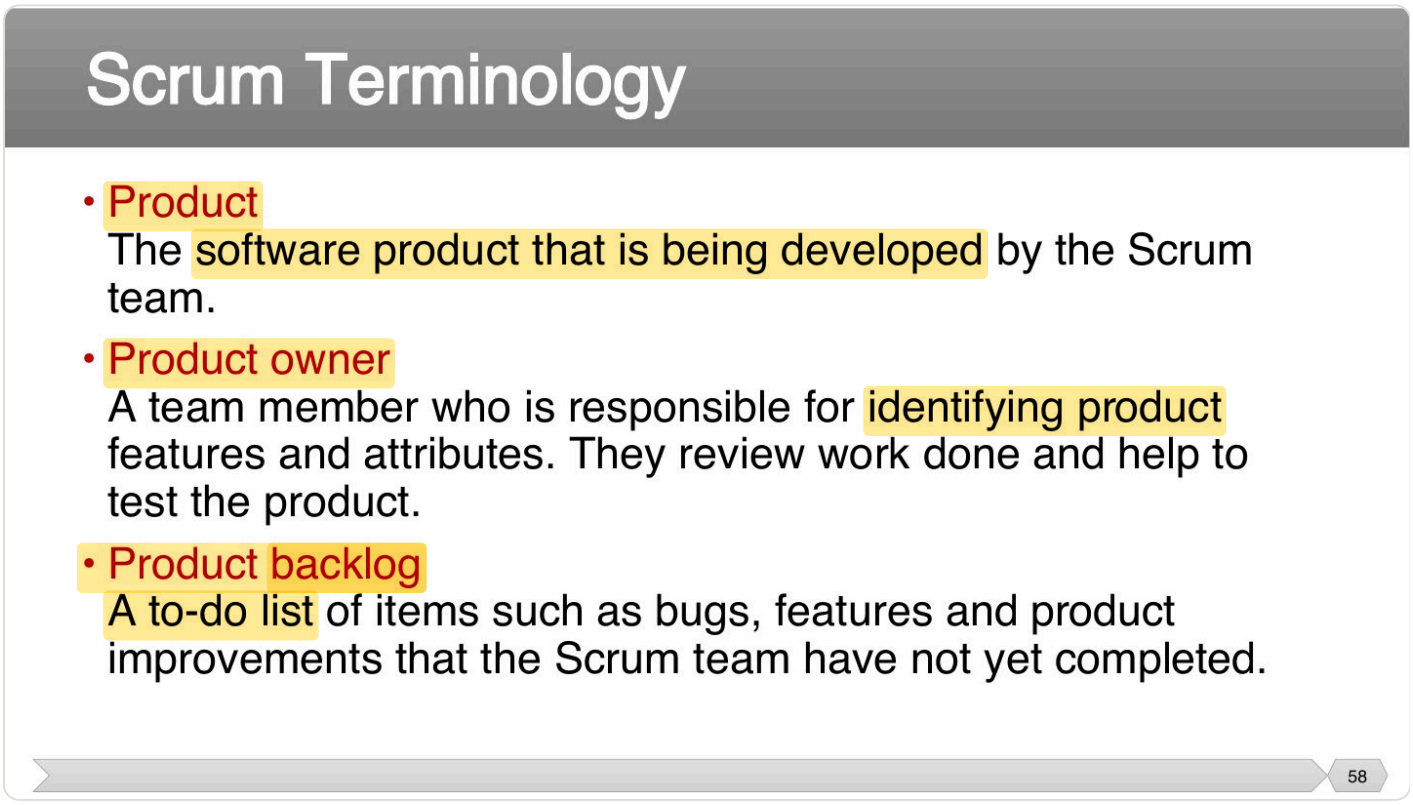
C **A daily team meeting**

> Progress is reviewed and the day's work agreed.

D A kind of test

> That's not it.

SLIDE 58 / 98 **Scrum Terminology**



NOTES

1. In Scrum, you focus on one **product** rather than a project with an end date. This keeps attention on what's being built over its whole life.
2. Someone has to decide what matters most. Having one **product owner** means developers get clear priorities, instead of conflicting requests from many people.
3. Everything that might get done lives in one place, so nothing is lost and it can be ordered. The **backlog** is the only source of work for the team.

EXAM ANGLE

Prediction

Q1

Who identifies product features and attributes?

A **The Product Owner**

> That's the role's definition.

B The ScrumMaster

> That's a coach for the method.

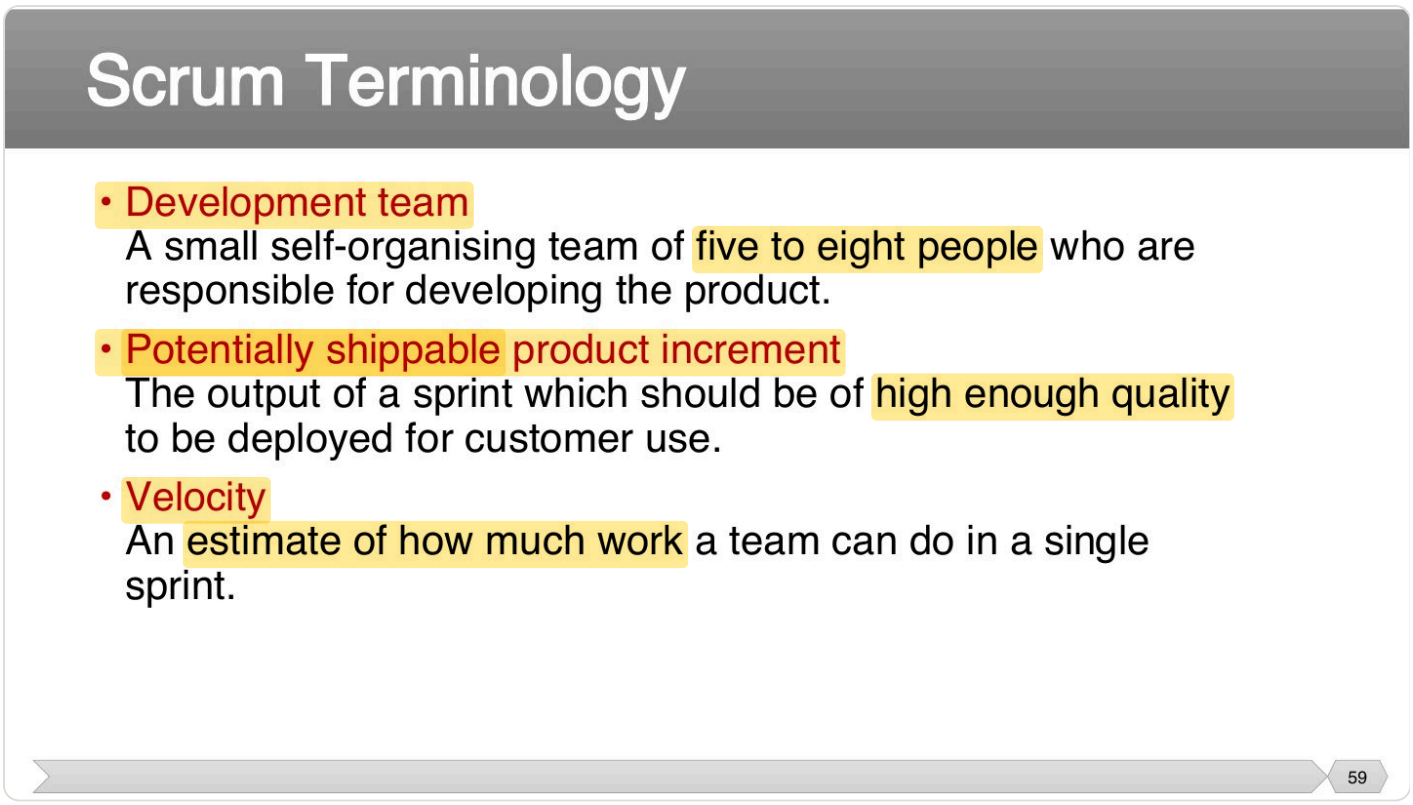
C An external auditor

> That isn't a Scrum role.

D The database

> That isn't a person.

 Scrum Terminology



NOTES

1. Small enough that everybody can talk directly, and large enough for a mix of skills. **Five to eight** keeps communication paths manageable.
2. This sets the bar: “mostly finished” doesn’t count. A **potentially shippable** result means there’s no hidden work left, so you could release it if you chose to.
3. **Velocity**: an **estimate of how much work** the team can do in one sprint, used to plan the next sprints.

EXAM ANGLE

Prediction Q1

What is velocity?

A The speed of the servers

That's not it.

B The number of defects

That's not it.

C **An estimate of how much work a team can do in a sprint**

It's used to plan sprints.

D The team's size

That's not it.

Prediction Q2

How large is a Scrum development team?

A Exactly two

That's a pair.

B 50 people

That's far too large.

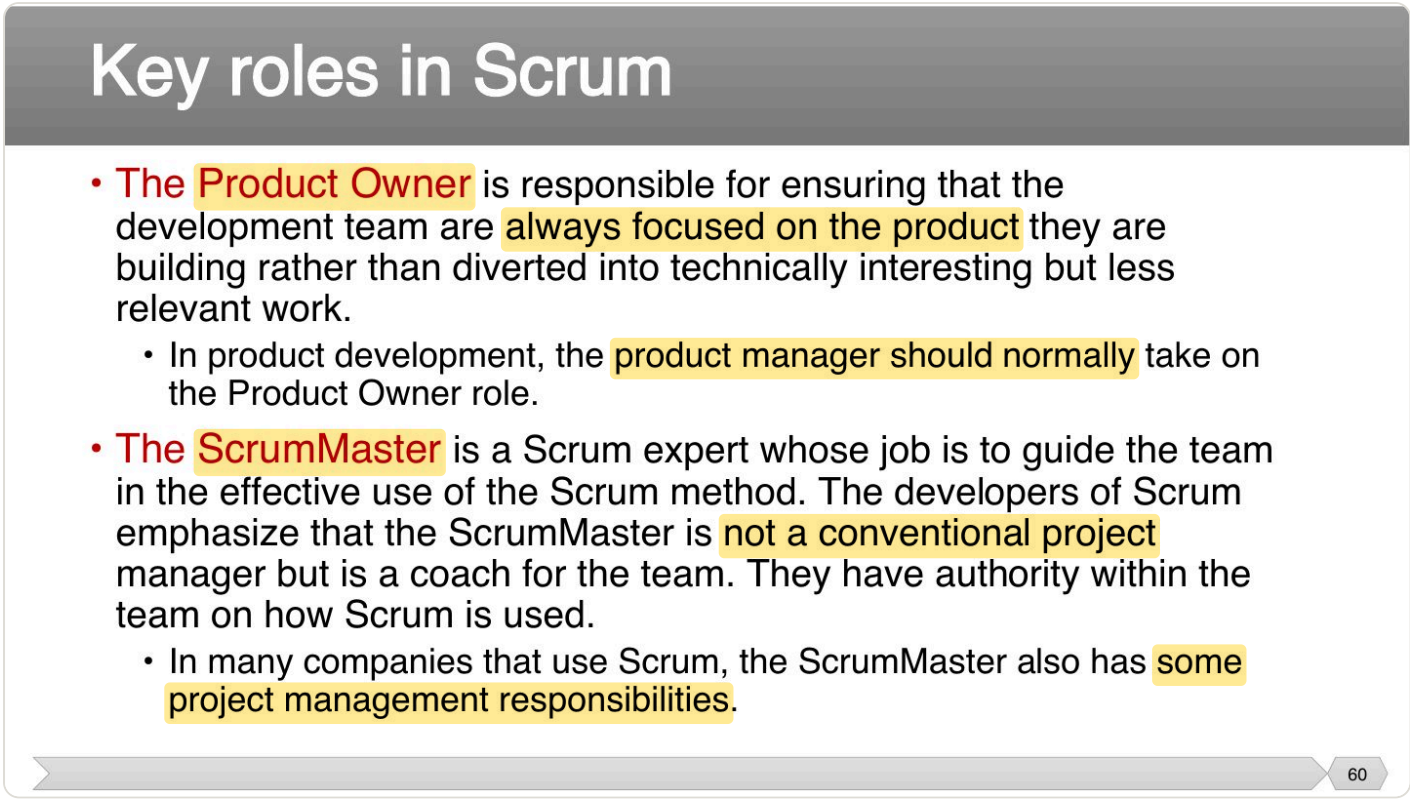
C One person

It's a team.

D **Five to eight people**

That's on the slide.

 Key roles in Scrum



NOTES

1. Engineers love interesting technical problems, which isn't always what customers need. The **Product Owner** keeps asking "does this help the product?" so the work stays useful.
2. Who's best placed to do this? Someone who already understands the market and the customers. So the **product manager** normally takes this role.
3. This trips people up: the **ScrumMaster** doesn't assign tasks or manage people. They're responsible for how the method is used, and they're there to coach.
4. Theory and practice differ. Someone still has to handle budgets and reports, so in many firms this person also takes on **project management** duties. Slide 70 lists these.

TIPS & TRICKS

- **PoST**: Product owner, ScrumMaster, Team.

EXAM ANGLE

Prediction Q1

Who keeps the team focused on the product instead of technically interesting but less relevant work?

A Sponsor

B Tester

C ScrumMaster

D **Product Owner**

Prediction Q2

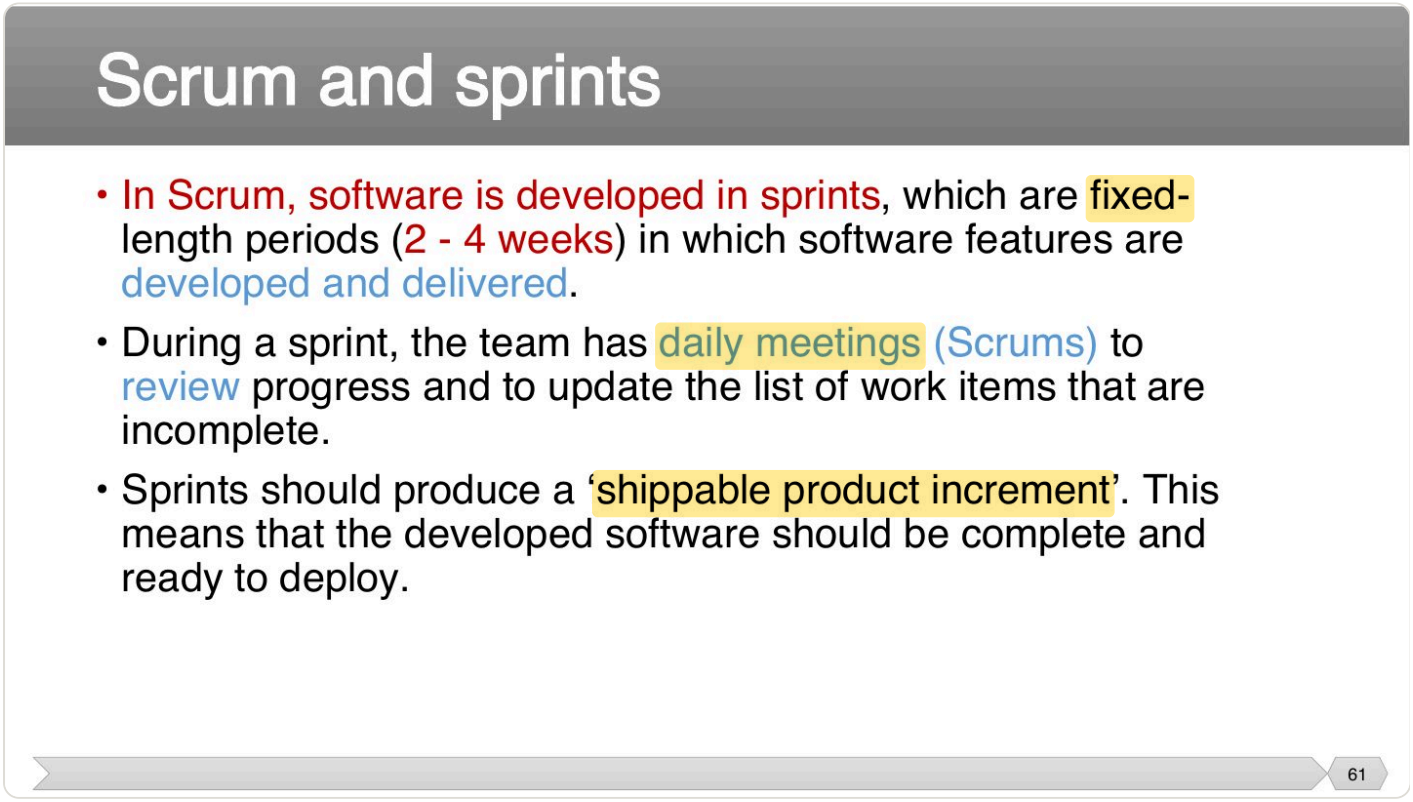
The ScrumMaster is a conventional project manager.

T True

F **False**

> The ScrumMaster is a coach for the team.

SLIDE 61 / 98 **Scrum and sprints**



NOTES

1. Since the length is fixed, a slipping task doesn't extend the **sprint**. Work gets dropped instead, and this makes delivery dates predictable.
2. A problem is visible within a day, and teammates can help right away. That's the value of **daily meetings**: short feedback, not status reporting to a boss.
3. "Shippable" stops the team from piling up unfinished work to fix later. Each sprint's **increment** really is complete, so you always have something you could release.

EXAM ANGLE

Prediction

Q1

Should a sprint get longer if work isn't finished?

A **No: sprints are fixed-length**

> Unfinished work goes back to the backlog.

B Only in December

> That isn't a thing.

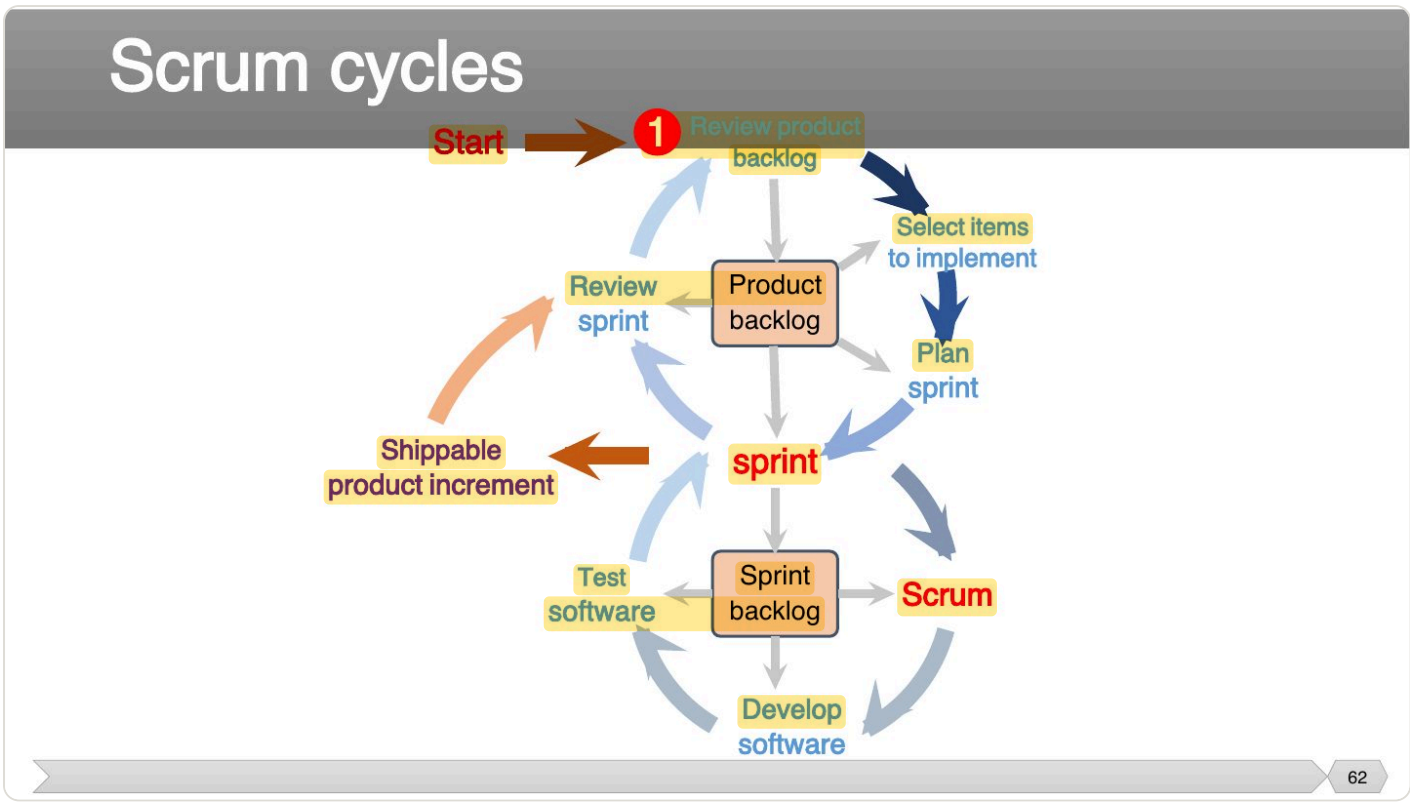
C Yes, always

> That breaks the timebox.

D Only for the ScrumMaster

> That isn't a thing.

Scrum cycles



NOTES

1. The team pulls the most valuable items it believes it can finish. Because it's chosen by the team, the **sprint plan** is more realistic than if someone else had set it.
2. During the sprint the team **develops and tests** software, with a daily Scrum.
3. A sprint's result is seen by stakeholders in the review, and what they say feeds the next one. Repeating this makes the **shippable increment** a regular way of learning.

EXAM ANGLE

Prediction Q1

Where do a sprint's items come from?

A The risk register

> That's for risks.

B **The product backlog**

> The team selects them for the sprint.

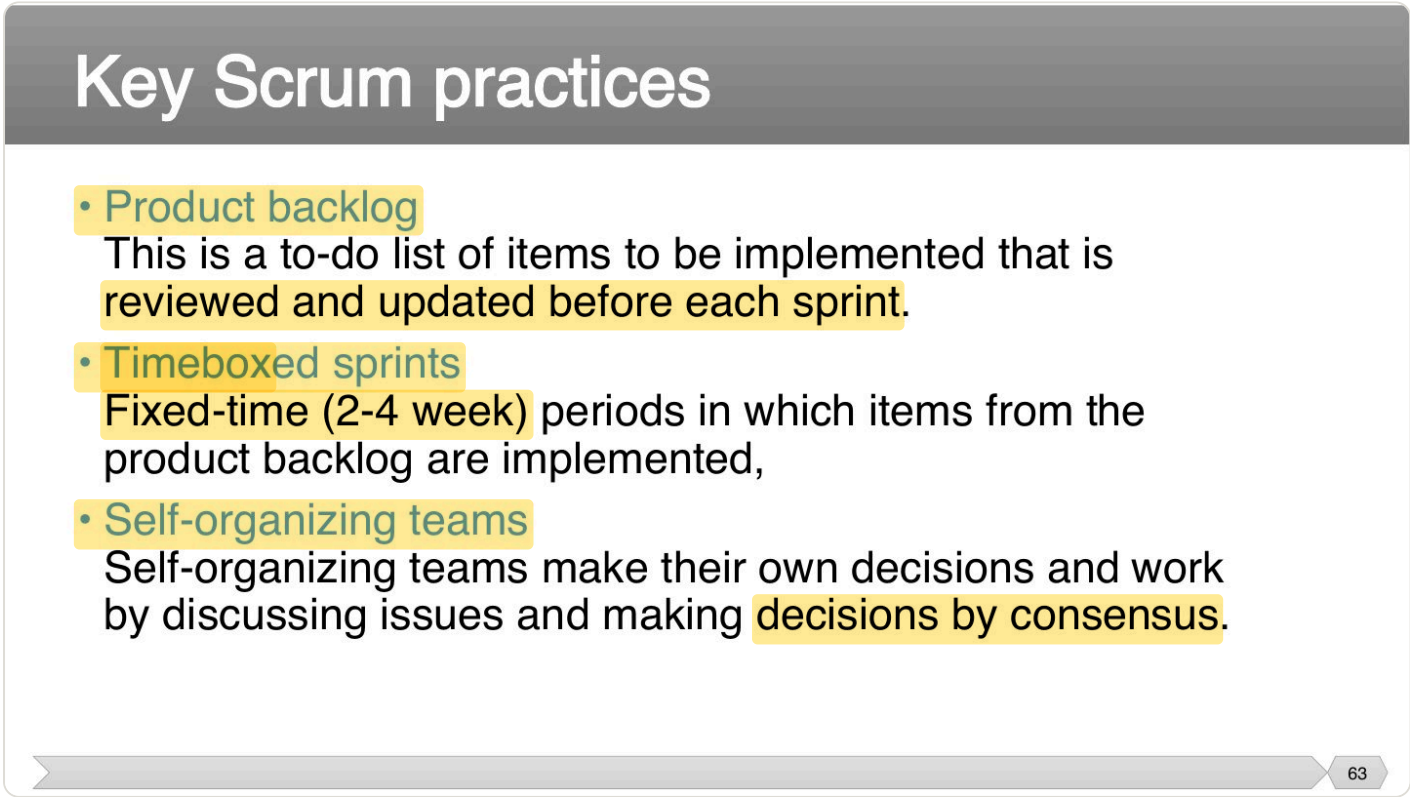
C The org chart

> That's for reporting lines.

D The contract's appendix

> That's not the source.

SLIDE 63 / 98 **Key Scrum practices**



NOTES

1 The list changes constantly as you learn. Updating it **before each sprint** means the team always works on what's most valuable right now.

2. Fixing the time forces prioritizing: you can't fit everything, so what's important has to go first. That's what makes a **timebox** powerful.

3. They're closest to the work, so they often decide best, and people commit to their own decisions. **Consensus** also spreads knowledge across the team.

TIPS & TRICKS

- **STeP**: Self-organizing teams, Timeboxed sprints, Product backlog.

EXAM ANGLE

Prediction Q1

How do self-organizing teams make decisions?

A By majority of managers

That's not how it works.

B **By discussion and consensus**

That's the slide's description.

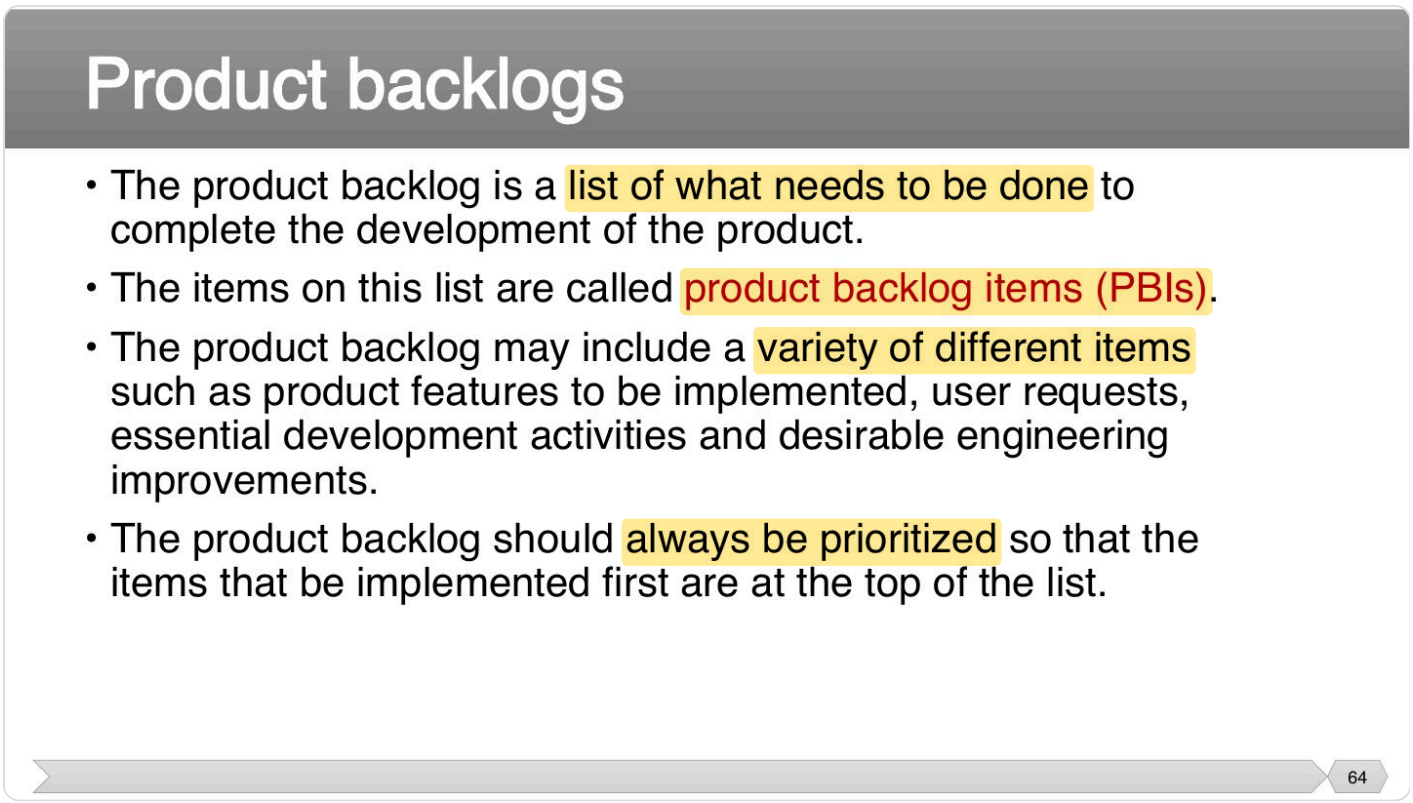
C By contract

That's not how it works.

D By the ScrumMaster's order

That's not how it works.

 Product backlogs



NOTES

1. If it's not on the list, it doesn't get done. This gives one place to see all remaining work, so the **product backlog** is what gets prioritized and discussed.
2. A **PBI** is just the name for a single entry. You'll see the short form in exam questions, so don't be thrown by it.
3. It's not only features! Engineering work needs a place too, or it never gets scheduled. Keeping all **varied** work on one list forces honest trade-offs.
4. Order matters, since the team always takes from the top. An **always prioritized** list means whoever is available starts on what's most valuable next.

EXAM ANGLE

Prediction Q1

Where should the next items to implement be in the backlog?

A At the top

It’s always prioritized.

B Anywhere

Order matters.

C At the bottom

That’s the opposite.

D In a separate list

There’s one list.

Prediction Q2

A product backlog can include engineering improvements as well as features.

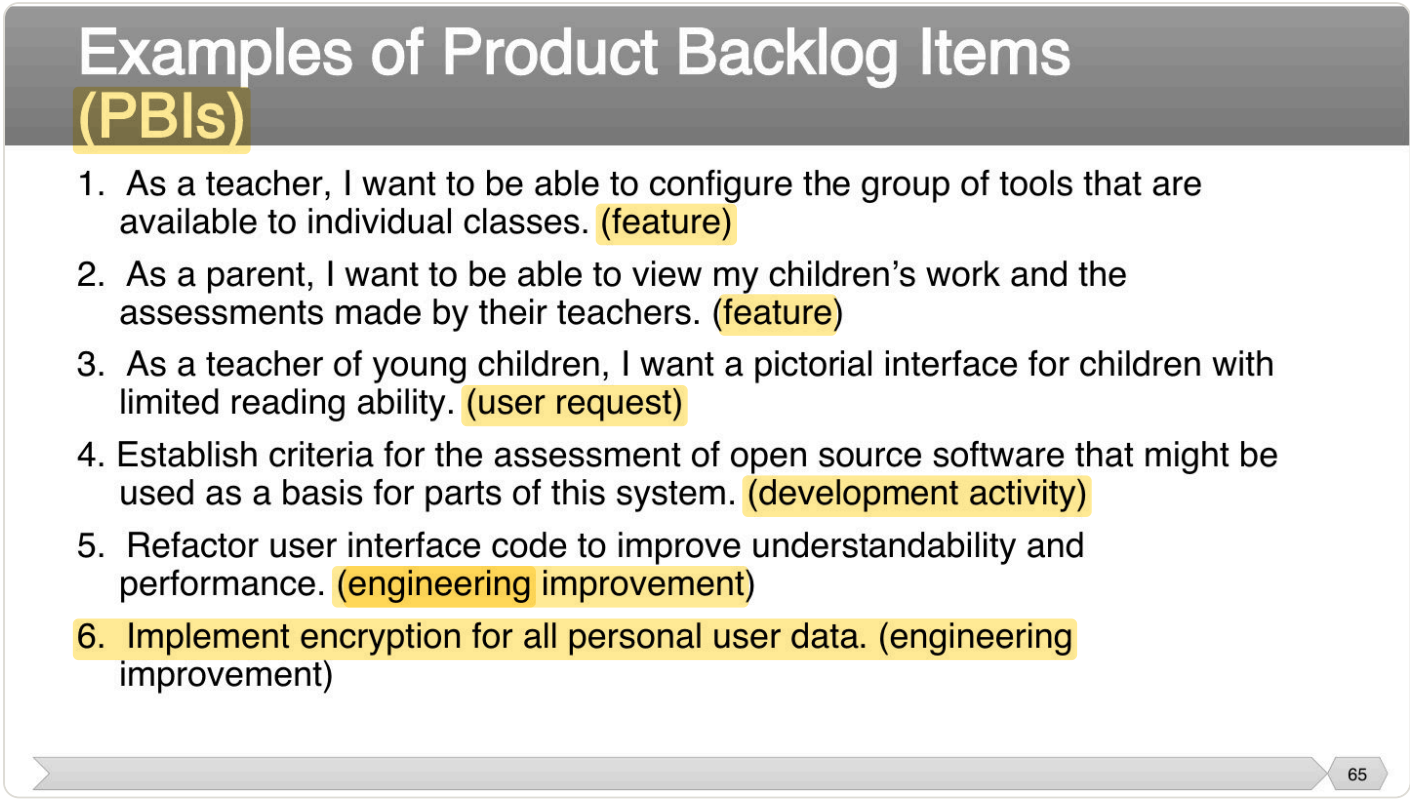
T True

It includes varied items.

F False

The slide lists these.

 Examples of Product Backlog Items



NOTES

1. Notice the form: who wants it and what they want. A **feature** is written from the user's point of view, so everyone can see who's helped by it.
2. This came directly from what users said they need. A **user request** may become a feature, but it starts as somebody's stated need.
3. No user will ever see this, and it's still needed. Choosing which open-source software to rely on is a **development activity** that keeps the team able to build.
4. Users won't notice this directly either, but it protects them or keeps the system healthy. An **engineering improvement** is easy to neglect if it isn't on the backlog.

EXAM ANGLE

Prediction Q1

"Refactor user interface code to improve performance" is which kind of PBI?

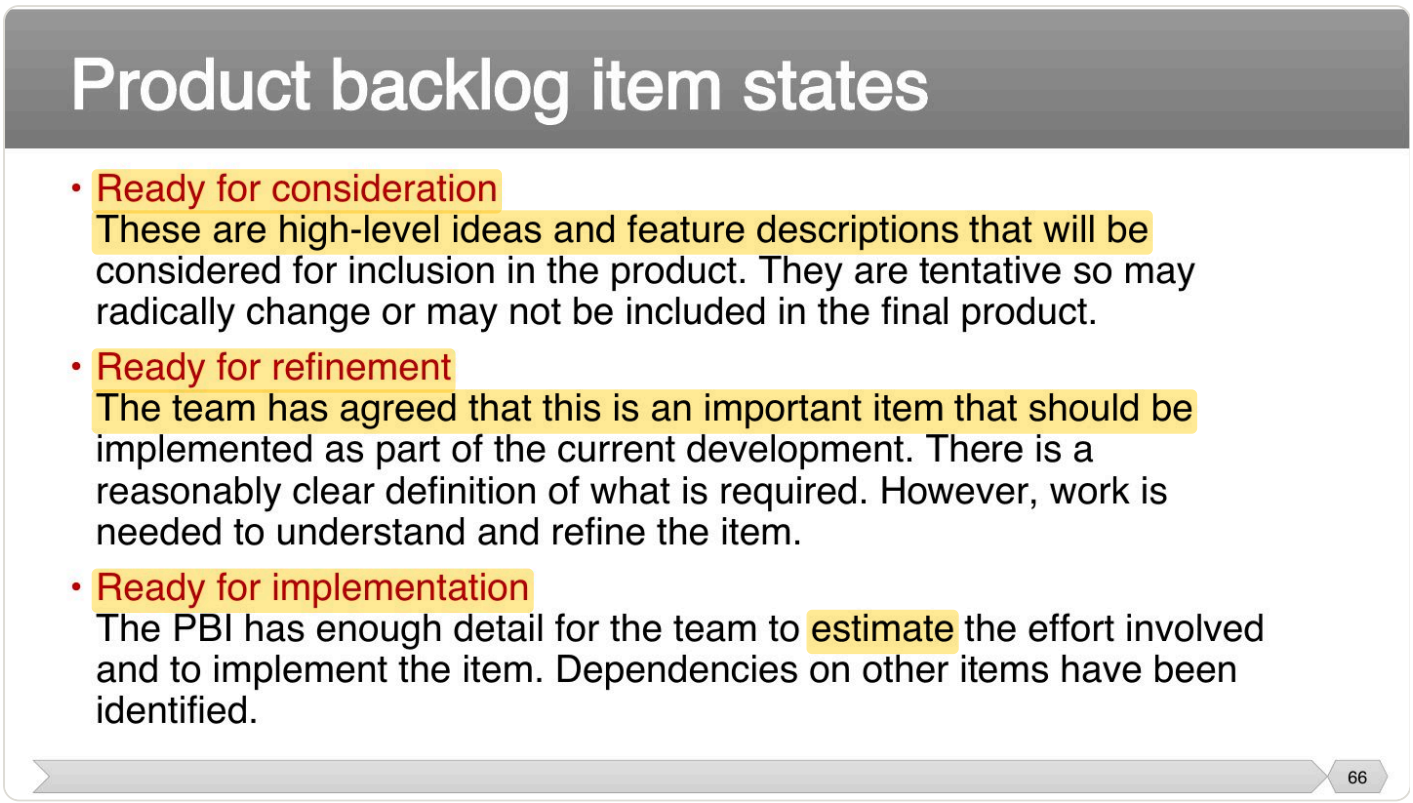
A Development activity

B **Engineering improvement**

C User request

D Feature

 Product backlog item states



NOTES

1. Don’t spend time detailing something that may never happen. At **consideration**, a line or two is enough, and dropping it costs nothing.
2. Now it’s worth investing some thought, as you’ve decided it matters. **Refinement** answers the open questions, so that it can be built.
3. The test for this state is whether the team could start tomorrow. If it’s possible to **estimate** it and nothing is blocking it, the item is ready.

TIPS & TRICKS

- **CRIb**, already in order: Consideration, Refinement, Implementation.

EXAM ANGLE

Prediction Q1

An item has a reasonably clear definition, but still needs work to understand it. Which state is it in?

A Ready for implementation

> That has enough detail to estimate.

B Ready for refinement

> That's how the slide describes it.

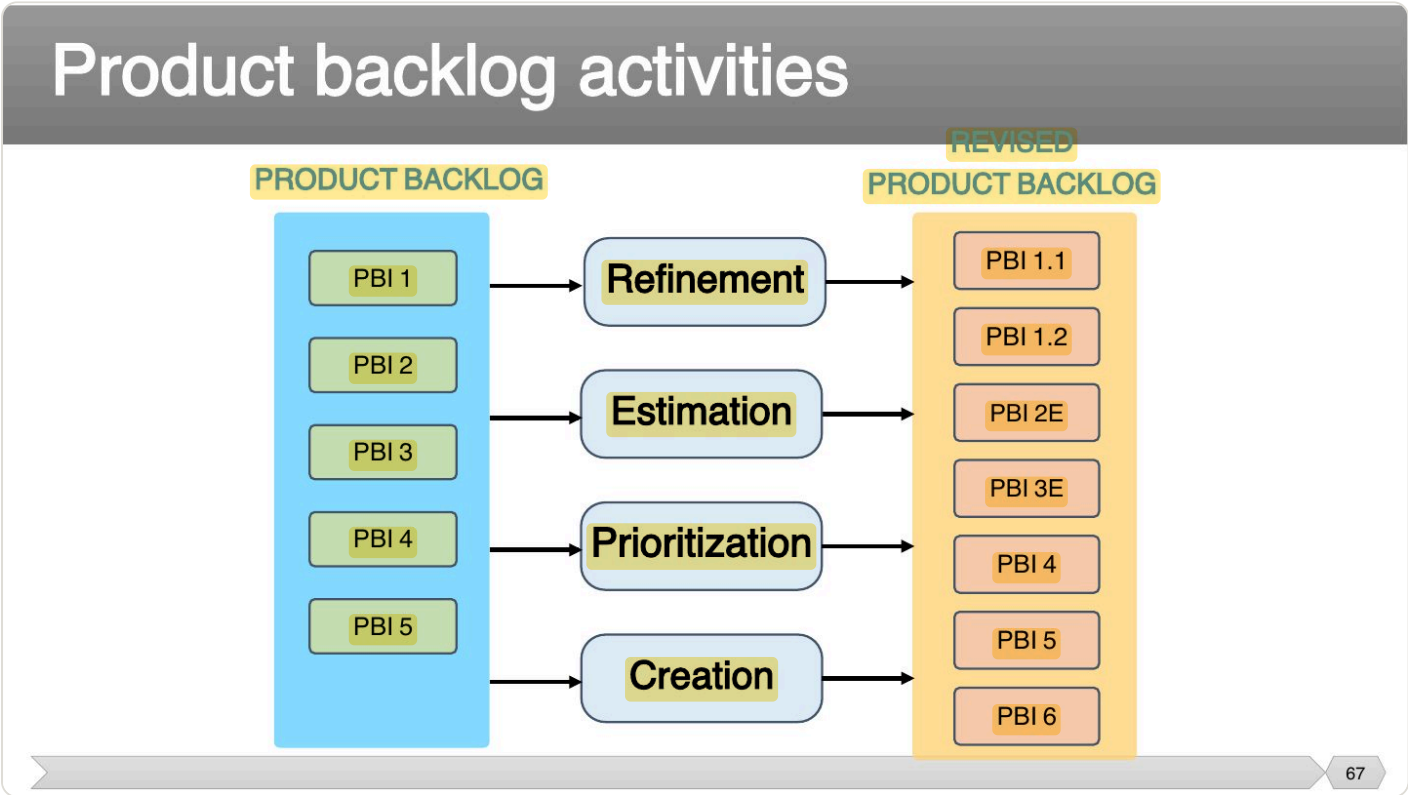
C Ready for consideration

> That's a tentative idea.

D Done

> It hasn't been built.

SLIDE 67 / 98 **Product backlog activities**



NOTES

1. **Refinement**: break PBIs into smaller, clearer items (PBI 1 becomes 1.1 and 1.2).
2. **Estimation**: estimate the effort for items (PBI 2E, 3E).

3. **Prioritization**: reorder the backlog by value.
4. **Creation**: add new items as they are discovered (PBI 6). The result is a revised backlog.

EXAM ANGLE

Prediction Q1

A PBI is broken into two smaller items. Which activity is this?

A Refinement

PBI 1 becomes 1.1 and 1.2.

B Creation

That adds new items.

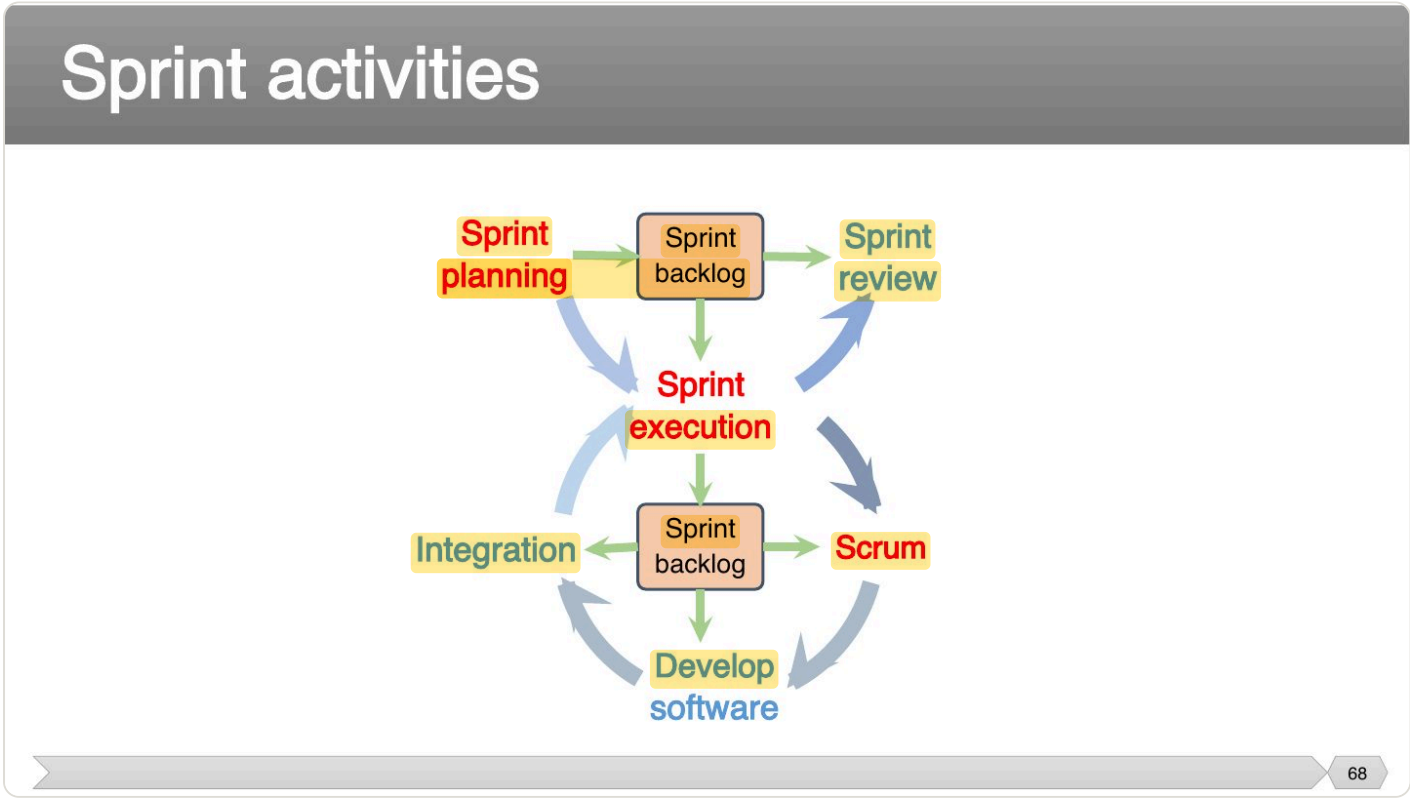
C Prioritization

That reorders.

D Estimation

That sets the effort.

SLIDE 68 / 98 **Sprint activities**



NOTES

1. Planning sets what this sprint is for, then **execution** is the work. The team checks in daily, so a problem can't hide for long.
2. Work is continuously **integrated**, and the **sprint review** closes the sprint.

EXAM ANGLE

Prediction Q1

What closes a sprint?

A The project's closure

> Only the sprint is ending.

B A contract signing

> That isn't a sprint activity.

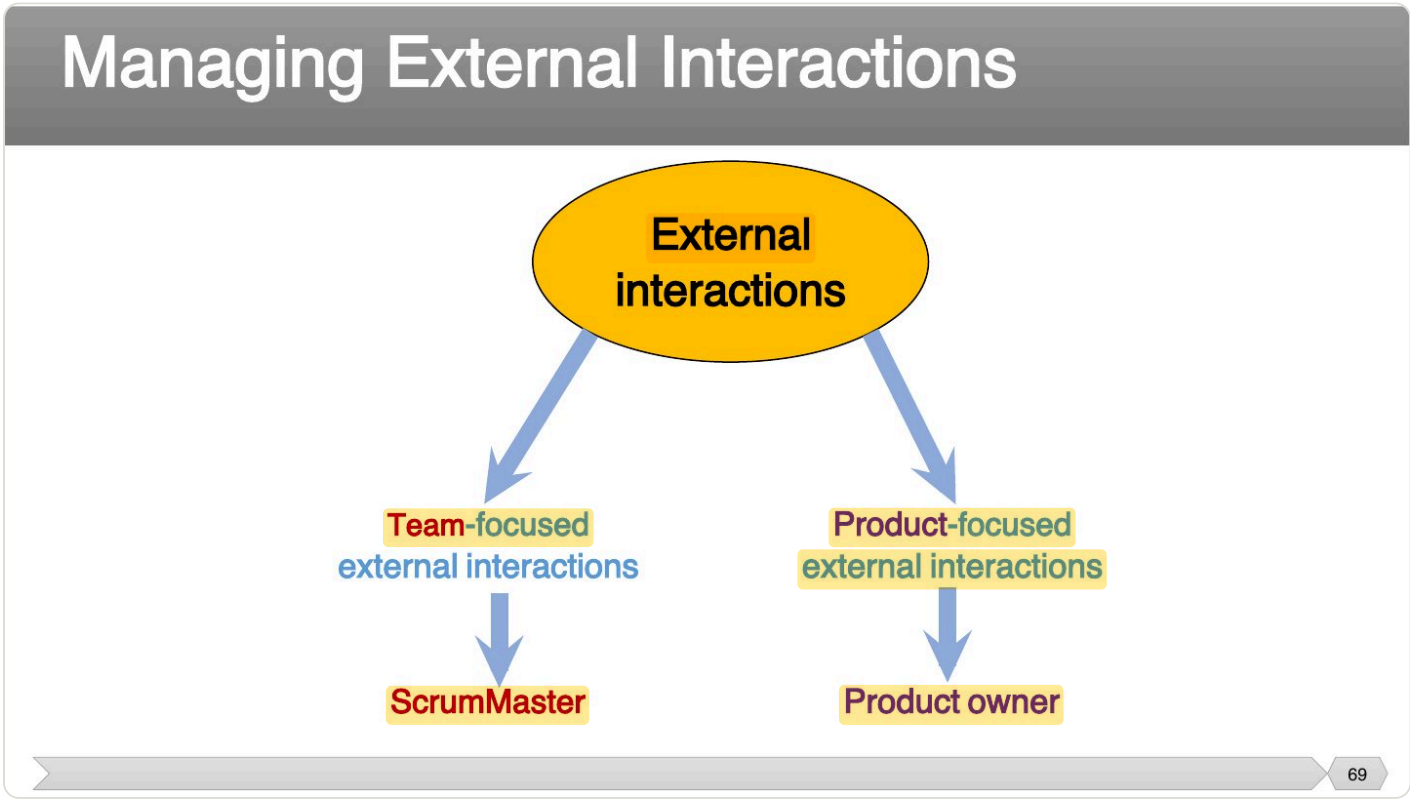
C **A sprint review**

> It's in the diagram.

D A phase gate

> That's predictive.

SLIDE 69 / 98 **Managing External Interactions**



NOTES

1. If everybody can interrupt developers, they stop getting work done. So the **ScrumMaster** handles requests about the team, like reporting and HR, and shields them.
2. **Product-focused** ones (customers, marketing, sales) go through the **Product Owner**. Either way, the developers are shielded from distractions.

EXAM ANGLE

Prediction

Q1

A customer has a product question. Who handles this external interaction?

A **The Product Owner**

> Product-focused interactions go here.

B Nobody

> Someone has to.

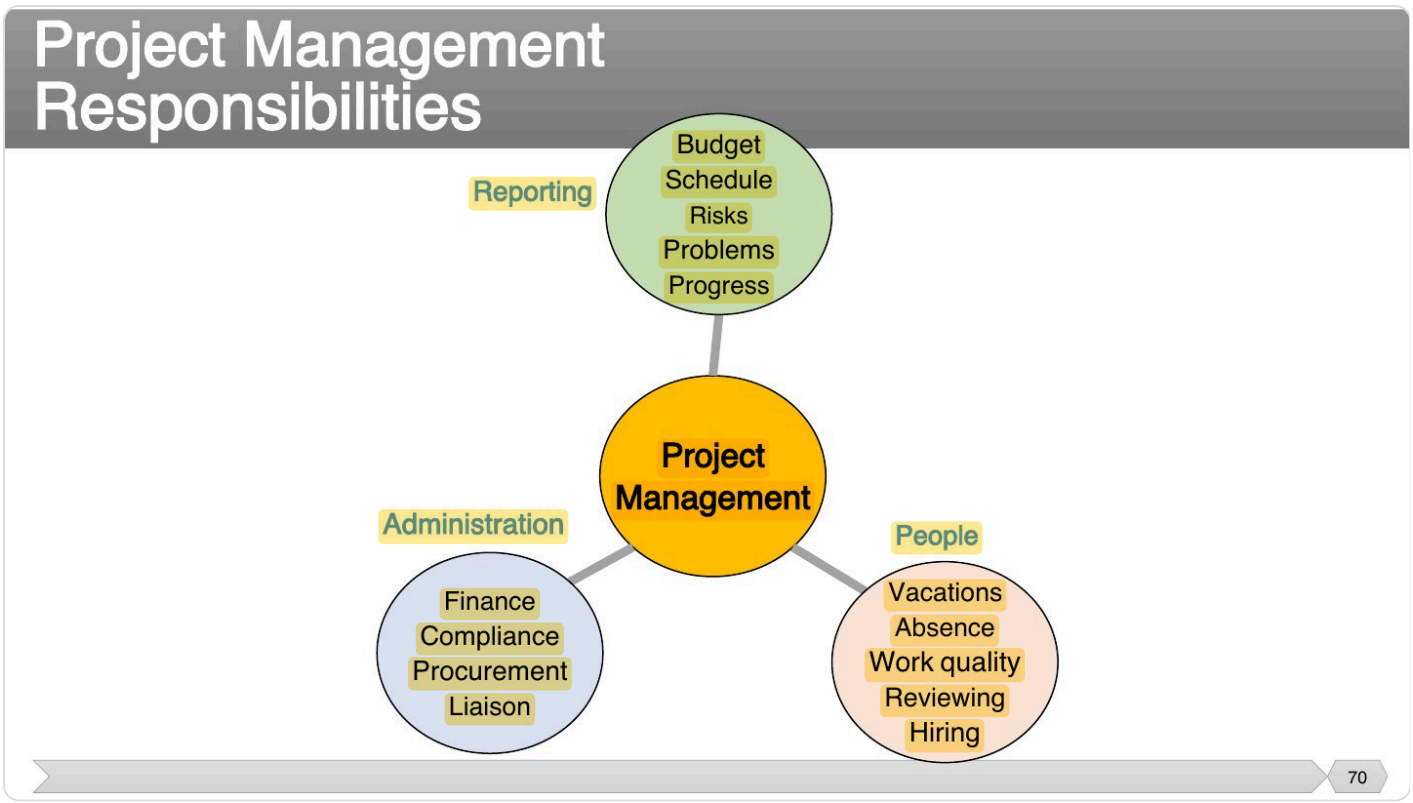
C The ScrumMaster

> Team-focused interactions go there.

D Every developer

> They’re being shielded.

 Responsibilities



NOTES

1. Scrum doesn't remove the business's need for information. Someone still has to **report** on money and progress, even if the team runs itself.
2. Rules still apply and bills still need paying, however agile the team is. So **administration** doesn't disappear, and Scrum doesn't say who handles it.
3. Scrum says little about holidays or hiring, but they still need handling. So **people** tasks often fall to a ScrumMaster or PM. Hence "ScrumMaster isn't a PM" is only partly true in practice.

EXAM ANGLE

Prediction Q1

Which responsibility is “Administration”?

A Compliance

> It’s in that group, with finance and procurement.

B Risks

> That’s under reporting.

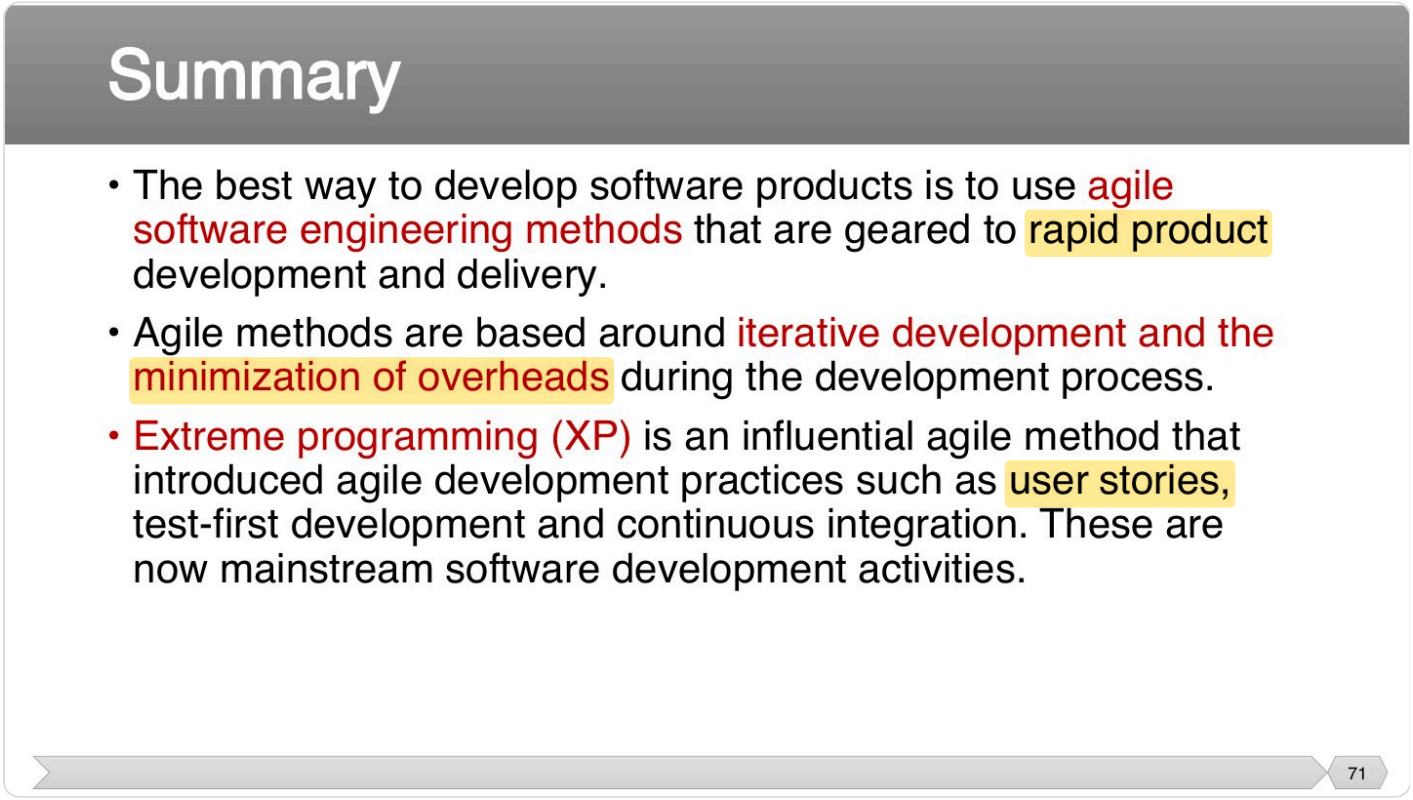
C Vacations

> That’s under people.

D Hiring

> That’s under people.

SLIDE 71 / 98 **Summary**



NOTES

1 “Best way” is the slide’s claim for products, where speed matters. For **rapid product** development, agile fits because it’s built for frequent delivery...

2 ...and how? By working in short cycles and cutting whatever doesn't help, so **minimal overhead** leaves more time for building...

3 You'll meet **XP's** practices even in teams that have never heard of XP, since **user stories** and continuous integration became standard everywhere.

EXAM ANGLE

Prediction Q1

Which practices did XP introduce that are now mainstream?

A Phase gates and audits

> Those are plan-driven.

B Gantt charts and WBS

> Those are older.

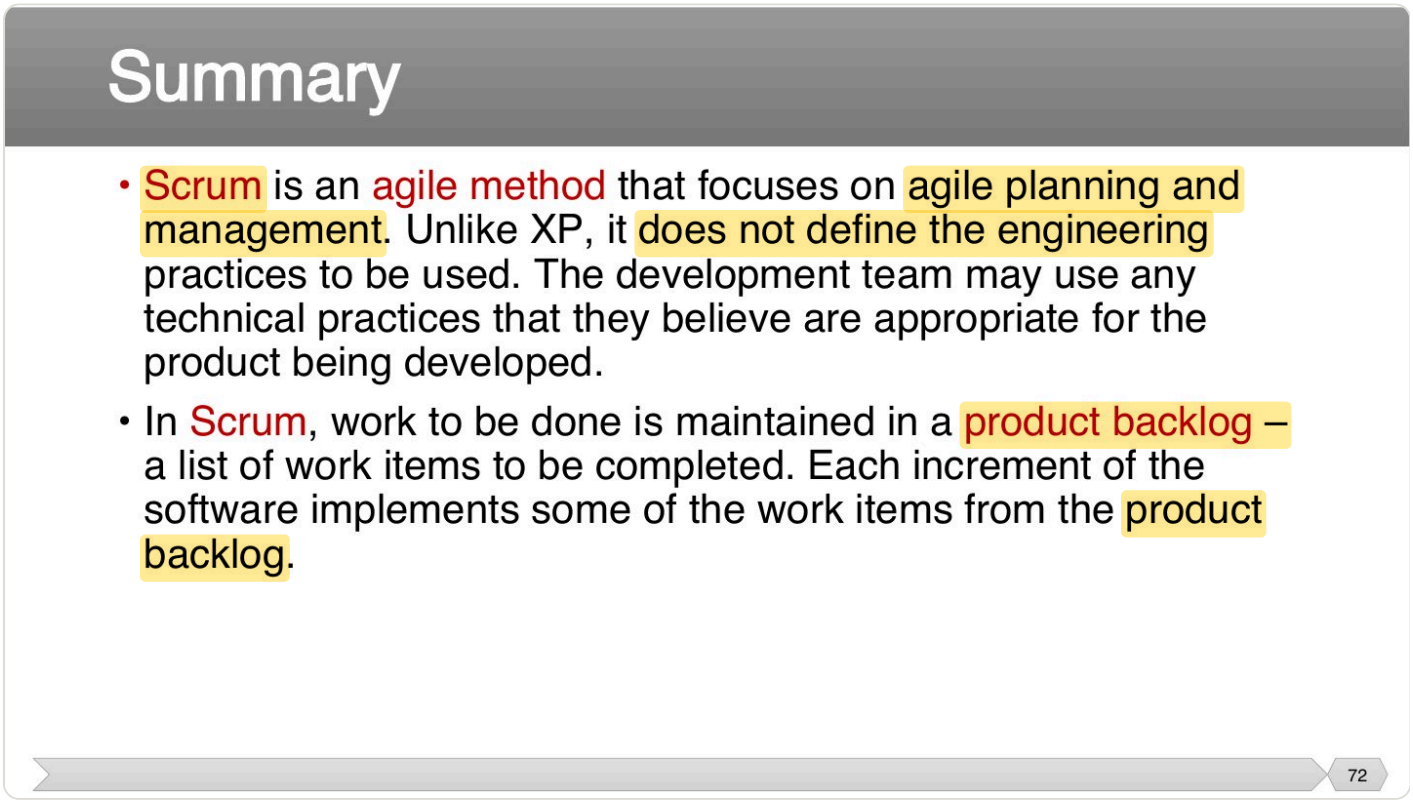
C **User stories, test-first development and continuous integration**

> That's the summary.

D Contracts and SOWs

> Those are procurement.

SLIDE 72 / 98 **Summary**



NOTES

1. Keep the two straight. XP says how to write the code, and **Scrum** says how to organize the work. So they fit together, and Scrum **does not define engineering practices**.
2. One list holds all the work, and each sprint takes some of it. The **product backlog** is what links long-term plans to daily work.

EXAM ANGLE

Prediction

Q1

How does Scrum differ from XP?

A **Scrum doesn't define the engineering practices**

> Teams pick their own.

B Scrum forbids testing

> It doesn't.

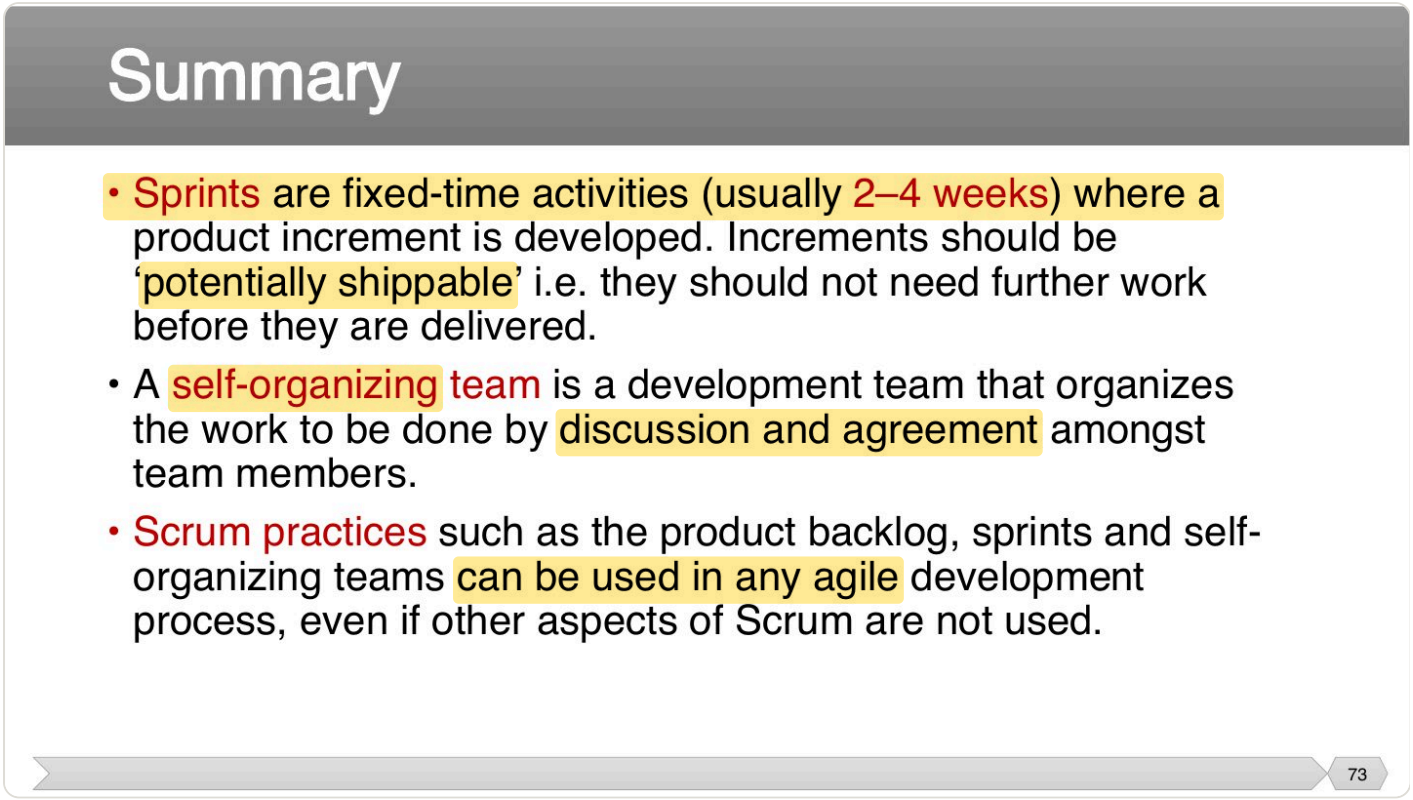
C Scrum requires pair programming

> It requires none.

D XP has no technical practices

> It has many.

 Summary



NOTES

1. No "we'll finish it next sprint". **Potentially shippable** means the increment is really done, so progress is real and not just reported.
2. The team decides together who does what. That only works with skilled people who trust each other, so **self-organizing** isn't something you can just announce.
3. You don't have to adopt all of Scrum. Many teams use just a backlog and sprints, which is why **any agile process** can borrow these.

EXAM ANGLE

Prediction

Q1

Can a team use a backlog without using all of Scrum?

A No, all or nothing

> The slide says otherwise.

B Only in hardware

> That isn't a requirement.

C **Yes, in any agile process**

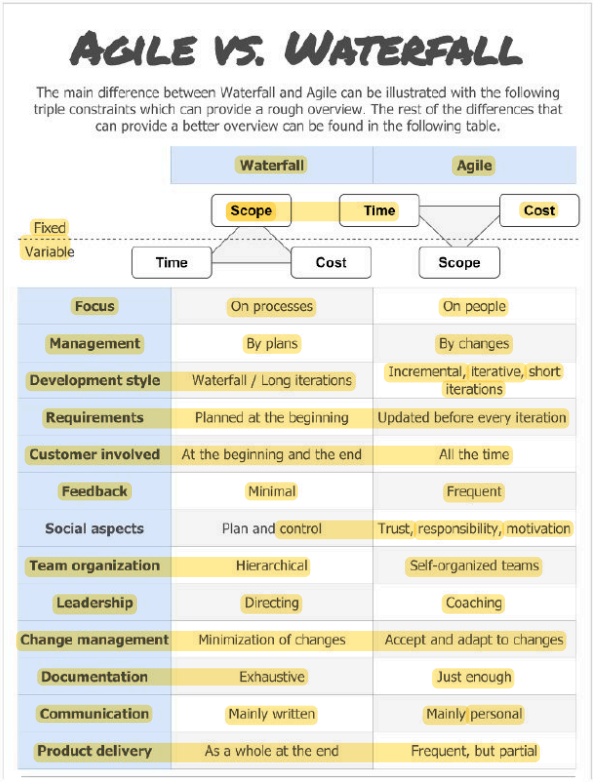
> The slide says these practices can be used on their own.

D Only with a contract

> That isn't a requirement.

SLIDE 74 / 98

Agile vs. Waterfall



NOTES

1. The key line in the triangle: **waterfall fixes scope** and lets time and cost vary; **agile fixes time and cost** (the timebox and team) and lets scope vary.

2 The table compares the rest: waterfall focuses **on processes**, manages by plans, plans requirements up front, involves the customer at the start and end, and delivers everything at the end; agile focuses **on people**, manages by change, updates requirements every iteration, involves the customer all the time, uses self-organized teams and coaching leadership, and delivers frequently in parts.

EXAM ANGLE

Prediction Q1

In agile, which of the triple constraints is typically allowed to vary?

A Scope

B None of them

C Cost

D Time

Past exam · Midterm 251 · V1 · Case Q2 (also V2) Q2

Given the evolving requirements (student usability, advisor requests, IT integration rules), which development approach (Agile, Waterfall, or Hybrid) would you recommend, and why would it be the best fit for this project?

Answer: Agile (Scrum) (0.5), because it handles the evolving requirements from students, advisors and IT, delivers iteratively so each sprint gets feedback, and integrates with DevOps pipelines for frequent tested releases (0.5). Justify it against the slide criteria: the requirements are reasonably clear but not stable, so waterfall's "clear, unambiguous, stable requirements" condition fails.

SLIDE 75 / 98 DevOps

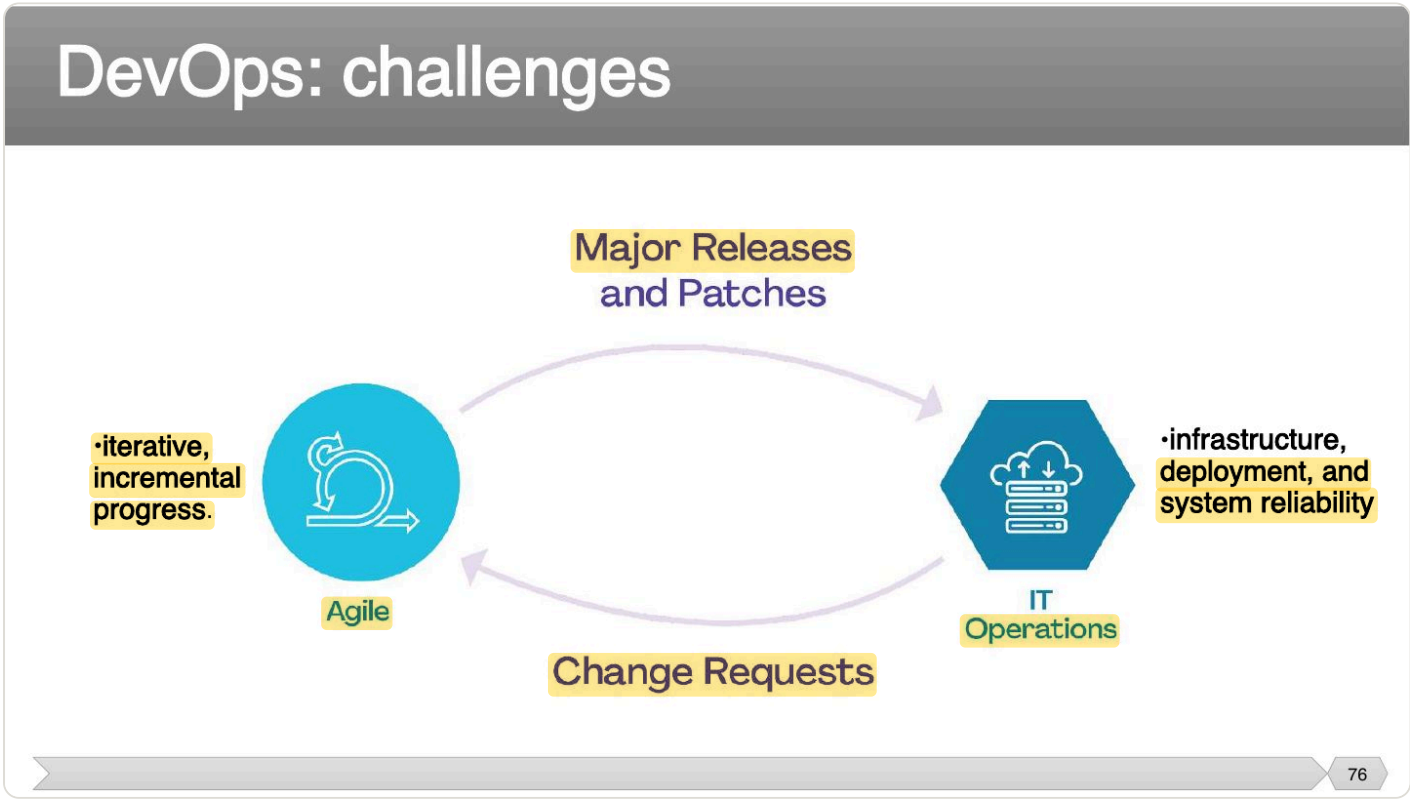
development
and operations.
DevOps

75

NOTES

1 Section divider: **DevOps** = **dev**elopment and **op**eration**s**.

SLIDE 76 / 98 DevOps: challenges



NOTES

1. The **DevOps gap**. Agile development delivers **iterative, incremental** progress and keeps sending change requests...
2. ...while IT operations care about infrastructure, deployment and **system reliability**, and resist frequent major releases and patches. Fast change and stability pull against each other.

EXAM ANGLE

Prediction

Q1

What is the DevOps gap?

A **Development wants frequent change, while operations wants stability**

> Those pull against each other.

B A budget shortfall

> That's not it.

C A lack of programmers

> That's not it.

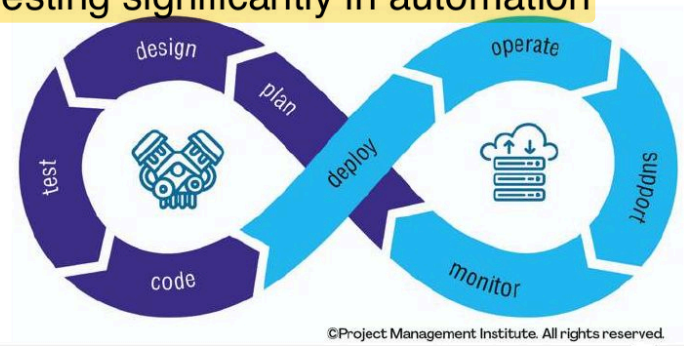
D The distance between offices

> That's not it.

DevOps

h

- When organizations address these challenges by removing the barriers that inhibit effective collaboration (operations <-> dev), they close the DevOps gap.
- Organizations do this by: adopting a mindset that promotes collaborative; learning-centric ways of working that are supported by agile practices; and very often investing significantly in automation
- On the development side they tend to adopt a continuous delivery (CD) approach and their operations efforts become streamlined and more automated.



©Project Management Institute. All rights reserved.

NOTES

1. The two sides often have opposed targets: one is judged on shipping, the other on stability. **Removing the barriers** starts with shared goals, not just talking more...
2. ...and a mindset of blame prevents this. If an outage leads to learning, rather than punishment, people share information. **Automation** then makes frequent releases safe to do...
3. Developers can ship often once **continuous delivery** is in place, and operations gets predictable, automated deployments. That way neither side has to give up its goal.

EXAM ANGLE

Prediction Q1

How do organizations close the DevOps gap?

A **Removing barriers to collaboration, and investing in automation**

> That's on the slide.

B Removing operations altogether

> They work together.

C Keeping the teams apart

> That's the problem.

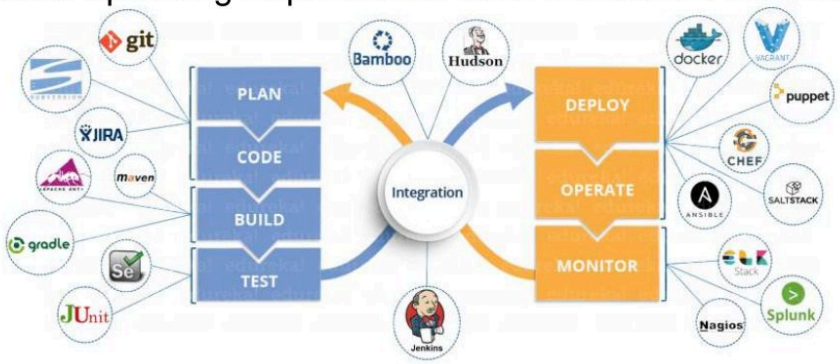
D Stopping all releases

> That's not the goal.

SLIDE 78 / 98 **DevOps**

DevOps

- DevOps is an approach to software development that extends the principles of agile development by integrating both the development and operations units of the IT team into one homogenous unit.
- This integration enables a more seamless flow between the traditionally siloed sides of the software development lifecycle and leads to fewer defects in production because operating requirements were considered from the outset of the project.



78

NOTES

1 Agile joined business and developers. **DevOps** extends that to the people who run the software. So it isn't a replacement for agile but an extension, which is why it's **one unit**.

2 When developers know how their software will run, they design for it from the start. That's why there are **fewer production defects**: operating needs aren't an afterthought.

EXAM ANGLE

Prediction Q1

DevOps extends agile principles by:

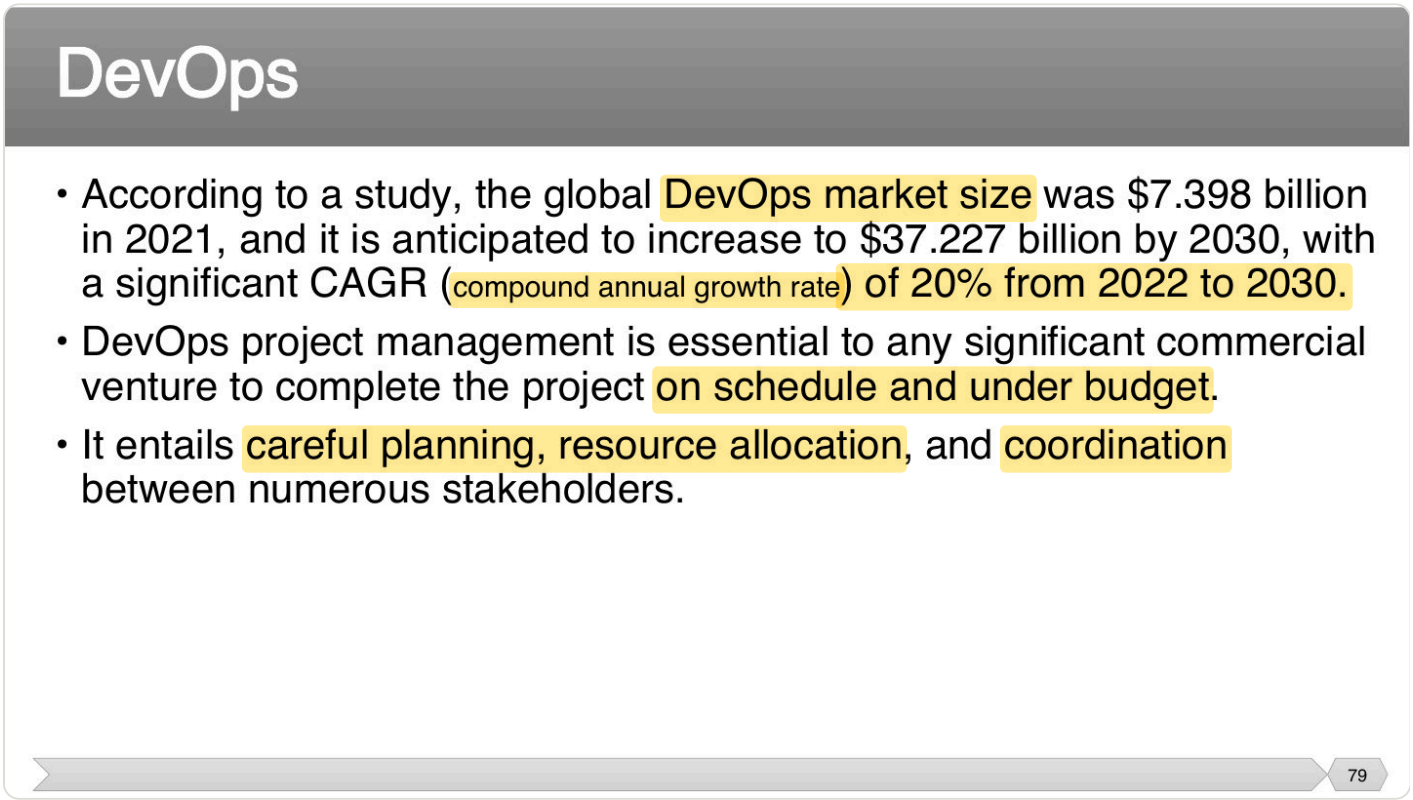
A Removing testing

B Separating developers from operations

C Fixing scope up front

D **Integrating development and operations into one unit**

SLIDE 79 / 98 **DevOps**



NOTES

1 The market figures are context only (not exam material): DevOps is growing fast.

2 Continuous delivery can make deadlines seem unimportant, but a business still has budgets and commitments. So DevOps still needs someone focused on **on schedule and under budget**...

3 ...and with many teams and tools involved, **coordination** becomes a real job. Fast-moving work needs this more, not less.

EXAM ANGLE

Prediction Q1

Does DevOps remove the need for planning?

A **No: it still needs planning, resource allocation and coordination**

> The slide says so.

B Only budgets remain

> It names planning too.

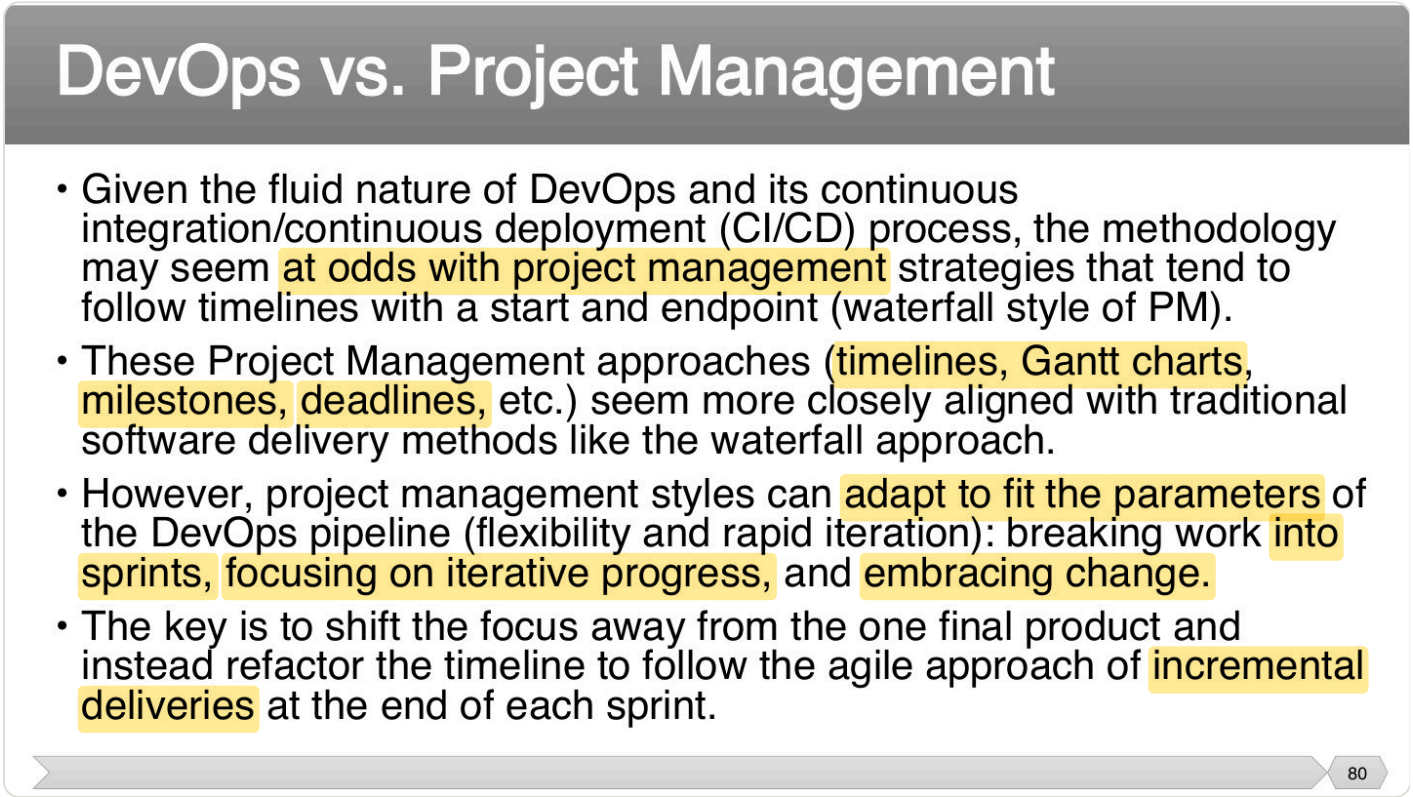
C Yes, entirely

> That's not what it says.

D Only customers plan

> That's not stated.

SLIDE 80 / 98 **DevOps vs. Project Management**



NOTES

1 Traditional PM assumes a project starts and ends, while DevOps never really stops. That's the apparent conflict: where's the finish line, if it seems **at odds with project management**?

2. A Gantt chart assumes that you know the tasks and order well in advance. **Timelines and milestones** suit predictable work, and they're harder to apply when everything keeps changing.

3. The tools change, while the job remains: know what's happening and help it go well. So PM can **adapt** by planning in short cycles.

4. Each sprint's output becomes the milestone, rather than one distant end date. So PMs plan **incremental deliveries**, which keeps the benefit of tracking without the fixed end.

EXAM ANGLE

Prediction Q1

How can project management adapt to DevOps?

A Insist on one final product only

> That's what it moves away from.

B **Plan incremental deliveries at the end of each sprint**

> That's the key on the slide.

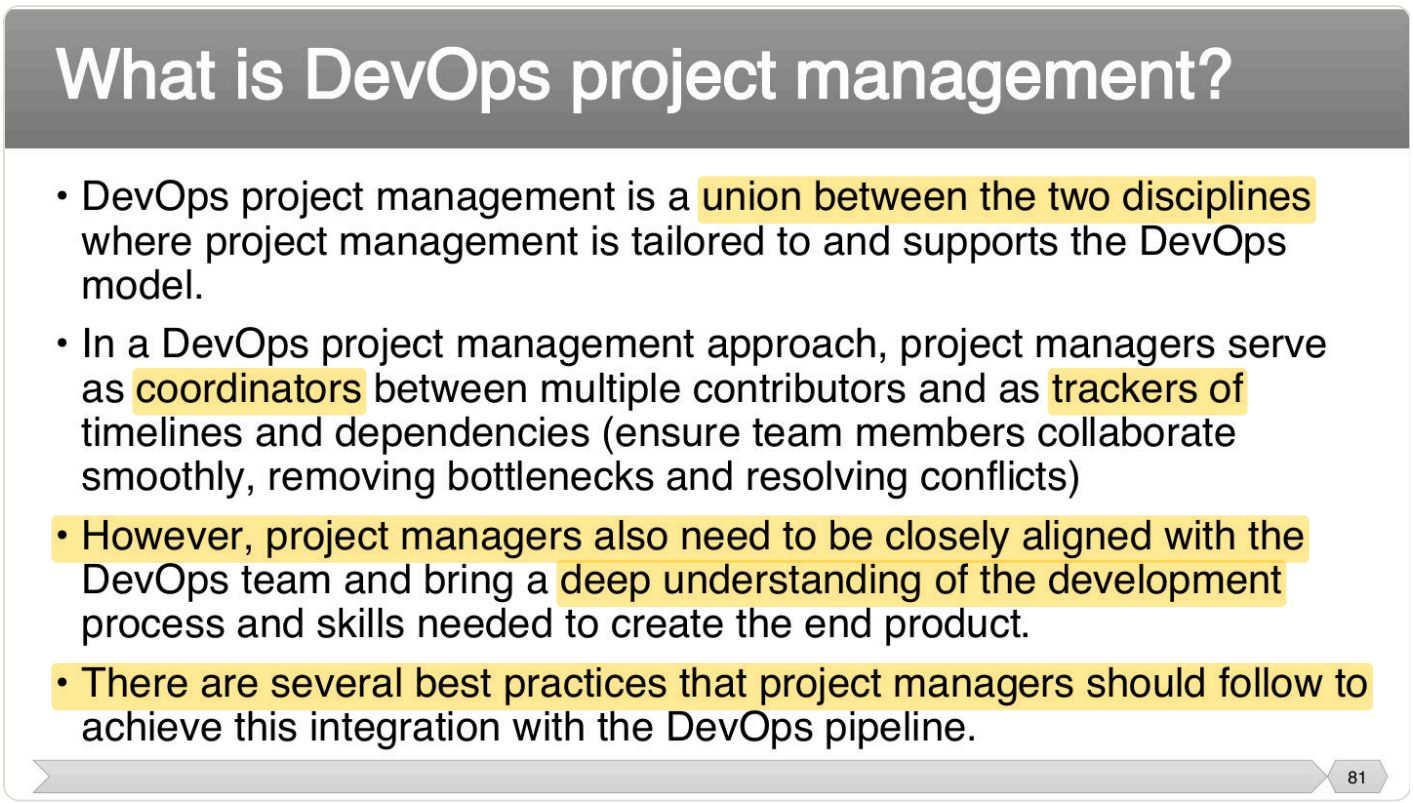
C Stop tracking work

> Tracking still matters.

D Ban Gantt charts by law

> That isn't suggested.

 What is DevOps project management?



NOTES

1. It's not choosing one over the other. **DevOps project management** keeps PM's planning and tracking, adjusted to fit a continuous way of working.
2. With so many people and tools, someone has to see the whole picture. A **coordinator** notices when one team's delay will affect another, and clears the way.
3. A PM who doesn't understand how code gets built and released can't spot problems or set sensible expectations. That's why a **deep understanding** of development is needed here.

EXAM ANGLE

Prediction Q1

What's a key role for PMs in DevOps project management?

A **Coordinating contributors and tracking timelines and dependencies**

> That's how the slide describes it.

B Only signing contracts

> That's far too narrow.

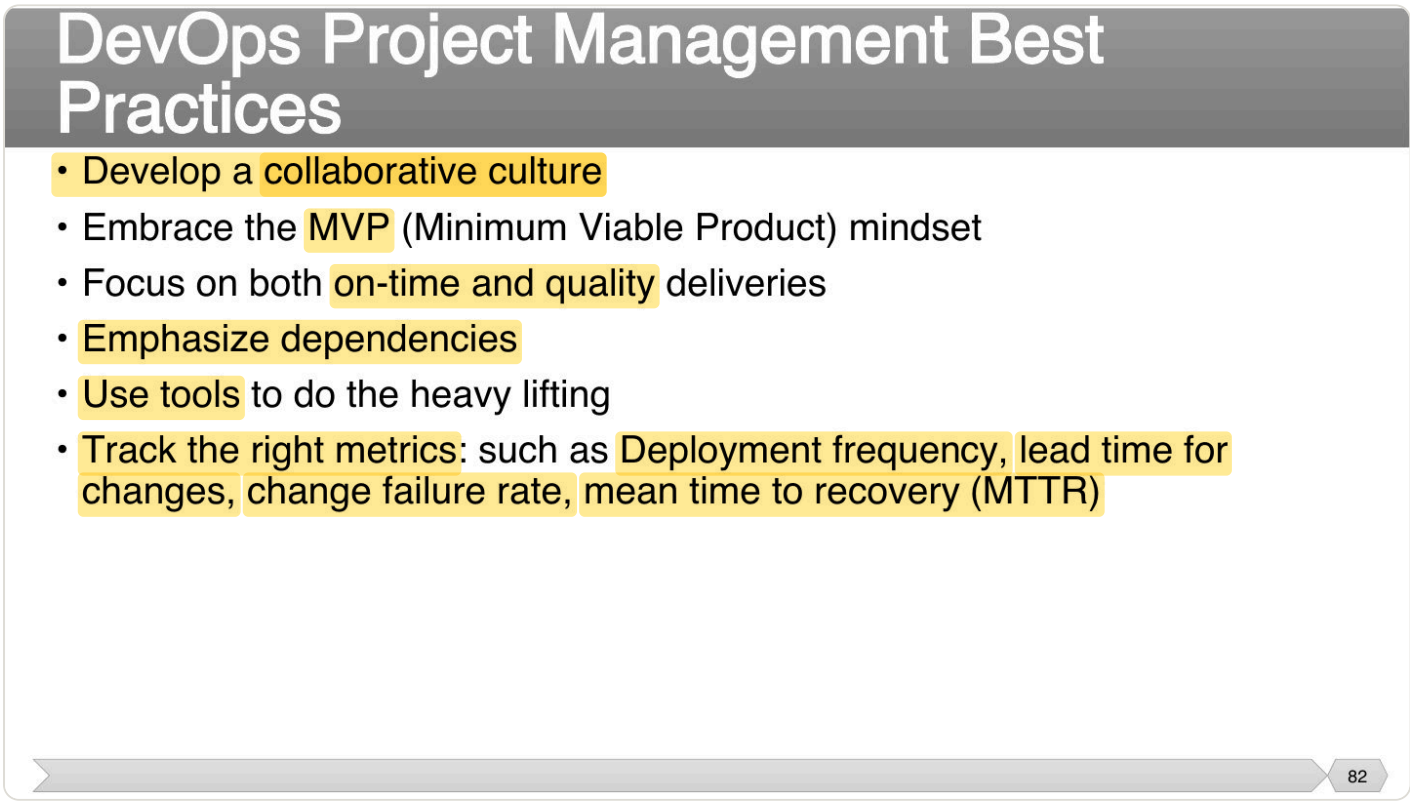
C Avoiding developers

> They must be closely aligned.

D Writing all the code

> That isn't the role.

SLIDE 82 / 98 **Practices**



NOTES

1 Everything else depends on this: tools can't fix teams that blame each other. So **collaborative culture** comes first on the list...

2. ...shipping the smallest useful thing keeps feedback coming, so the **MVP** mindset fights the urge to wait until it's perfect...
3. ...speed and quality aren't opposites if testing is built in, but you do need to watch both. So **on-time and quality** are listed together...
4. ...with frequent releases, a blocked task can hold up many others quickly. **Dependencies** need tracking all the time, rather than once at the start...
5. ...people shouldn't do by hand what a machine does reliably. **Tools** free up attention for the problems that actually need thinking...

TIPS & TRICKS

- **COMMiTteD**: Collaborative culture, On-time and quality, MVP mindset, Metrics, Tools, Dependencies.

EXAM ANGLE

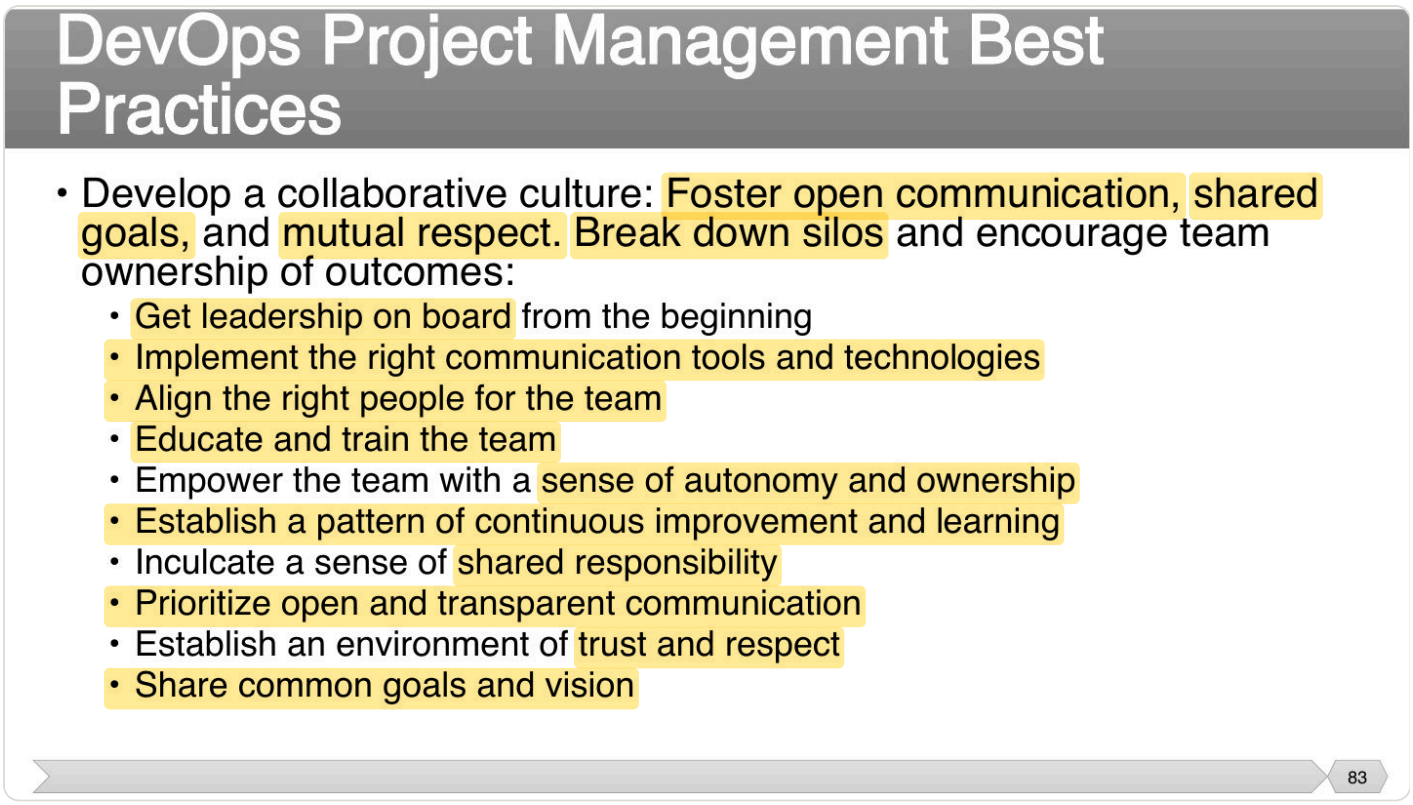
Past exam · Final 251 · Q1.4 — Q1

Give an example of a project management tool or technique that you might use to manage a complex software development project, and explain how it could help you ensure project success.
Answer: A Work Breakdown Structure: it decomposes the complex system into smaller deliverables and work packages that can be estimated, assigned and tracked, so nothing is forgotten, ownership is clear, and it is the base for the schedule, budget and risk analysis. Other valid answers, each with its benefit explained: a Gantt chart, CPM, a risk register, or a Scrum board with a burndown chart and Jira/CI dashboards for real-time visibility (slide 87).

Prediction — Q2

Which is one of the DevOps metrics listed on the slide?

- **A** **Mean time to recovery (MTTR)**
- **B** Number of meetings
- **C** Lines of code per day
- **D** Cost per printed page



NOTES

1. A **silo** is a team that protects its own area. DevOps can't work with these, because a problem belongs to everyone, so shared goals matter more than any tool.
2. Culture change fails without backing from the top, since old targets keep rewarding old behavior. So get **leadership on board** first. People need new skills too, so **train** them.
3. People take care of what they own. Giving them real **autonomy** means they fix what they see, and **shared responsibility** stops the old "not my problem" reaction.
4. Without **trust**, people hide mistakes, and you can't learn from what you can't see. A common vision gives everybody a reason to help one another.

EXAM ANGLE

Prediction Q1

Which practice helps build a collaborative culture?

A Getting leadership on board from the beginning

> It's on the slide's list.

B Keeping silos in place

> That's what to break.

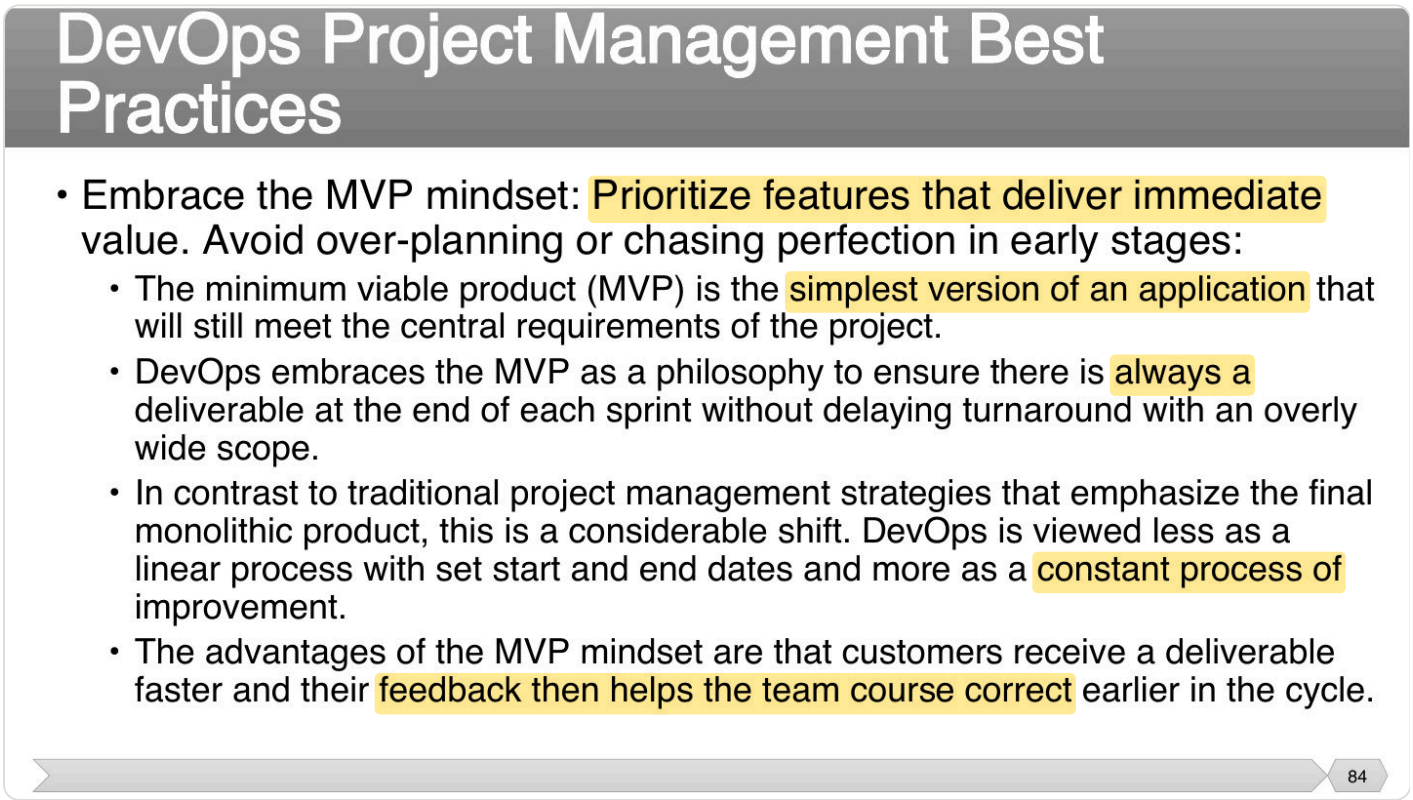
C Avoiding training

> Teams should be trained.

D Blaming individuals

> That destroys trust.

SLIDE 84 / 98 **Practices**



NOTES

1 Perfect is the enemy of shipped. **Immediate value** is what to prioritize, since early feedback may change what "perfect" even means.

2. Not a poor product, but the smallest useful one. **Simplest** still has to meet the central need: a car without air conditioning counts, four wheels and no engine doesn't.
3. A wide scope might mean nothing's ready when the sprint ends. Keeping scope small means there's **always a deliverable**, so stakeholders always see something real.
4. Traditional PM aims for one final result, while DevOps never quite "finishes". So it's a **constant process** of improving, which changes how success gets judged.
5. You'd rather hear "this isn't what we need" after two weeks than after a year. Early **feedback** makes changing course cheap.

EXAM ANGLE

Prediction Q1

What is an MVP?

A A prototype nobody can use

> It must still meet the core need.

B The most expensive version

> That isn't it.

C The final, perfect product

> That's the opposite.

D The simplest version that still meets the central requirements

> That's the definition.

DevOps Project Management Best Practices

- Focus on both on-time and quality deliveries: Set realistic timelines, monitor quality metrics (e.g., defect rates, rollback frequency), and ensure testing is embedded in the pipeline:
  - Project management places a heavy emphasis on meeting deadlines. While this is important, many of the metrics DevOps teams use to measure success focus on more qualitative factors to gauge if the customer is meeting their goals with the application.
  - This means that quick turnaround times are important. So is allocating enough time and resources to ensure the deliverable quality meets a high standard. Project managers need to weigh both these factors when deciding on the goals for a sprint and assigning tasks.
  - Part of an effective delegation of responsibilities is understanding the dependencies for each task.

NOTES

1. Teams under deadline pressure tend to skip quality, so measure it directly. **Rollback frequency** shows how often releases had to be undone, a clear sign of rushing.
2. Shipping on time means little if customers can't achieve anything with it. **Qualitative** measures ask if it actually helped, not just whether it arrived.
3. If you only reward speed, people cut corners, and if you only reward quality, nothing ships. So the PM sets sprint goals that allow **enough** time for both.
4. You can't hand out work sensibly if you don't know what must happen first. Knowing **dependencies** tells you who can start now and who's waiting.

EXAM ANGLE

Prediction Q1

Which is a quality metric from the slide?

A Team name

That's unrelated.

B Meeting count

That's not it.

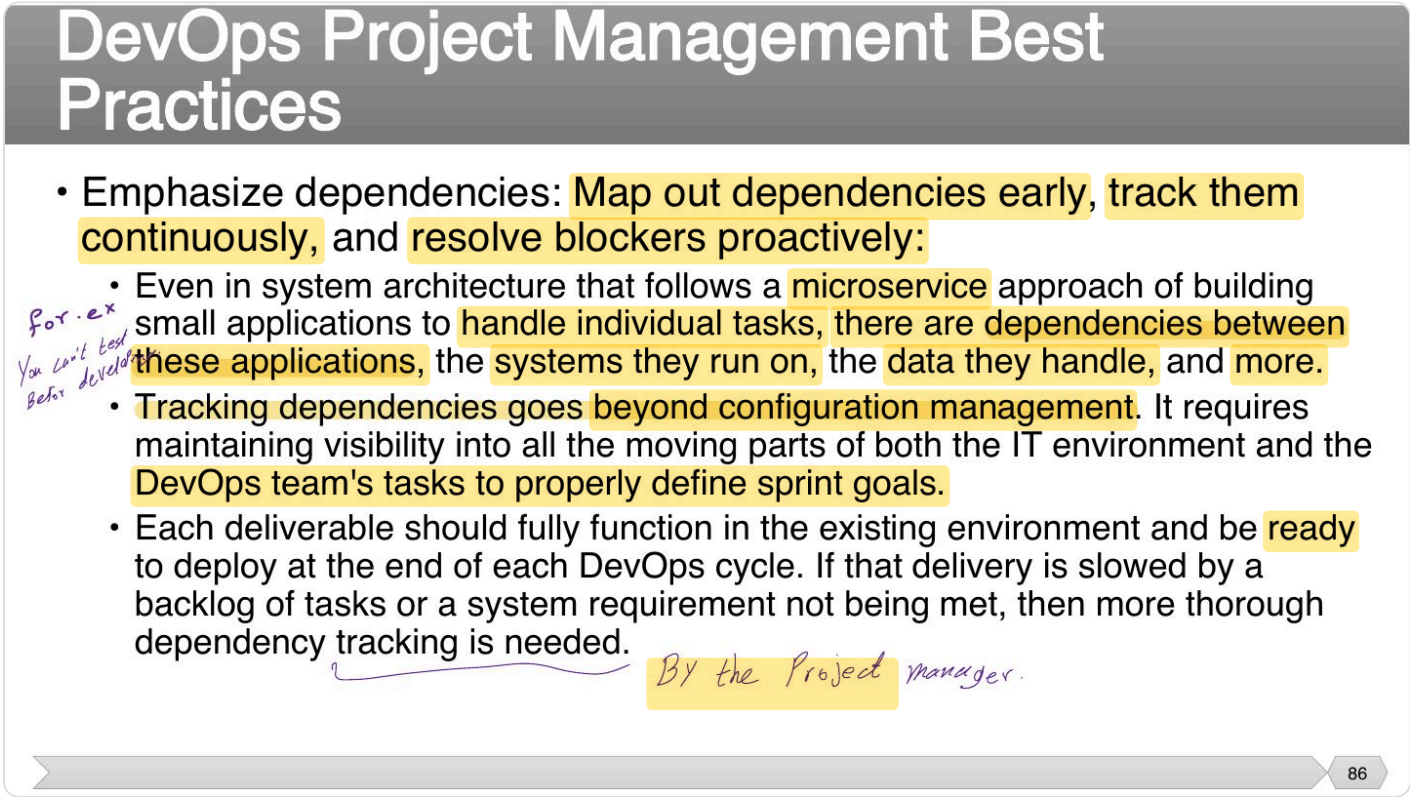
C Office size

That's unrelated.

D **Rollback frequency**

It's on the slide, with defect rates.

SLIDE 86 / 98 **Practices**



NOTES

1 A blocker found late can stop several people. **Map them early**, and you can reorder work or warn someone before it becomes a problem.

2. Don’t assume that small, separate services mean independent ones. Even with **microservices**, one service’s change can break others that rely on it.
3. Tracking versions isn’t enough. You also need to see what people are working on, as well as the systems, so **beyond configuration management** means both together.
4. If deliveries keep getting stuck, treat that as a signal. Each deliverable has to be **ready to deploy**, so repeated blocking means your dependency tracking needs work.

EXAM ANGLE

Prediction Q1

Using microservices removes all dependencies.

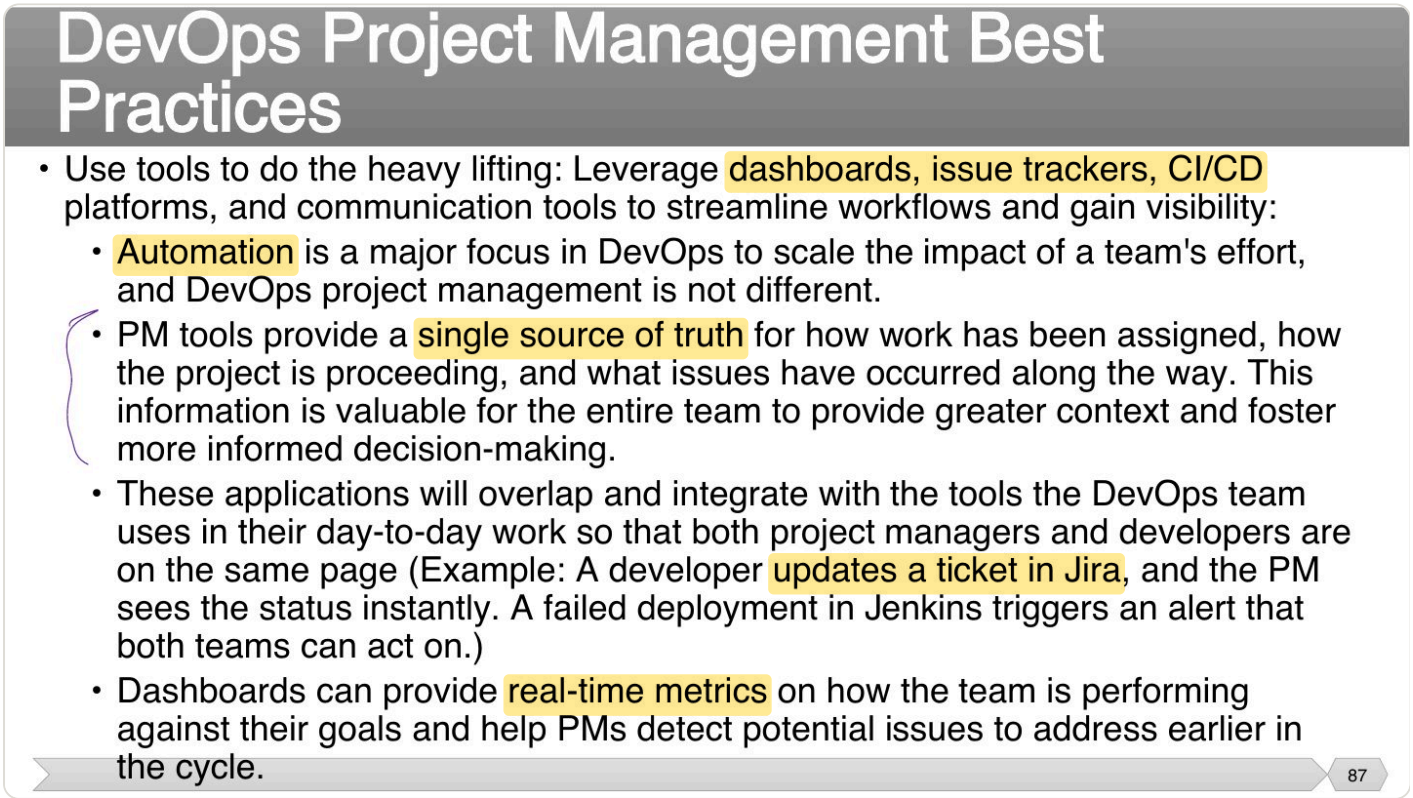
T True

> The slide says they remain.

F False

> There are still dependencies between services, systems and data.

SLIDE 87 / 98 **Practices**



NOTES

1. A team that does everything by hand can’t release often. **Automation** and shared dashboards let the same people do much more, and see what’s going on.

2. When different people keep different records, you waste time arguing about which is right. A **single source of truth** means everyone trusts the same information.

3. Nobody has to write a status report if the tools share it automatically. When a developer **updates a ticket**, the PM sees it at once, without interrupting anyone.

4. Checking last month's report means learning too late. **Real-time** metrics let a PM spot a problem today, and act while it's still small.

EXAM ANGLE

Prediction Q1

What does a single source of truth give the team?

A No need for developers

> That isn't the point.

B Hidden status

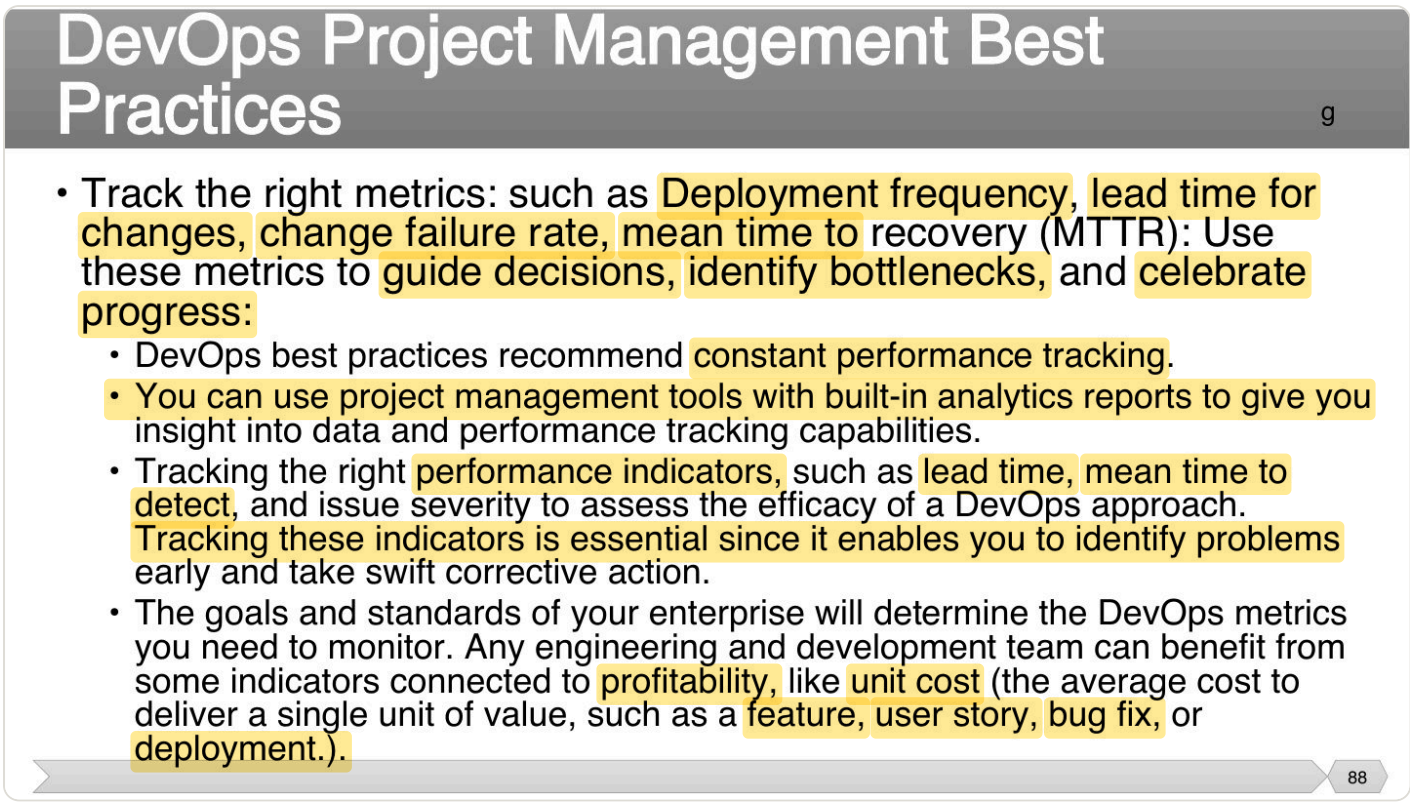
> It makes status visible.

C Several conflicting spreadsheets

> That's the problem it solves.

D One trusted record of work, progress and issues

> PM tools provide this.



NOTES

1. These four cover speed and stability together. Fast deployment that keeps breaking isn't success, so you measure **deployment frequency** and failure rate together. They're worth learning by name.

2. Things change fast here, so an occasional check isn't enough. **Constant tracking** comes built into tools, which makes it almost free to collect.

3. The sooner you know, the less damage. **Mean time to detect** asks how long a problem exists before anyone notices, while severity tells you what to fix first.

4. Measure what the business actually cares about. **Unit cost** links engineering to money, which makes it easier to justify spending on improvements.

EXAM ANGLE

Prediction Q1

Which is one of the slide's DevOps metrics?

A Length of meetings

> That's not on the list.

B **Change failure rate**

> It's on the slide.

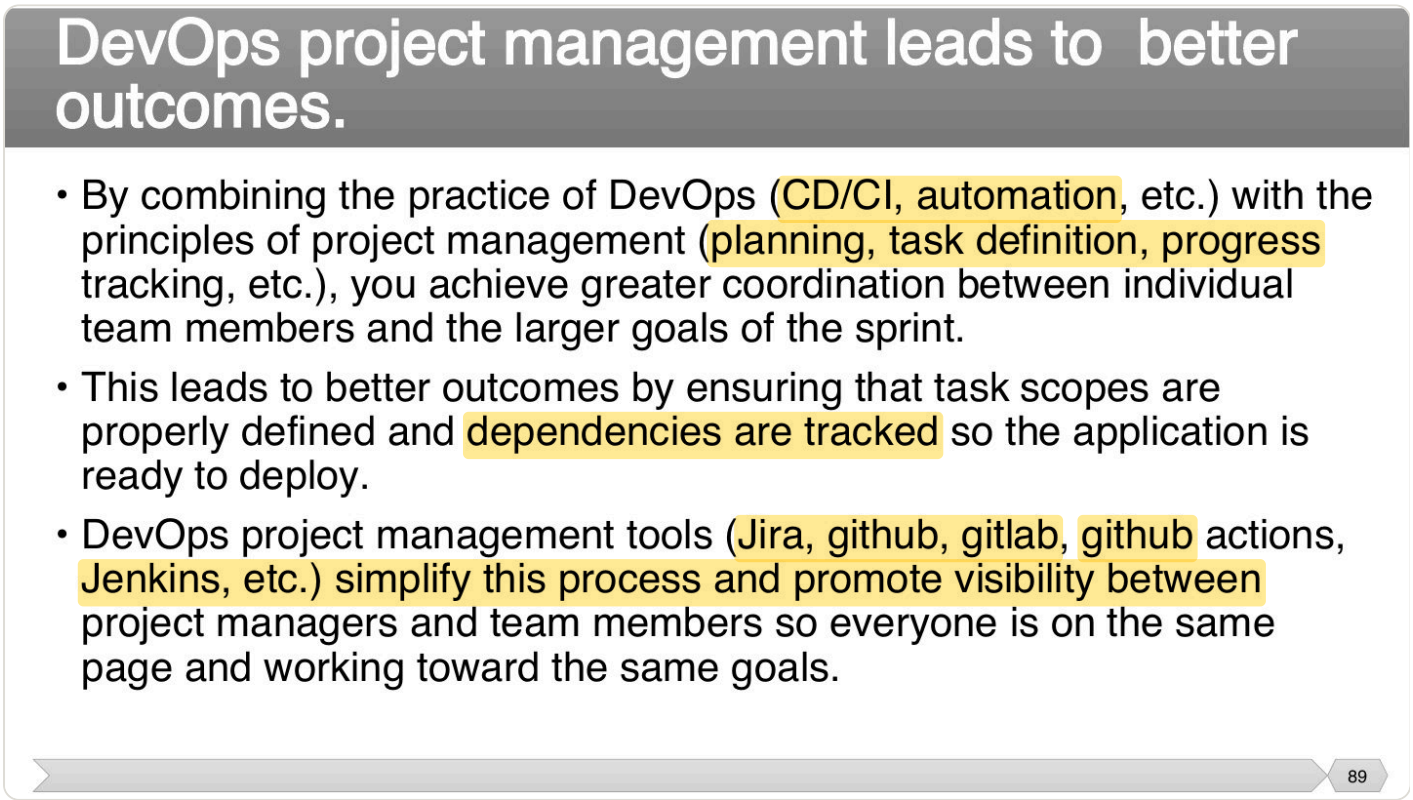
C Number of chairs

> That's unrelated.

D Pages of documentation

> That's not on the list.

SLIDE 89 / 98 **outcomes.**



NOTES

1 DevOps moves fast, while PM keeps everything pointed the same way. Combining **CI/CD** with planning means fast work is also the right work.

2. A task can be unclear, or depend on something unfinished, and that blocks deployment. Good planning avoids this, so **dependencies** get handled before they stop a release.

3. You don't need to know these tools in depth. They matter because they make the work visible to both sides, such as **Jira** for tasks and **GitHub** for code, without extra reporting.

EXAM ANGLE

Prediction Q1

How does combining DevOps with PM principles help?

A It removes dependencies

> They get tracked.

B It removes automation

> It keeps it.

C It ends task definition

> Tasks are well defined.

D **Individuals' work stays coordinated with the sprint's goals**

> That's the benefit given.

SLIDE 90 / 98 **DevOps tools (Jenkins)**



NOTES

1 **Jenkins**, an open-source automation server, at the center of a DevOps toolchain (Docker, Kubernetes, AWS, Azure, Terraform, Git). It automates build, test and deployment.

EXAM ANGLE

Prediction Q1

What does Jenkins automate?

A Contract signing

> That's unrelated.

B Hiring

> That's unrelated.

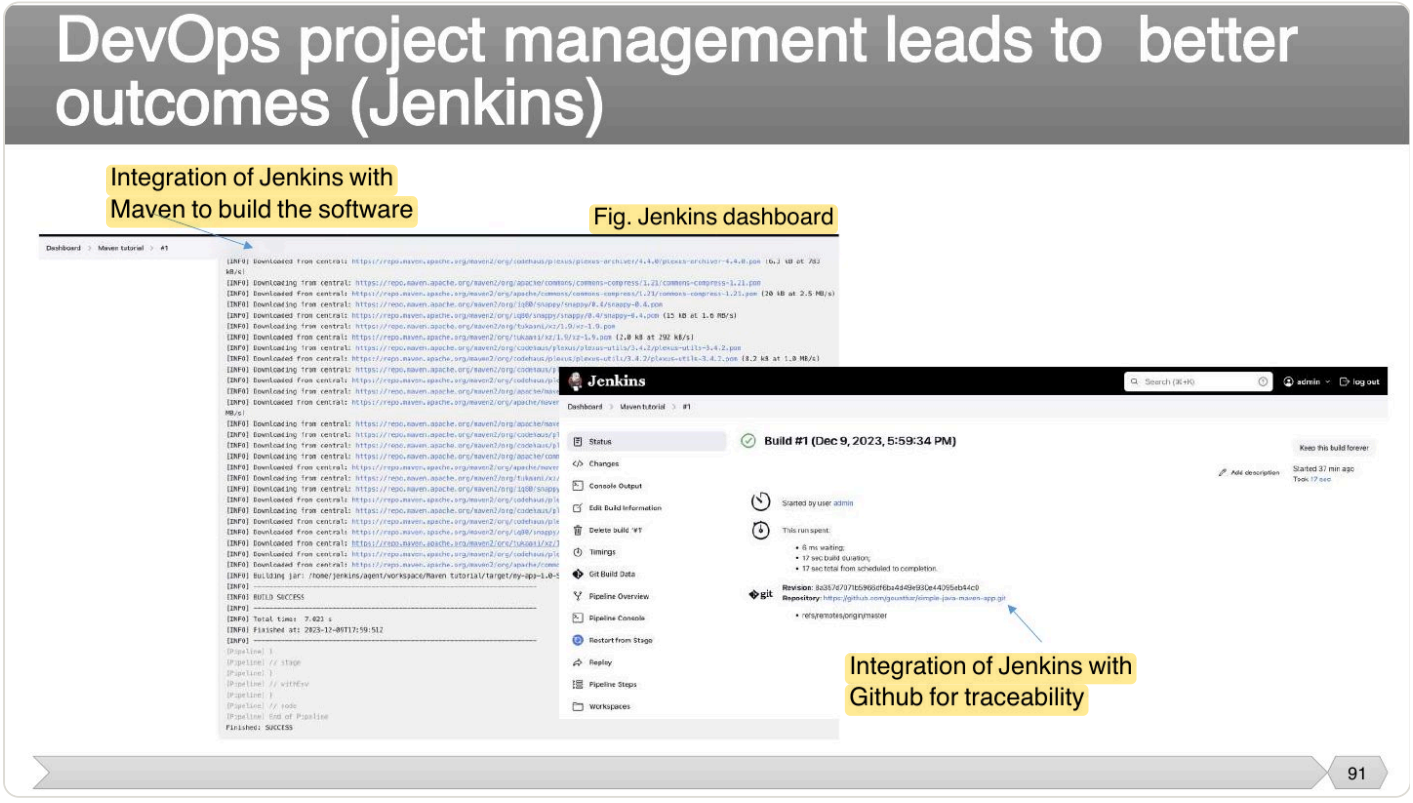
C **Build, test and deployment**

> It's an automation server.

D Budget approval

> That's unrelated.

SLIDE 91 / 98 **outcomes (Jenkins)**



NOTES

1. Nobody has to remember how to build the software correctly each time. Linking Jenkins with **Maven** makes the build repeatable and automatic...
2. ...and with **GitHub for traceability**: each build links back to the commits that triggered it.

EXAM ANGLE

Prediction Q1

What does linking Jenkins with GitHub provide?

A **Traceability from builds to commits**

> That's on the slide.

B Payroll

> That's unrelated.

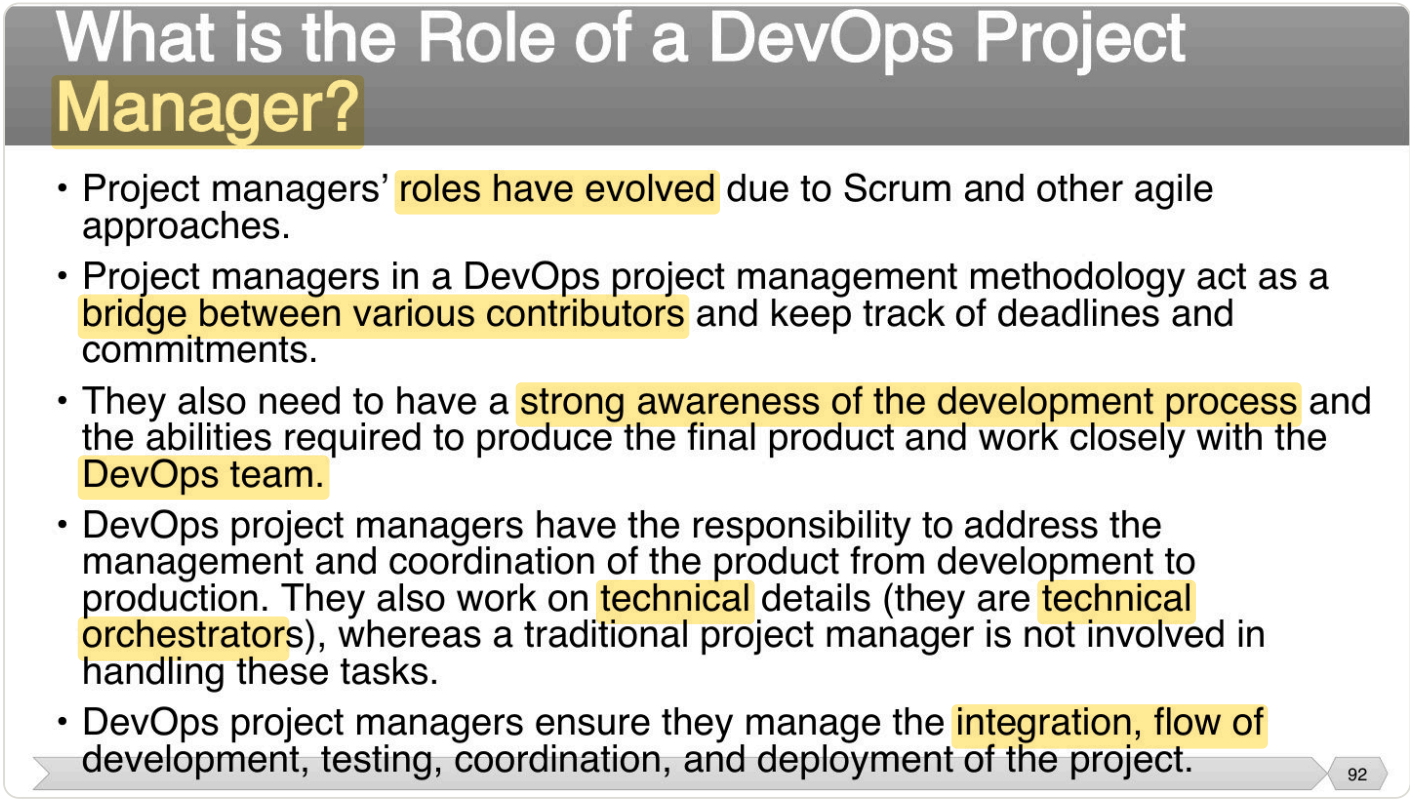
C Stakeholder analysis

> That's unrelated.

D Maven's build logic

> That's the Maven link.

SLIDE 92 / 98 **What is the Role of a DevOps Project**



NOTES

1. Not only the methods changed, the job did too. With self-organizing teams, a PM gives fewer orders and helps more, so **roles have evolved**.
2. Developers, operations and business people have different concerns, and someone has to connect them. A PM as a **bridge** keeps commitments visible to everybody.
3. You can't coordinate work you don't understand. **Strong awareness of the development process** lets the PM spot a real risk rather than just asking for status.
4. A traditional PM may leave technical details to others. Here, the PM gets closer to them, as a **technical orchestrator**, and follows the product all the way into production.
5. Compare with development alone: this covers the whole path to release. **Integration, testing and deployment** all have to fit together, and managing that flow is the job.

EXAM ANGLE

Prediction Q1

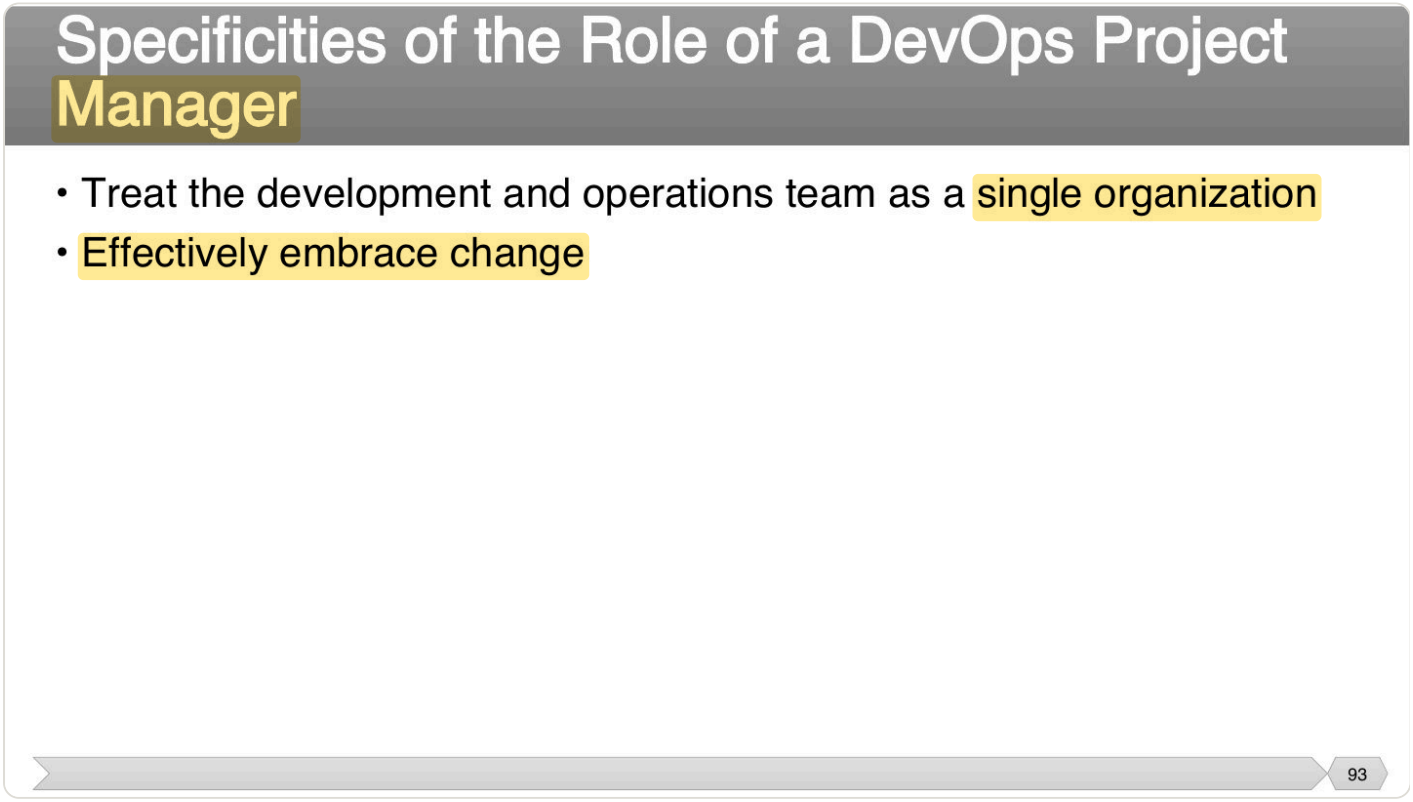
A DevOps project manager stays away from technical details, like a traditional project manager.

T True

F **False**

> DevOps PMs work on technical details as technical orchestrators.

SLIDE 93 / 98 **Specificities of the Role of a DevOps Project**



NOTES

1. If they stay as two groups with different targets, nothing else on these slides works. So **single organization** is a precondition, not an extra...
2. ...change is constant in DevOps, so a PM who treats each change as a problem will be exhausted. To **embrace change** means making it routine.

EXAM ANGLE

Prediction

Q1

What are the two specificities of a DevOps PM's role?

A Keep Dev and Ops apart and avoid change

> That's the opposite.

B Audit vendors and hire staff

> That isn't it.

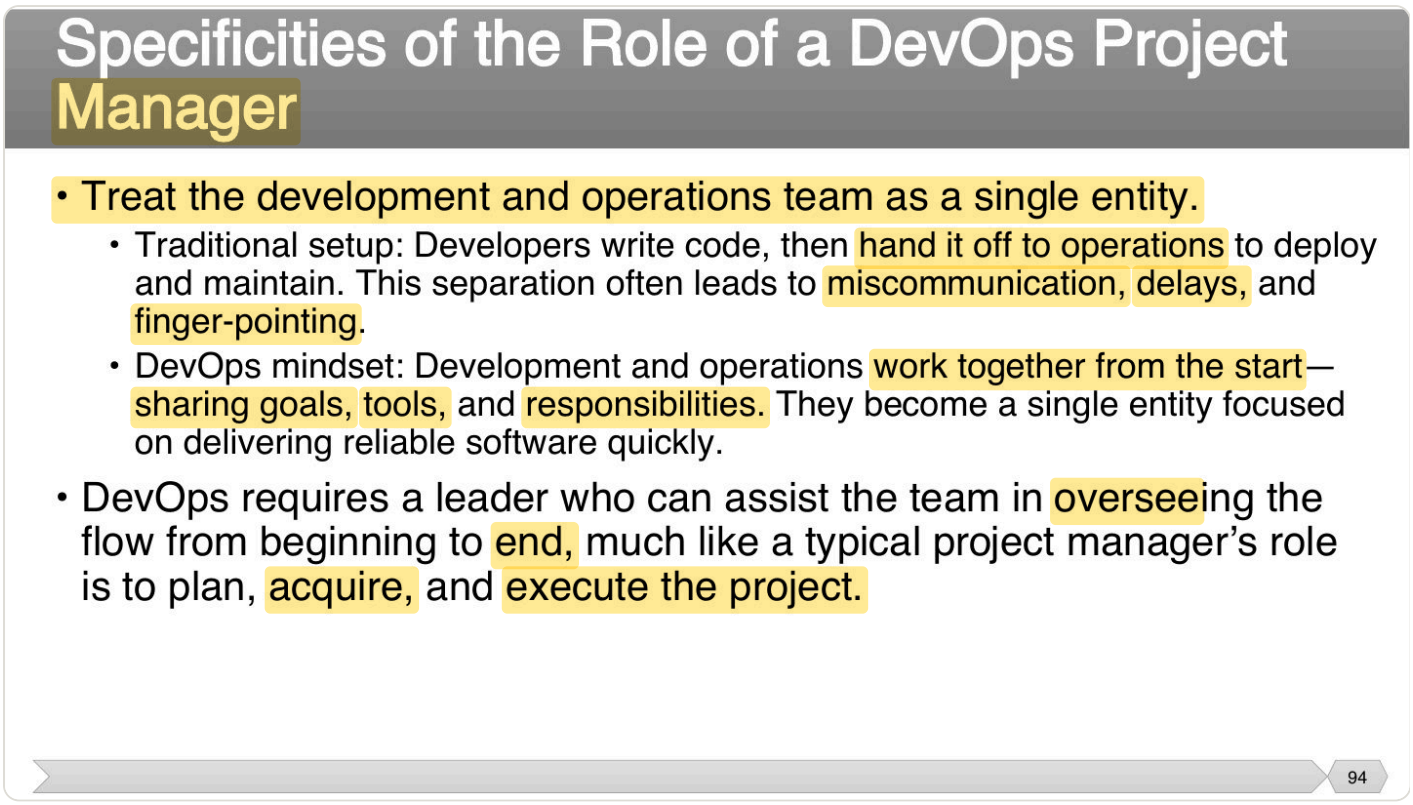
C **Treat Dev and Ops as a single organization, and embrace change**

> These are on the slide.

D Write code and sell the product

> That isn't it.

 Specificities of the Role of a DevOps Project



NOTES

1. Developers want new features out, while operations want stability, and each blames the other when something breaks. **Hand-offs** are where information gets lost.
2. When both are responsible for the outcome, blaming stops being useful. **Working together from the start** means problems in operating the software get raised while they're cheap to fix.
3. Joined teams still need someone watching the whole flow, or gaps appear between steps. So a leader who can **oversee** it end to end is what makes this work.

EXAM ANGLE

Prediction Q1

What can happen in a traditional hand-off between dev and ops?

A Miscommunication, delays and finger-pointing

> That's what the slide lists.

B Shared goals

> That's the DevOps mindset.

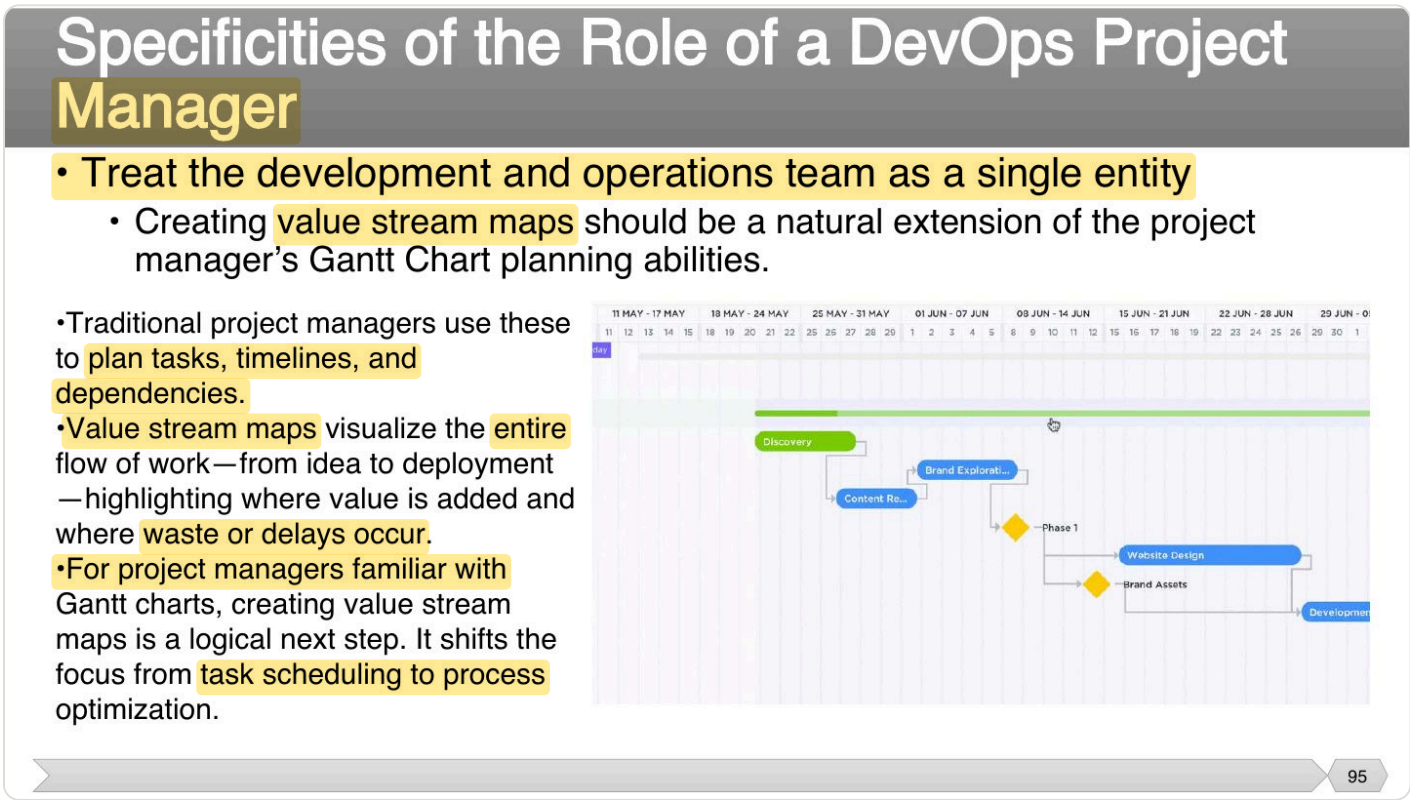
C No need for testing

> That's not stated.

D Faster delivery with no issues

> That's the opposite.

SLIDE 95 / 98 **Specificities of the Role of a DevOps Project**



NOTES

1 If you can read a Gantt chart, you're most of the way there, because both show work over time. **Value stream maps** just ask a different question about it...

2 ...rather than “when will this happen?”, they ask “where does work get stuck?”. If a change waits five days for one approval, that’s **waste** you’d never see on a task list.

3 A schedule tells you when, but it can’t tell you if the process is any good. This shift from **task scheduling to process optimization** is the point of the slide.

EXAM ANGLE

Prediction Q1

A value stream map mainly shows:

A The org chart

That’s unrelated.

B Only people’s salaries

That’s unrelated.

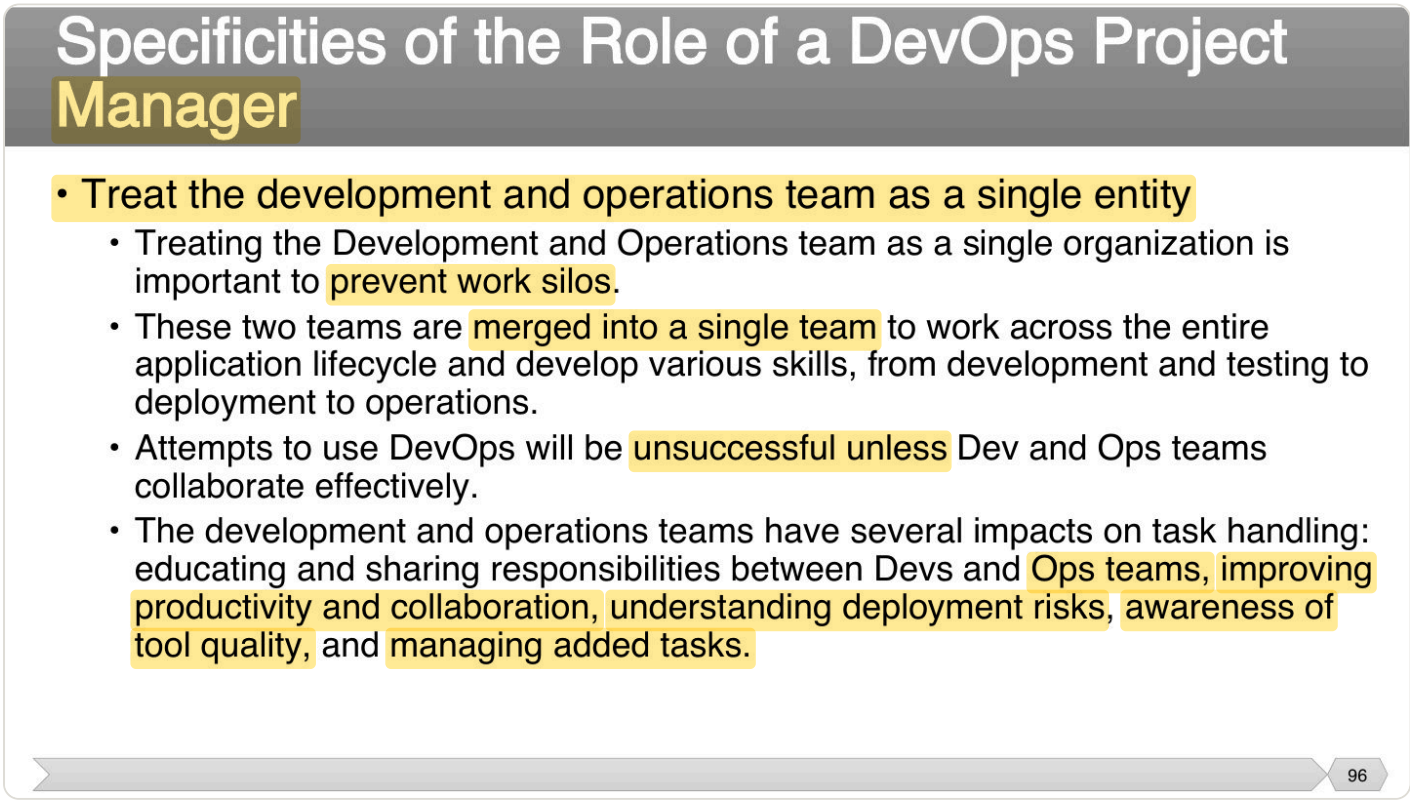
C Only task start dates

That’s closer to a Gantt chart.

D **Where value is added, and where waste or delays occur**

It shows the whole flow.

SLIDE 96 / 98 **Specificities of the Role of a DevOps Project**



NOTES

1. If you only change the names but keep separate goals, nothing changes. A **silo** forms whenever people are judged only on their own part of the work.
2. When people do a bit of everything, they see each other's problems, and they're less likely to create them. That's what **merged** teams gain: a wider view.
3. Buying tools isn't adopting DevOps. Without real collaboration, you get the same conflict with newer tools, which is why it's **unsuccessful unless** they truly work together.
4. A developer who's paged at night writes more careful code. **Understanding deployment risks** is a benefit of sharing responsibilities: people see the consequences of what they do.

EXAM ANGLE

Prediction Q1

What happens to DevOps if Dev and Ops don't truly collaborate?

A It becomes cheaper

> Not stated.

B It becomes waterfall

> Not stated.

C It still works fine

> That's not what it says.

D **It will be unsuccessful**

> The slide says so.

Specificities of the Role of a DevOps Project

Specificities of the Role of a DevOps Project Manager

- Effectively embrace change
  - The agile project management methodology clarifies that embracing change rather than sticking to the plan is crucial.
  - It takes a lot of communication and teamwork to manage change and release features incrementally. Communication is a project manager's primary skill. Enterprises ought to make use of this potential.

NOTES

1. A plan is useful until it stops matching reality. Sticking to it anyway means building what nobody needs, so **embracing change** is what keeps a project useful.
2. When things keep changing, people need to know what changed and why. So a PM's **communication** is what stops frequent change becoming chaos, and it's their strongest tool here.

EXAM ANGLE

Prediction

Q1

What is the PM's primary skill for managing incremental change?

A Procurement

> That isn't named.

B Accounting

> That isn't named.

C Coding

> That isn't named.

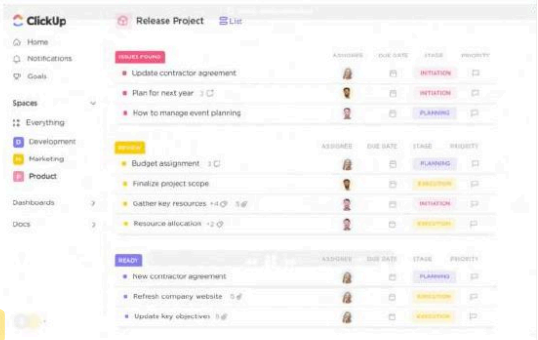
D Communication

> The slide calls it their primary skill.

 Specificities of the Role of a DevOps Project

Specificities of the Role of a DevOps Project Manager

- Effectively embrace change
  - Change isn't just inevitable—it's a catalyst for growth. To thrive in fast-paced DevOps environments, project managers must evolve their mindset and practices:
- Begin with small projects to test ideas and reduce risk
- using the MVP approach: to deliver value early and often
- using the right agile planning tools: to stay adaptive and transparent
- eliminating silos: by fostering cross-functional collaboration
- reducing project handoffs: to streamline workflows, minimize delays
- creating real-time project visibility: with shared dashboards and live updates
- reducing project overhead: by simplifying processes and focusing on value delivery
- managing change collaboratively: Involve your team in planning and adapting to change



98

NOTES

1. A small project limits the damage if the idea fails, and teaches you cheaply. **Start small**, then an **MVP** gets real results quickly, while you learn how the new way of working fits.
2. Every hand-over is a chance to lose information and wait in a queue. So **reduce hand-offs**, and work moves faster with fewer misunderstandings.
3. People make better decisions when they can see what's going on without asking. **Real-time visibility** also makes unnecessary status meetings go away.
4. People resist changes done to them far more than changes they helped shape. **Managing change collaboratively** turns those affected into people who make it work.

EXAM ANGLE

Prediction Q1

Your organization wants to move a waterfall team to DevOps. As the PM, what would you change first and why?

Answer: Treat development and operations as one team with shared goals and tools, to stop hand-offs and finger-pointing. Start with a small project and an MVP so there is a deliverable every sprint; build a CI/CD pipeline with automated tests; map and track dependencies; use shared dashboards (Jira, Jenkins) for real-time visibility; and track metrics such as deployment frequency, lead time, change failure rate and MTTR to show improvement.

Original slides from the SE423 lecture deck, with study notes. Check exact wording against the original deck.
Notes and questions are study aids. Made by Shoug Alomran.