



Aston University

BIRMINGHAM UK

Software Project Management

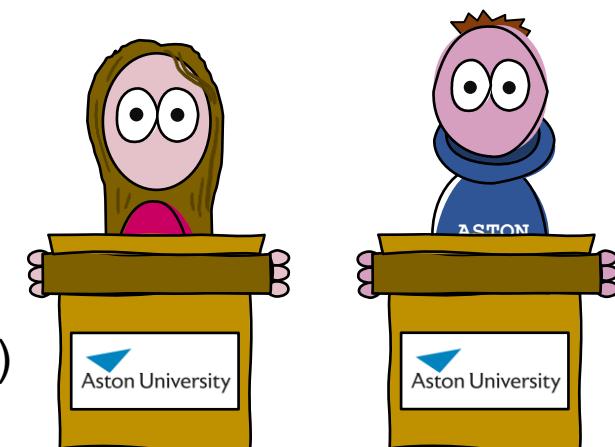
Unit 1: Introduction (part 1)

Thais Webber
Richard Lee



Module staff

- Module tutors:
 - Thais Webber
 - Richard Lee
- Details can be found on Blackboard:
 - Links to WASS calendars
 - e-mail addresses
- Module information
 - Module Specification CS3SPM
 - Module Overview (our planned lectures/tutorials)



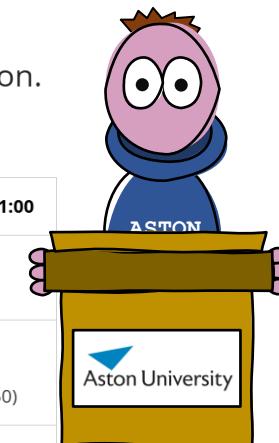
Module Overview (Blackboard)

(2025-26 CS3SPM) Software Project Management

Module Overview

Visible to students   



Welcome to CS3SPM module!

Below you find our weekly content available and scheduled tutorial sessions information.

| Week# | W/C (online) 9:00-11:00 | Content | Tutorial (in-person): 10:00-11:00 |
|-------|-------------------------|---|--|
| 1 | 22-Sep-2025 | Introduction to Software Project Management | - |
| 2 | 29-Sept-2025 | Measurement, estimation & data analysis (part 1) | Measurement tutorial 03-Oct-2025 - Great Hall (450) |
| 3 | 06-Oct-2025 | Measurement, estimation & data analysis (part 2) | Estimation tutorial 10-Oct-2025 - Great Hall (450) |



Motivation

Why Software Project Management module?

- Who has started the Final Year Project (FYP) CS3IP?

Motivation

Why Software Project Management module?

- Who has started the Final Year Project (FYP) CS3IP?
- Do you know that one of the main reasons of failure of the FYP is ...
 - the lack of good strategies to manage the time to develop the project (e.g., following a plan and updating it when required)?
 - the lack of communication with stakeholders (e.g., supervisor, tutor)?



How the customer
explained it



How the project leader
understood it



How the analyst
designed it



How the programmer
wrote it



What the beta testers
received



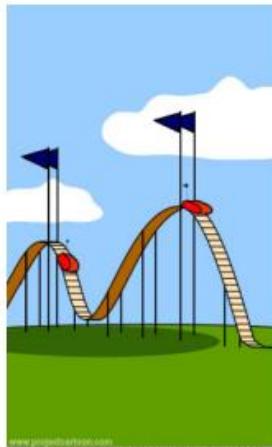
How the business
consultant described it



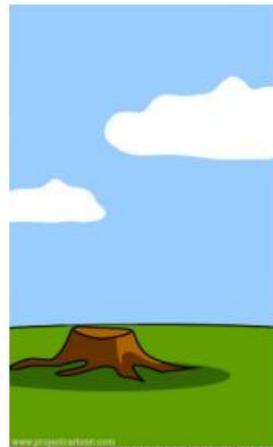
How the project was
documented



What operations
installed



How the customer was
billed



How it was supported



iSwing

What marketing
advertised



What the customer
really needed

What about your Final Year Project?

- Prepare yourself in advance. Ask yourself:
 - What should you do to succeed?
 - What are the risks to avoid?
 - What are the best strategies to follow?
 - What to do if you are confronted with a problem?
- This module will help you to answer those questions

When you have completed this unit, you will be able to:

- Appreciate the need for a project management module dedicated to software development
- Understand the aims, scope and structure of the module
- Define key concepts and terminology associated with software project management

Module information

- Exploring the underlying principles of software project management, and their use in quantitative approaches to managing software projects (Throughout the module)
- How to define software metrics and how to measure, estimate and analyse data associated with software projects (Weeks 2-3)
- How to plan and schedule resources in software projects (Weeks 4-5)
- How to manage, monitor and control risk, quality and people in software projects (Weeks 6-10)

Need for principled, quantitative approaches

Example 1*:

As the software manager responsible for building a new system, you are required to tell the sales team when the system will be ready. You have 25 use cases as requirements, and 5 software engineers who tell you that they can have the system ready to ship in four months.

Do you accept this estimate or not?

**Adapted from Laird & Brennan, Software Measurement and Estimation, Wiley, 2006.*

Need for principled, quantitative approaches

Example 2*:

You are responsible for making a go/no-go decision on releasing a new software application. The testing team tells you that there are approximately eight defects per thousand lines of code left in the system.

Should you say 'yes' or 'no'?

**Adapted from Laird & Brennan, Software Measurement and Estimation, Wiley, 2006.*

This module will provide the knowledge and tools required to take these decisions with confidence, based on rigorous analysis of available data

Why a module dedicated to *software* project management?

- Software is a relatively new field
 - decades old, compared to thousands of years for building bridges/houses
- Subject to rapid change in tools and techniques
- Difficult to learn from experience when all projects are unique
- Lack of physical artefacts
- Large number of unsuccessful / late / over-budget projects

What makes a successful project?

- User involvement
- Executive management support
- Clear statement of requirements
- Proper planning
- Realistic expectations



For reflection...
*What are some examples of
unrealistic expectations in a project?*

Topics CS3SPM

- Introduction ([Week 1](#))
- Measurement, Estimation & Data Analysis ([Weeks 2-3](#))
- Software Project Planning ([Weeks 4-5](#))
- Agile vs Traditional Methodologies ([Week 6](#))
- Risk Management ([Week 8](#))
- Monitoring, Control and Quality Management ([Week9](#))
- Revision ([Week 10](#))

*On CS3SPM Blackboard page:
> Module Information > Module Overview

Mode of learning

- Lecture
 - Mondays, 9-11am
 - Online
 - Recorded and slides provided
- Tutorial
 - (some) Fridays, 10-11am (in-person)
 - On-campus, room can vary (check your timetable)
 - Run by a lecturer and teaching assistants
 - Problem solving based on exam-type questions
 - Questions ideally should be answered in advance
 - Solutions are provided after each tutorial

Assessment

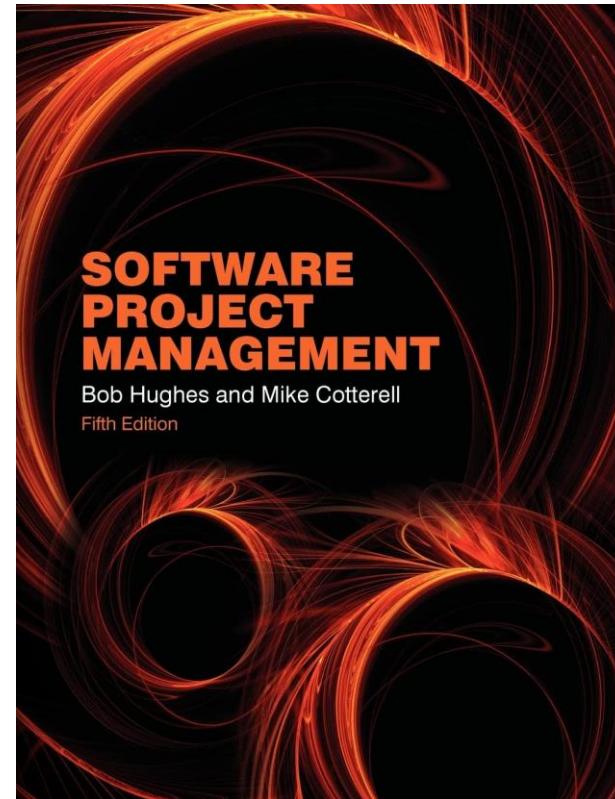
- 100% closed-book exam (in January):
 - Questions on theoretical aspects of Software Project Management covered in lectures and tutorials
 - Problems similar to those addressed in tutorials and revision session
 - Exam contains sections
 - A mandatory section (general questions on all topics covered)
 - A section comprising a choice of questions
 - A guide for self-study for preparation for the exam (and quizzes) will be made available on Blackboard and reviewed in Week 10 (last lecture).

Feedback

- Formal feedback:
 - Example solutions for tutorial problems
- Informal feedback:
 - Discussions during tutorials
 - Questions answered during lectures and labs
 - Office hours

Textbook

- Bob Hughes & Mike Cotterell, *Software Project Management*, 5th edition, McGraw Hill, 2009
- Essential for this module, available from the library as a physical book and eBook
- *Other resources*
 - *Blackboard Module information*
 - *Please check >Talis Reading list*



Additional (recommended) reading

1. The art of scrum : how Scrum masters bind dev teams and unleash agility.
Book author: Dave McKenna, 2016.
2. Agile project management with Kanban. Book author: Eric Brechner, 2015.
3. The Mythical Man-Month. Frederick Brooks (author), 1995; latest Essays on Software Engineering, 2021 ed.

Additional (further) reading

1. Linda M. Laird & M. Carol Brennan, *Software Measurement and Estimation: A Practical Approach*, Wiley-Blackwell, 2006.
2. James Cadle & Donald Yeates, *Project Management for Information Systems*, 5th edition, Prentice Hall, 2007.
3. Norman E. Fenton & Sharli L. Pfleeger, *Software Metrics: A Rigorous and Practical Approach*, 2nd edition, PWS Publishing, 1998.
4. Shari L. Pfleeger & Joanne M. Atlee, *Software Engineering: Theory and Practice*, 4th edition, Prentice Hall, 2009.



Aston University

BIRMINGHAM UK

Software Project Management

Unit 1: Introduction (part 2)

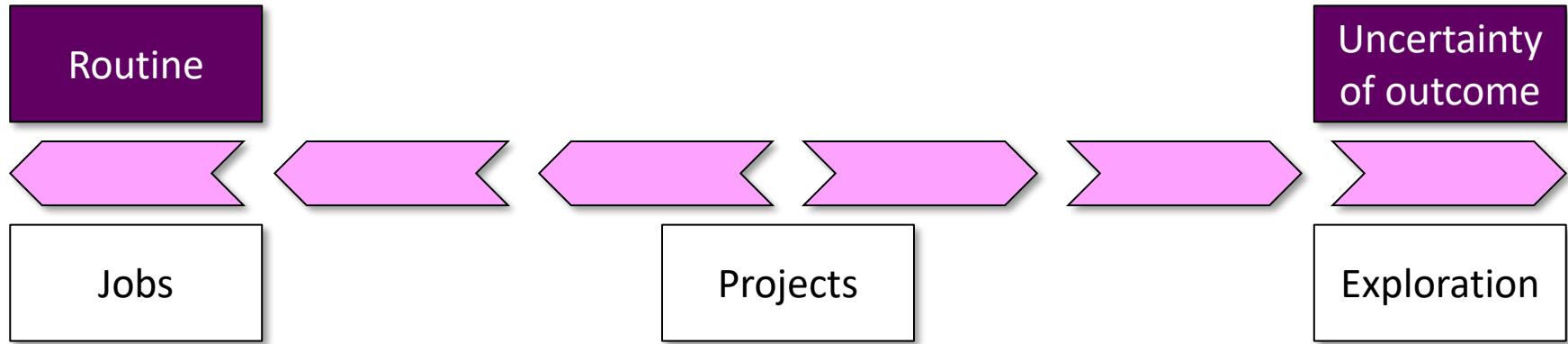
Thais Webber
Richard Lee



What is a project?

- Some dictionary definitions:
 - A specific plan or design
 - A planned undertaking
 - A large undertaking, e.g., a public works scheme
- Key points above are **planning** and **size** of task

Jobs versus projects



- **Jobs**: repetition of well-defined, well-understood tasks with little or no uncertainty
- **Exploration**: e.g., finding a cure for a disease; the outcome, duration and budget are highly uncertain
- **Projects**: lie in the middle of both

Characteristics of projects

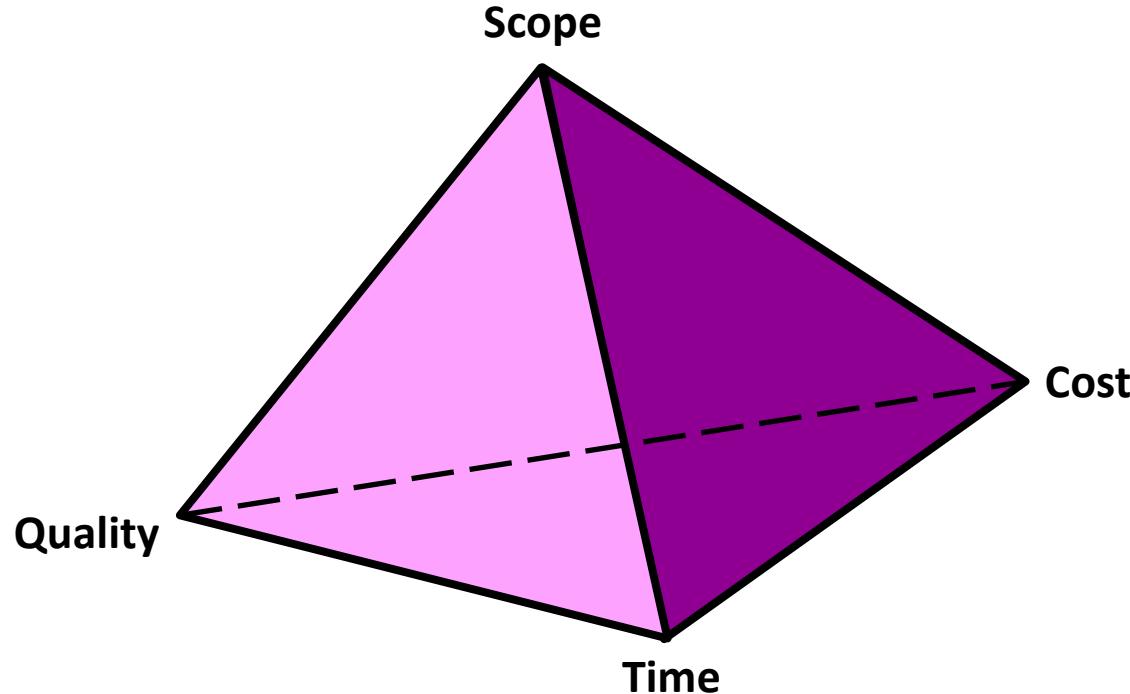
- Non-routine
- Planned
- Aiming at a specific target
- Carried out for a customer
- Carried out by a temporary work group
- Involving several specialisms
- Made up of several different phases
- Constrained by time and resources
- Large and/or complex

Some ways of categorising projects

- Voluntary systems (e.g., computer games) *versus* Compulsory systems (e.g. the order processing system in an organization)
- Information systems (used by human operators directly) *versus* Embedded systems (controlling a device or machine)
- Objective-based (client provides problem that can be solved in multiple ways) *versus* Product-based (client provides product specification)

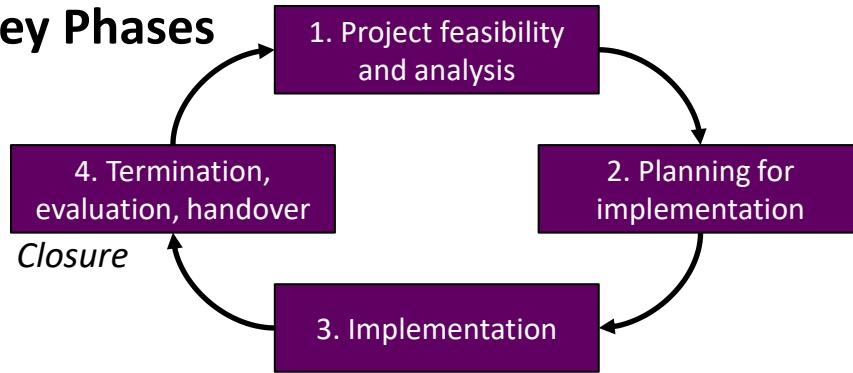
Distinguishing different types of project is important as different types of tasks need different project approaches

Project factors in management

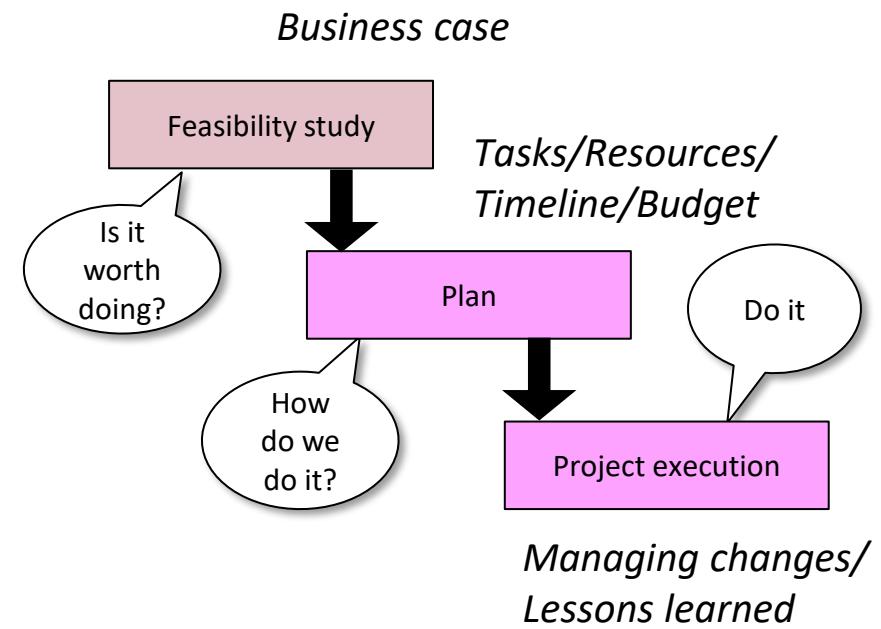


Project management cycle

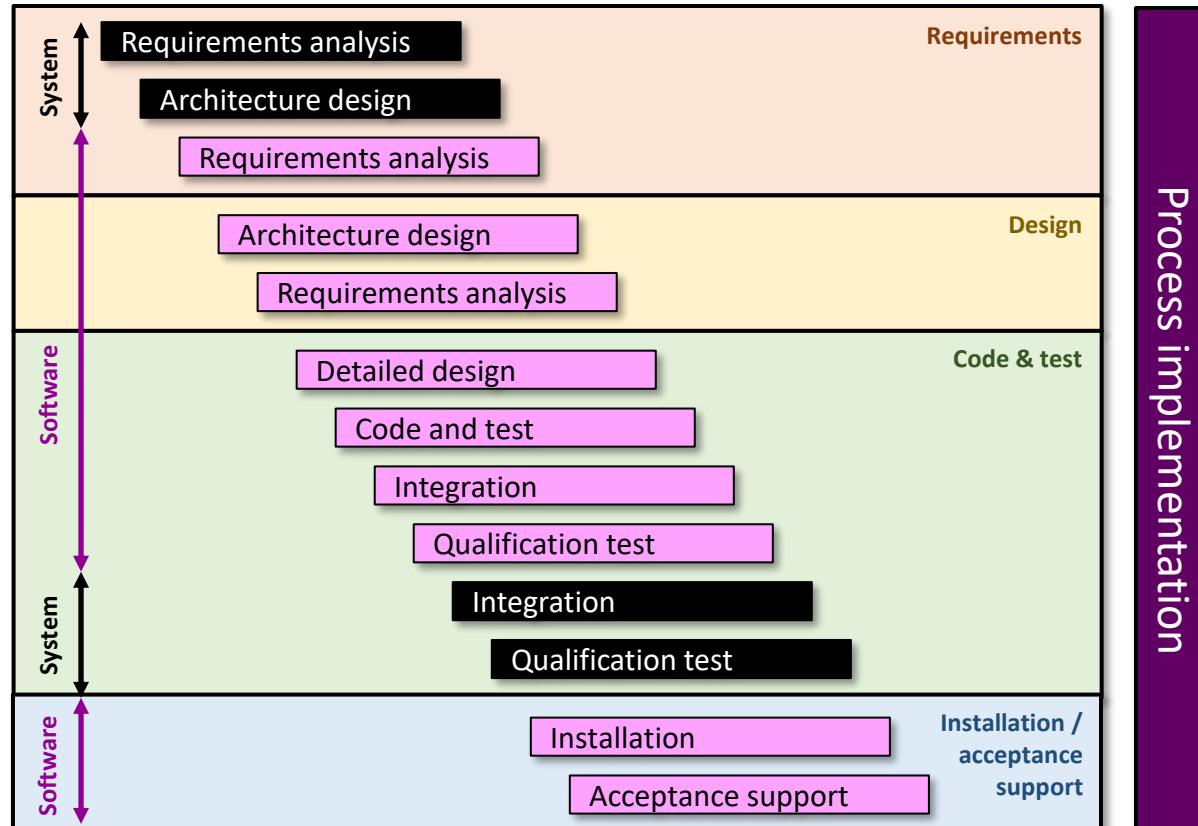
Key Phases



Source: book SPM pg.5



Software lifecycle (ISO 12207)



*A new ISO/IEC/IEEE draft update (P12207) is in development

ISO 12207: requirements

- **Requirements elicitation:** what does the client need?
- **Requirements analysis:** converting ‘customer-facing’ requirements into equivalents that developers can understand
- **Types of requirements**
 - **Functional** requirements: What the software should do—its main features or functions.
 - **Non-functional** requirements: How well the software should work—for example, its speed (performance), reliability, security, and quality.
 - **Resource constraints**: Limitations or restrictions, like cost (budget), deadlines, and technology choices.

Process phases

- **Architecture design**
 - Based on system requirements
 - Defines components of system: hardware, software, organizational
 - Software requirements will come out of this
- **Code and test**
 - Of individual components
- **Integration**
 - Putting the components together

ISO 12207 continued

- Qualification testing
 - Testing the **system** (not just the **software**)
- Installation
 - The process of making the system operational
 - Includes setting up standing data, setting system parameters, installing on operational hardware platforms, user training, etc.
- Acceptance and Support
 - Including maintenance and enhancement

What is a stakeholder?

A stakeholder is any person or organisation impacted by, or able to influence, a software project's requirements or outcome.

- **Why they matter**

- Shape requirements
- Spot problems early
- Provide feedback
- Ensure product is useful and adopted
- Missing stakeholders: unmet needs & higher risk

- **Different stakeholders may have different/conflicting objectives**

- Need to define common project objectives
- Need to agree on an acceptable compromise

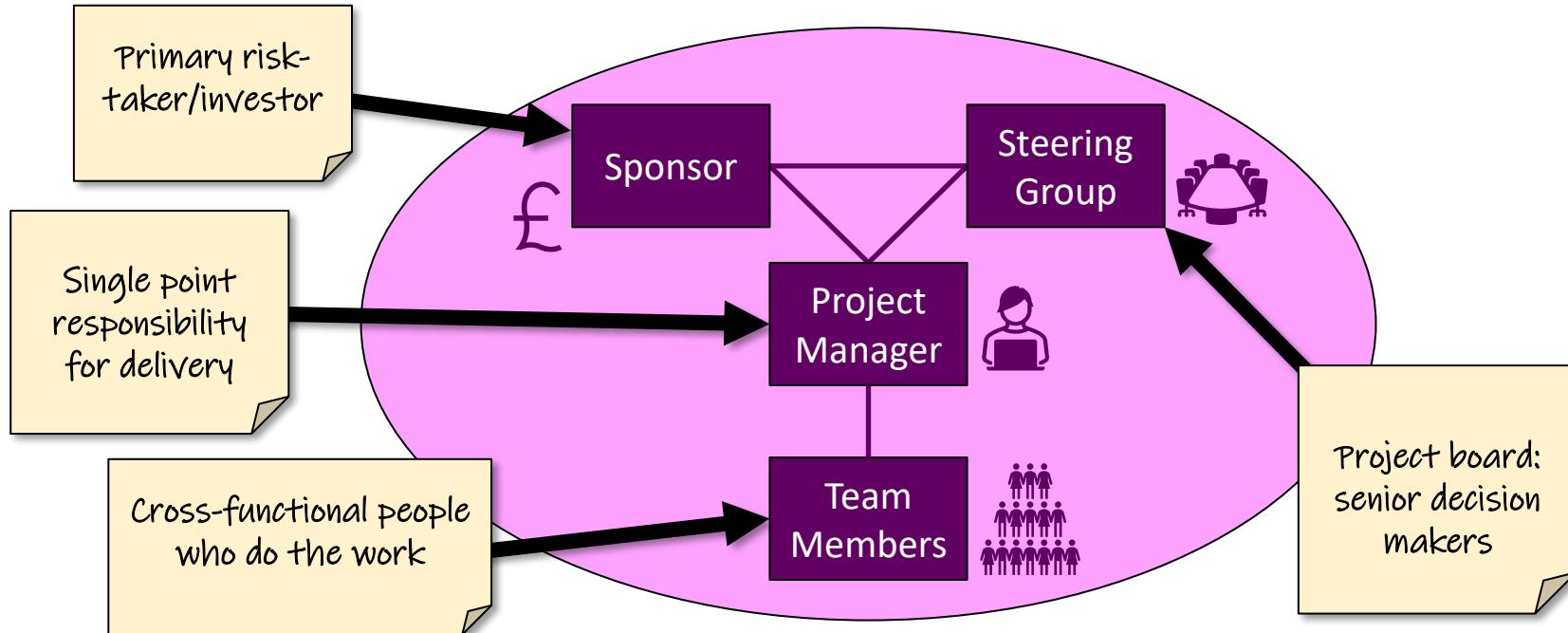
Stakeholders

- Who counts as a stakeholder?
 - **End users** (the ones that use and benefit from the system)
 - **Managers and clients** (sponsors, strategic decision makers)
 - **Project build team** (developers, designers, testers, and project managers)
 - **External groups** (regulatory bodies, legal experts, customers, suppliers, shareholders, and community representatives)
 - **Support and maintenance staff**

Involving the right stakeholders from the start leads to clearer requirements, better software, and smoother project progress.

Stakeholders

Representing stakeholders whilst ensuring a single-point responsibility for delivery



Setting clear project objectives

- Objectives define **what** must be achieved — not the detailed tasks. They focus on the *outcome*, not the *activity*.
- Focus on what will be put in place, rather than how activities will be carried out
 - **YES** Deliver an online booking system that allows users to register, make appointments, and receive email confirmations will be live by 1st December. 
 - **NO** Design and code a booking website using HTML and JavaScript. 

Objectives should be SMART

Example: Deliver an online booking system that allows users to register, make appointments, and receive email confirmations will be live by 1st December.

- S** – **Specific**: concrete and well-defined
- M** – **Measurable**, that is, satisfaction of the objective can be objectively judged
- A** – **Achievable**: it is within the power of the individual or group concerned to meet the target
- R** – **Relevant**: the objective must be relevant to the true purpose of the project
- T** – **Time-constrained**: time-bound, there is a defined point in time by which the objective should be achieved

Goals of Software Project Management

- Given a set of SMART objectives, we want:
 - **Q** – quality software product
 - **C** – cost: no significant budget overshoot
 - **T** – time: no significant delays
 - **P** – good productivity
- Questions:
 - What is ‘software quality’ and (how) can it be measured?
 - Can we determine in advance what a project is going to cost?
 - How do we know if a significant deadline overshoot is likely?
 - Is ‘productivity’ measurable in a software engineering environment?

Levels of project objectives

- **Level 1: Operational:** activity level (short-term) Day-to-day activities which vary greatly depending on the project's process
- **Level 2: Tactical:** project level (mid-term) Quality, Time, Cost, Productivity
- **Level 3: Strategic:** organisational level (long-term)
 - reducing costs
 - expanding customer base
 - increasing user satisfaction
 - improving productivity
 - reducing time-to-market

Tactical project questions (examples)

- How should we ‘choose between competing designs’ (e.g., by complexity)?
- How long will this ‘phase of the project’ take?
- Are the ‘current project risks’ worth addressing? Do we really need to worry about them?
- Which ‘modules’ need extra attention during testing?
- Has ‘team productivity’ changed over the last few months?

(Project-Specific Issues)

Strategic project questions (examples)

- Why are projects in this department often '*over schedule by 30%*' or more?
- Can we '*reduce future costs*' by consistently using this new software tool?
- Does '*Team X consistently produce better code than Team Y*'? If so, why?
- Would adopting a '*new team structure*' significantly improve time-to-market for in-house projects?
- Has our '*cost estimation model*' (used over the last 18 months) been accurate? Can it be calibrated for our environment?

(Long-Term, Organisation-Wide Issues)

Components of Software Project Management*

- Planning – deciding what is to be done
- Organising – making arrangements
- Staffing – selecting the right people for the job
- Directing – giving instructions
- Monitoring – checking on progress
- Controlling – taking action to remedy hold-ups
- Innovating – coming up with solutions when problems emerge
- Representing – liaising with clients, users, developers and other stakeholders

Summary of key points in this unit

- **Software projects differ significantly from other engineering projects:** new field, rapid change, high failure rate & complexity, no physical artefacts
- We need principled, quantitative approaches that can support informed management decisions at strategic, tactic and operational level
- **Specific – Measurable – Achievable – Relevant – Time-constrained SMART** project objectives are required to build quality software within budget, without delays and with good productivity

Review Questions

- EXPLAIN WHY should the objectives of a software project be SMART? What does SMART stand for?
- EXPLAIN WHY is software project management important? Why is it needed?
- EXPLAIN the differences between a job, a project, and outright exploration.
- EXPLAIN WHY is software project management differs from other project management disciplines.
- Start devising SMART objectives for your FYP

Next lectures (Weeks 2-3)

- Unit 2: Measurement, Estimation and Data Analysis



Aston University

BIRMINGHAM UK

Software Project Management

Unit 1: Introduction

Thais Webber
Richard Lee





Aston University

BIRMINGHAM UK

Software Project Management

Unit 2: Measurement,
estimation & data analysis

Thais Webber
Richard Lee



Unit objectives

When you have completed this unit, you will be able to:

- Appreciate the key role of measurement, estimation and data analysis in software project management
- Understand measurement theory and use appropriate scale types to quantify the attributes of software projects
- Understand the need for estimates and use estimates to judge software project characteristics including size, effort and cost

Outline

- Role of measurement, estimation and data analysis
- Key measurement concepts and measurement theory
- Theory and practice of estimation and prediction

Reminder: software process issue

- **Strategic:** organisational level (long term)

- reducing costs
- expanding customer base
- increasing user satisfaction
- improving productivity
- reducing time-to-market

- **Tactical:** project level (medium term)

- Quality, Time, Cost, Productivity
- Process varies from project to project - compare:
 - RAD* of low-cost single-user desktop tool
 - enterprise-wide IS project

- **Operational:** activity level (short term)

- Day-to-day, week-to-week activities which vary greatly depending on the project's process

Role of measurement, estimation & data analysis

- To support reliable quantitative comparisons, evaluations, predictions and decision-making relevant to software projects and their artefacts

| | |
|---|------------|
| Project resource estimation | C, T |
| Project health tracking | Q, C, T, P |
| Productivity measures and models | P |
| Quality measures and models | Q |
| Structural and complexity measures | Q |
| Evaluation of models, methods and tools | Q, C, T, P |

Key: C = cost; T = time; Q = quality; P = productivity

At operational level (short-term)

- collecting 'raw' data
- measuring estimating & predicting
- analysing data (e.g., statistically)
- developing & applying quantitative models
- interpreting (e.g., the outcomes of data analysis)
- experimenting
- evaluating

At tactical level (medium term)

- Obtaining meaningful information that facilitates effective management of software projects
- Better PLANNING
- Improved UNDERSTANDING
- More effective CONTROL

At strategic level (longer-term)

- Contribute to organisation-wide improvement of software processes, and hence to organisation's strategic aims
- Diagnose what is not so good
- Determine changes needed to improve
- Monitor the effects of applying those changes

Use as a management tool to answer questions

- **How well are we doing currently?** (on this project, or projects in general)
- **This (such and such) has an atypical feature ... Why is that?** (Can we make some positive use of this fact?)
- **Do (...X) and (...Y) exhibit some kind of consistent relationship?** (If so, what is it? Could we make good use of it?)
- **How do (...A) and (...B) compare? Is one 'better' than the other?** (If so, why?)
- **Why has (such and such) happened** (or keeps happening)?
- **If we make (such and such) a change, will (such and such) be the (hoped-for) outcome?**

What is measurement?

- Measurement is the process of assigning a value to some attribute of an entity, where the value is obtained on a particular scale
 - e.g. elapsed time of the testing phase of a software package.
- Measurement may require a measurement instrument and may be carried out by a measurer

What to measure?

- Product
 - Specifications, designs, test plans, modules, programs...
- Process
 - Change control, design, project scheduling, review...
- Resource
 - Personnel, money, hardware, software...
- The fact that we can name an attribute does not make it easy or even possible to measure:
 - Complexity, productivity, quality,
 - Length of time before the program stops (the halting problem)

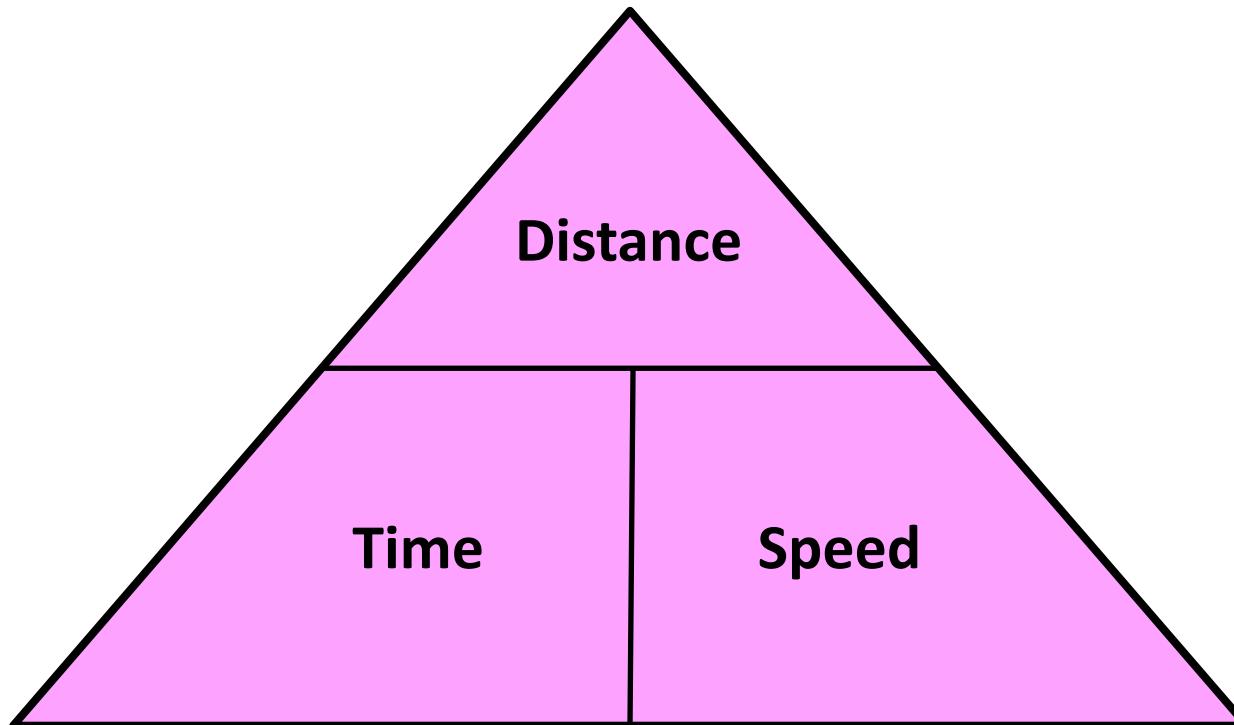
Units

- A measurement unit is a scalar quantity, defined and adopted by convention, with which any other quantity of the same kind can be compared to express the ratio of the two quantities as a number.
 - For example, length can be measured in cm, inches, miles, nanometres...
 - Software size can be measured in characters, bytes, modules, classes, source lines of code (SLOC)

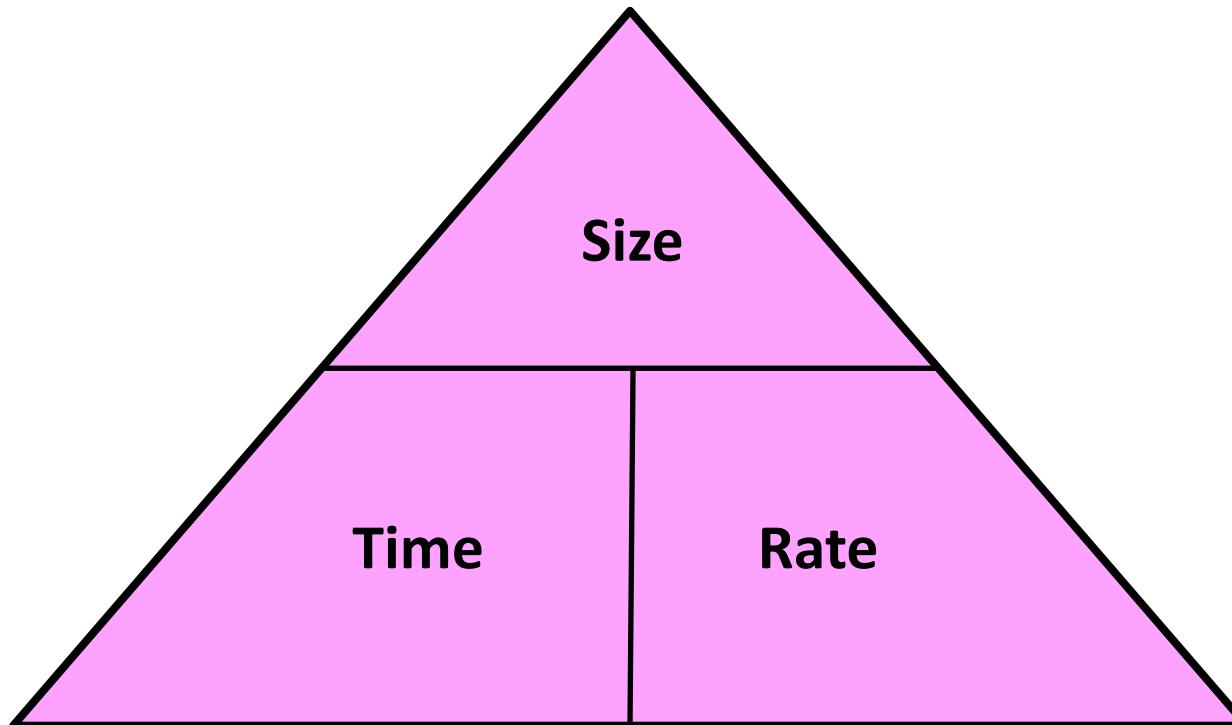
Example

- The size **S** of a Java package to be developed has been estimated at **10,000 SLOC**.
- The average rate **R** at which Java code is produced is **500 SLOC per developer-day**.
- The number of (full-time) developers **N** assigned to the task is **2**
- **How long will it take them to develop the package?**

Example



Example



Direct measurement

- An attribute that can be measured without the need to measure another attribute:
 - Length of source code
 - Duration of testing process
 - Number of defects discovered

Indirect measurement

- An indirect measurement is calculated from direct measurements, typically via addition, multiplication, division:
 - Length of a project (length of development + length of testing)
 - Developer-days (number of developers * number of days)
 - Programmer productivity (SLOC / time spent)
 - Module defect density (number of defects / module size)
 - System spoilage (effort spent fixing defects / total effort spent on project)

Measurement validity

- A measurement is valid if it is:
 - **accurate** – measures the true value of the attribute
 - **reliable** – repeated measurements give similar values
 - **meaningful** – the measurement answers a question that we want answered

Measurement accuracy

- Accuracy is important, but needs to be tempered by context
- Acceptability of accuracy varies
 - If we're measuring file size, we expect 100% accuracy
 - If we're measuring number of defects, perhaps $\pm 10\%$
- Timeliness is more important
 - An approximate number of defects tells us whether the software is ready to ship
 - A precise number of defects doesn't add value, but may take time to determine (and can never be guaranteed as accurate)

Scale types

- We need some way of understanding one measurement in relation to other measurements of the same type of attribute.
- The five types of scale that are available are as follows:
 - Nominal
 - Ordinal
 - Interval
 - Ratio
 - Absolute
- Data on a different scale conveys a different level of precision

Nominal

- This means that the data can only be categorised
- When we compare two measurements, we can say "they are in the same category" or "they are not"
- For example:
 - types of defect (data-flow error, erroneous module interface, incomplete module specification)
 - process model (waterfall, prototyping, incremental)
 - operating system (Windows, MacOS, Linux)

Ordinal

- We can categorise **and rank** data into order, but we don't have any indication of the gap between two ranked items.
- For example:
 - Programmer skill (beginner, intermediate, expert)
 - Survey questions on quality (strongly disagree ... strongly agree)
 - Top-five paying clients
- We know, for example, that an intermediate programmer has greater skill than a beginner programmer, but we do not know how that gap compares with the intermediate-expert gap

Interval

- We can categorise and rank data points
- We can infer equal intervals between them
- There is no 'zero point'
- For example:
 - Progress across time (**P1** = 23/09/2024, **P2** = 30/09/2024, **P3** = 07/10/2024)
 - Credit scores (no one has a score of zero)
 - Celsius (there is a zero on this scale, but that does not mean a lack of temperature)
- We can say "one week later", but we can't say "twice as late"

Ratio

- We can categorise and rank data points
- We can infer equal intervals between them
- There is a true 'zero point'
- For example:
 - Source lines of code
 - Number of days delayed from deadline
 - Number of full-time programmers
- If you can say "there are twice as many" or "there is twice as much", and if zero has a specific meaning, it's a ratio scale

Absolute

- A subset of 'ratio'
- Cannot be determined indirectly from any other measurements
- Typically used for counting
- For example:
 - Number of defects detected by a certain test method
 - Number of lines of code reused from a previous project

Admissible transformations

- Nominal scale
 - $M = \{\text{RAD, Incremental, Waterfall, Spiral, Prototyping}\}$
 - $M' = \{1, 18, 6, 5, 36\}$
- Ordinal scale
 - $M = \{1, 2, 3, 4, 5, 6\}$ **e.g. programmer skill**
 - $M' = \{2, 6, 12, 20, 42, 88\}$
- Interval scale
 - $M = \{1, 2, 3, 4\}$ **e.g. dates at which progress is regularly reviewed**
 - $M' = \{7, 9, 11, 13\}$
- Ratio scale
 - $M = \{0, 1, 2, 3\}$ **e.g. number of hours spent on a task**
 - $M' = \{0, 60, 120, 180\}$
- Absolute scale
 - $M = \{0, 1, 2, 3\}$ **e.g. number of lines of code re-purposed from a previous project**
 - $M' = \text{N/A}$

Which scale?

1. You are assessing the programming languages used in a software project. The data includes: Python, Java, C++, Ruby, and PHP.
2. You are rating the quality of customer service on a scale of 1 to 5, with 1 being "Poor" and 5 being "Excellent."
3. You are measuring the time taken by a computer program to execute various tasks in seconds.
4. You are measuring temperature in degrees Celsius, where 0°C represents freezing point.
5. You are counting the number of errors in a software code.

Case study 1: software complexity

- What do we mean by complexity?
 - a. size
 - b. control flow
 - c. structure
 - d. interface
 - e. connection
 - f. algorithm
- Software complexity: $SC = F(a, b, c, d, e, f)$
- How would we calculate complexity?
- Any attempt to reach a single-value complexity is likely to fail

Case study 2: developer productivity

- Software productivity is a key management concern, considering:
 - cost containment
 - monitoring performance
 - comparison
 - assessment
- Straightforwardly, $Pr = \text{output} / \text{input}$
- In software, $Pr = S / E$ (size / effort)
- What problems can arise from using SLOC
- What problems can arise from using defects resolved

The problem with productivity

- Validity issues with $Pr = \text{SLOC} / E$
- It's an indirect measurement, but of what?
 - SLOC ignores quality
 - SLOC ignores difficulty/complexity
 - SLOC ignores functionality/utility
 - SLOC ignores task value
 - SLOC ignores those who do not directly write code
- SLOC is easy to measure, and superficially insightful
- What might be a better measurement?



Aston University

BIRMINGHAM UK

Software Project Management

Unit 2: Measurement,
estimation & data analysis

Thais Webber
Richard Lee





Aston University

BIRMINGHAM UK

Software Project Management

Unit 2: Measurement,
estimation & data analysis

Thais Webber
Richard Lee



Unit objectives

When you have completed this unit, you will be able to:

- Appreciate the key role of measurement, estimation and data analysis in software project management
- Understand measurement theory and use appropriate scale types to quantify the attributes of software projects
- Understand the need for estimates and use estimates to judge software project characteristics including size, effort and cost

Outline

- Role of measurement, estimation and data analysis
- Key measurement concepts and measurement theory
- Theory and practice of estimation and prediction

What is estimation?

- **Estimation** is the process of predicting an expected value for some attribute (whose value is not known)
- The **uncertainty** is expressed in terms of probabilities
- **Estimates** are forecasts or predictions; we estimate what we cannot measure
 - the entity whose attribute we estimate may not exist yet or is not sufficiently mature to be measured accurately
 - not because we don't understand what we are trying to measure

What is estimation?

- Estimation is closely linked to measurement because estimates are based (explicitly or implicitly) on data obtained from measurement
- Representational validity and scale types are still just as relevant
- The most common use of estimation in software projects is **resource prediction**

Resource prediction

- When needed?
 - determining if a project proposal is feasible resource-wise
 - bidding for a contract
 - planning a project: predicting cost and duration, etc.
 - tracking resource expenditure during a project

If estimates are poor, the resulting losses can take a company out of business

Modelling uncertainty

- In principle, measurements are 'exact'
 - predictions are uncertain
- An estimate of an attribute A lies within some overall range of possible values on the scale X to which the (actual) value of A belongs
 - different values on X will have different 'likelihoods' of corresponding to the actual value of A
- **Estimates and probability are closely linked**
 - **Reminder:** a probability is formally expressed as a value in the interval [0.0, 1.0], though you will also see it as a percentage (i.e., in the interval [0.0, 100.0])

Estimates and probability

- **Continuous variables**

- Example: the duration of a software project, the weight of a server
- If we estimate that attribute A will have the specific value a , then a is termed a ‘point estimate’, and the probability of our estimate being precisely correct is zero
- If we estimate that A will have a value that lies in a non-empty interval , then the probability of being correct can be non-zero

- **Discrete variables**

- Example: number of people, SLOC
- It is still likely that there is uncertainty - each of the possible values a_1, a_2, \dots, a_n may have a non-zero probability

Confidence intervals

- $[L, U]$ denotes an **interval**; all values v such that $L \leq v \leq U$
 - L is the **lower bound**
 - U is the **upper bound**
- The probability P of the actual value a (of attribute A) lying in the interval $[L, U]$ is written as:
$$P(a \in [L, U])$$
- If $P(\text{project_duration_in_months} \in [8, 12]) = 0.95$, then $[8, 12]$ is a 95% confidence interval

Interval notation for estimates

- (L, M, U) is a triple spanning the interval $[L, U]$ where M , the **central value** of the triple, is the **expected value** of the estimate
- If M in (L, M, U) is the **mid-point** of the interval $[L, U]$ (it does not have to be), then:

$$M - L = U - M = \delta$$

and $(L, M, U) = (M - \delta, M, M + \delta)$, which is more usually written:

$$M \pm \delta$$

Example: the estimate for the number of defects in a software application is 20 ± 3 per thousand SLOC

Improved estimates

- Ways to improve on an expert judgement:
 - Group estimation (the Delphi method)
 - Task decomposition
 - Analogy
 - Formal modelling

The Delphi method

- **Iterative prediction technique involving a group of experts**
 - Each expert provides an estimate in each iteration
 - The estimates from all experts are “summarised” (e.g., averaged) and the result is communicated back to the experts...
 - ... and used to come up with a new set of expert estimates in the next iteration
- Estimates obtained by a group of experts are more accurate than those provided by a single expert



Estimation by decomposition

- **Decompose**

- a project's software process into phases
- a project phase into 'activities' or 'work packages'
- a system into separate subsystems or 'functional units'

- **Top-down:** Estimate an overall value and then proportionately distribute this over the components of the distribution.

- When is this appropriate? When the project completion time is fixed

- **Bottom-up:** Estimate a value for individual components, and then obtain the overall figure (e.g. by summing them).

- When is this appropriate? e.g., for an open-ended or research project

Formulae

- **S size** - example units SLOC (but it could also be number of modules)
- **EpS** (effort per size) - example of units used: pm/SLOC
- **SpE** (size per effort) called **Productivity** - example of units: SLOC/pm
- **CpS** unit cost - example of units used: £/SLOC
- **CpE** unit cost rate - example of units used: £/pm
- **Effort = S x EpS** - example working units: SLOC x pm/SLOC → pm
- **Effort = S/SpE** working the units: (SLOC) / (SLOC/pm) → pm
- **Cost = S x CpS** example of units £ but it could also be € (EUROS) - example working the units: SLOC x £/SLOC → £
- **Cost = Effort x CpE** - example working the units: pm x £/pm → £
pm = person-month, but it could be person-week (for example)

Worked example 1

- Attributes estimated: system size, development effort and cost
- Decomposition: system into ‘functional units’/ subsystems (USI, GRD, DBMS)
- Strategy: bottom-up with multiple conversion factors
- **Delphi average = (Min + 4 * Likely + Max) / 6**

Worked example 2

- Attributes estimated: development effort and cost
- Two dimensional decomposition: system into subsystems; process into phases
- Strategy: bottom-up using expert judgement
- What will this project cost?

| Function | Requirement | Design | Code | Test | Total |
|-------------------|-------------|--------|-------|-------|-------|
| USI | 1.0 | 2.0 | 0.5 | 3.5 | 7.0 |
| GRD | 1.5 | 11.0 | 4.0 | 10.5 | 27.0 |
| DBMS | 2.0 | 6.0 | 3.0 | 4.0 | 15.0 |
| Total pm | 4.5 | 19.0 | 7.5 | 18.0 | 49.0 |
| CpE (£/pm) | 5,200 | 4,800 | 4,250 | 4,500 | |

Estimation by analogy

1. Look for a past project (or unit) **Q** that is ‘very similar’ to the new project **P**
2. Obtain recorded effort **value*** **E** for **Q**: **Q.E**
3. Identify orthogonal factors accounting for significant differences between **Q** and **P** & expected to impact **P**’s effort
4. Adjust **Q.E** to take into account factors identified in (3):

$$\mathbf{P.E} = \mathbf{Q.E} * (\mathbf{Rf}_1 * \mathbf{Rf}_2 * ...) + (\mathbf{Af}_1 + \mathbf{Af}_2 + ...)$$

What might be typical ratio factors **Rf_i** or additive factors **Af_i**?

- E.g., size or development rate for ratio factors; different components between P and Q for additive factors

* effort might be time, cost, development time, etc.

Worked example

- We will develop a new graphics project G
- We have developed in the past a similar project called Q
- Suppose we know from project Q, that $Q.E = 100pm$
 - to develop Q, 100 person-months were needed
- Question: What is the effort that you predict for the new project G?
- One significant difference between G and Q is size
 - $Q.S = 25,000 \text{ SLOC}$
 - $G.S \approx 20,000 \text{ SLOC} \pm 2,500 \text{ SLOC}$

Finding analogous projects

- Need database of relevant past project data
- What is an ‘analogous project’?
 - Need attributes that can be measured at estimation time, that are reasonably independent and correlated with variable you are trying to predict
 - Analogue identification mostly by expert judgement
- Identify adjustment factor(s) and estimate them quantitatively for new project

Summary of key points in this unit

- Measurement, estimation and data analysis are key software project management tools
 - they support reliable quantitative comparisons, evaluations and decision-making
- The values of software project/artefact attributes can be established
 - using measurement for existing entities
 - using estimation/prediction for entities that will exist in the future

Tips for FYP this week

- Consider the Top-Down Decomposition of tasks
- Useful tools: Trello - <https://trello.com/en> & Gantt Chart
- SMART Goals / Objectives

Unit objectives

When you have completed this unit, you will be able to:

- Appreciate the key role of measurement, estimation and data analysis in software project management
- Understand measurement theory and use appropriate scale types to quantify the attributes of software projects
- Understand the need for estimates and use estimates to judge software project characteristics including size, effort and cost



Aston University

BIRMINGHAM UK

Software Project Management

Unit 2: Measurement,
estimation & data analysis

Thais Webber
Richard Lee





Aston University

BIRMINGHAM UK

Software Project Management

Unit 3: Software Project Planning (1)

Thais Webber
Richard Lee



Unit objectives

When you have completed this unit, you will be able to:

- Approach project planning in an organised, step-by-step and top-down manner
- Select an appropriate development and life cycle approach for projects
- Apply estimation techniques to predict software development effort, using bottom-up estimation to avoid unrealistic predictions
- Produce an activity plan for a project, identify required resources and devise work plans and resource schedules

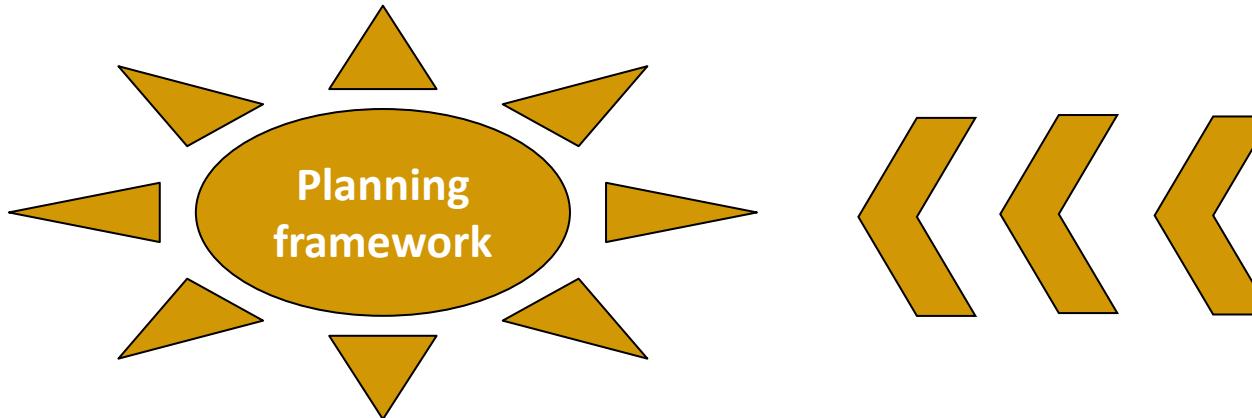
Outline

- Frameworks for software project planning
 - Motivation
 - Step Wise project planning
 - PRINCE2 project planning
- Selection of software project approaches
- Effort estimation for software projects
- Activity planning and resource allocation

Motivation for the use of frameworks

- People are often at a loss as to where to start in project planning

- Students involved in individual and group projects (consider your own experience in individual and group projects)
 - Professionals new to software project management



Motivation for the use of frameworks

- Structured approaches or frameworks for software project planning address the need for systematic and repeatable planning:
 - They list the main **steps** involved in project planning
 - They suggest **best practices** for carrying out these steps
 - They indicate logical **order of steps**, including possible iterations or feedback loops
 - They remain **applicable** across different application domains
 - Their concepts and techniques apply to manage **small** or **large/complex** projects

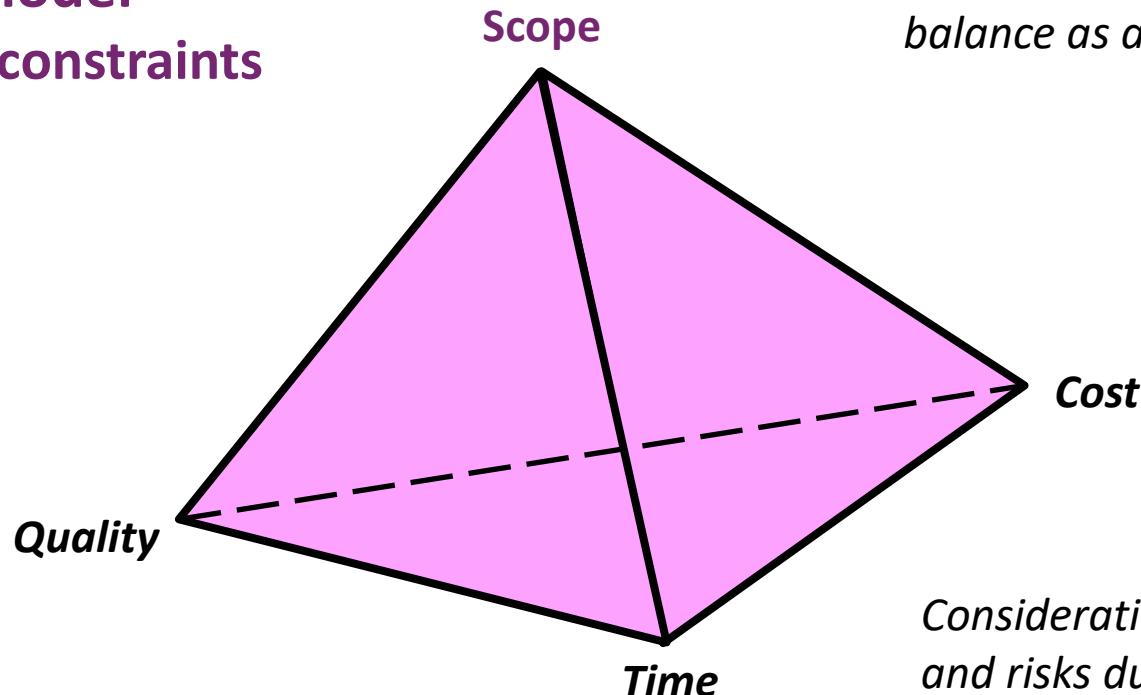


Motivation for the use of frameworks

- Two planning frameworks covered in the module
 - **Step Wise** – focuses on planning techniques
 - **PRINCE2** – focuses more on procedural aspects
- Both can be used in combination with:
 - Traditional methodologies (such as Waterfall, Incremental, Iterative, Spiral, V-model, etc.)
 - Agile approaches (such as Scrum, Kanban, XP, etc.)

Project factors (from SPM Introduction)

QCTS model
Project constraints



Interdependent constraints to balance as a project manager

Considerations on resources and risks during planning

Success factors for project management

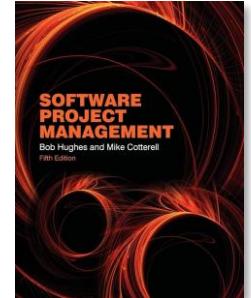
1. Agree on all project goals
2. Develop clearly defined plans with assigned responsibilities and accountabilities
3. Manage the project scope effectively
4. Cultivate constant effective communication
5. Make sure you have management support

Outline

- Frameworks for software project planning
 - Motivation
 - Step Wise project planning
 - PRINCE2 project planning
- Selection of software project approaches
- Effort estimation for software projects
- Activity planning and resource allocation

Step Wise project planning framework

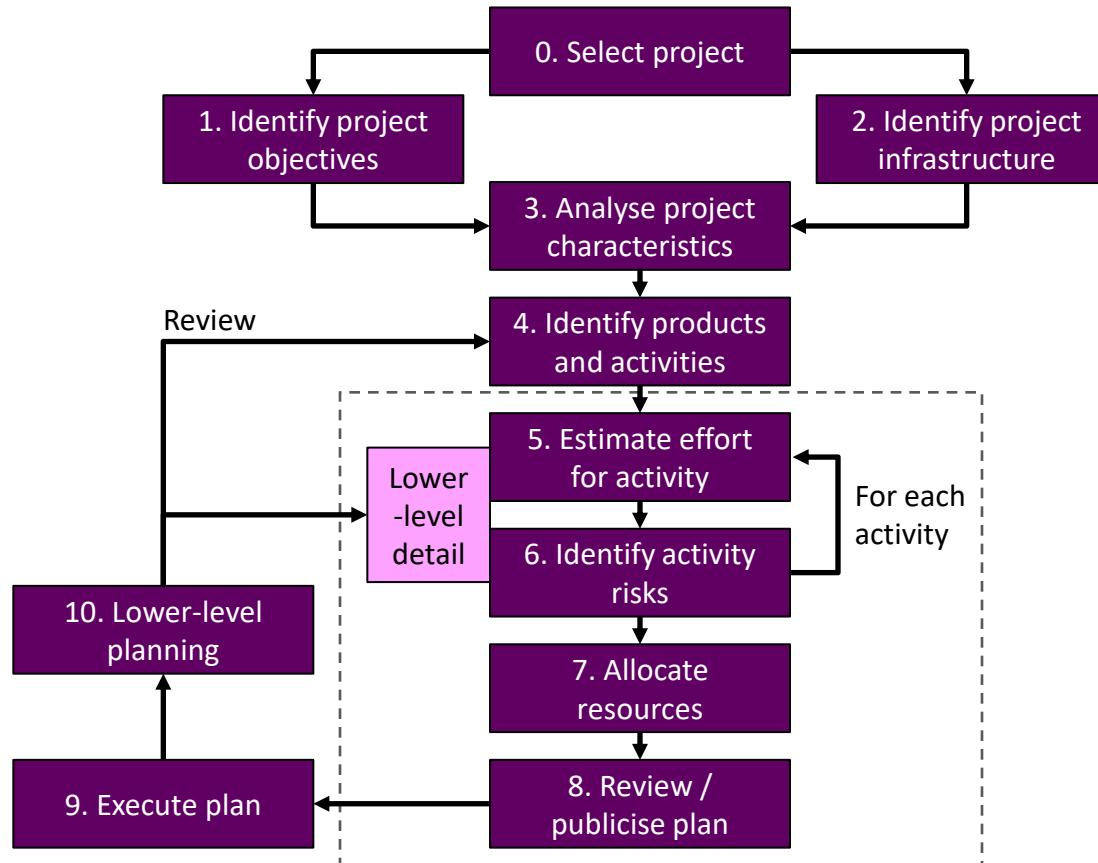
- Developed and enhanced over the years by Robert Hughes at the University of Brighton
- Tries to answer the question: “What sequence of steps should we follow to produce a realistic, defensible project plan for this software project?”
- Scalability
 - useful for small project as well as large
 - range of applications including software development projects



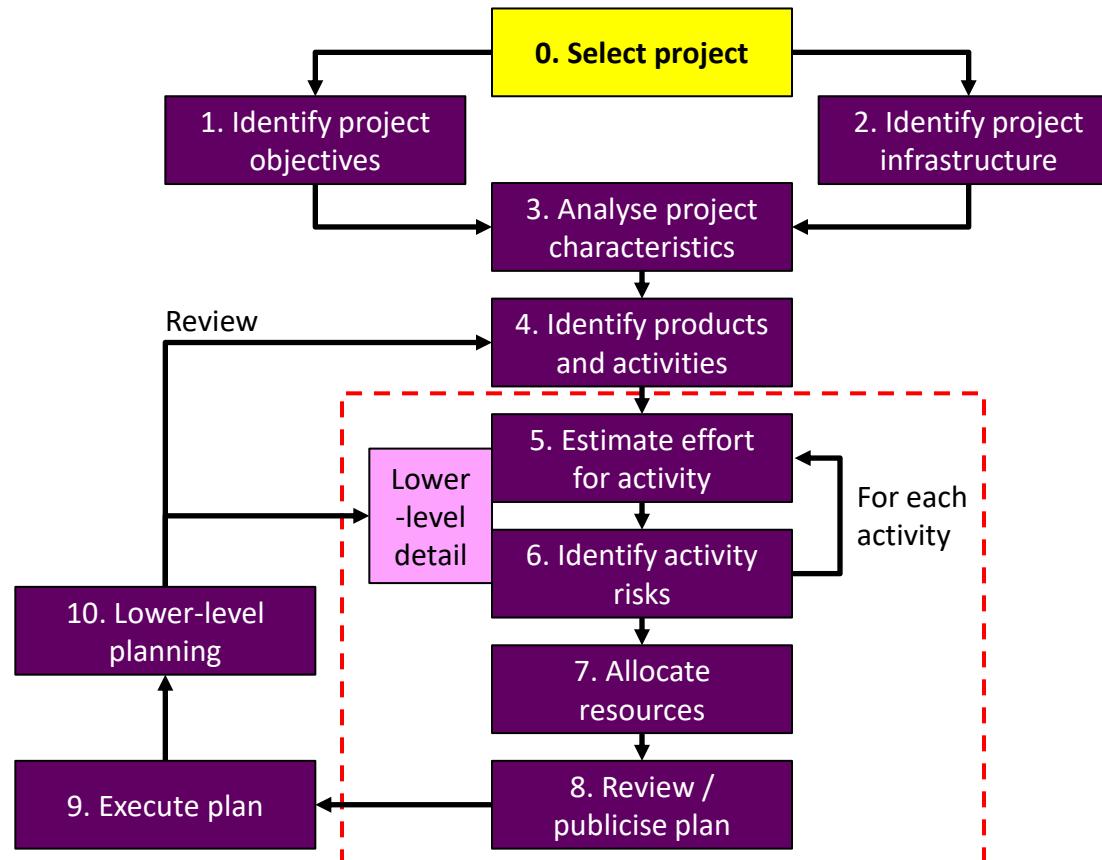
(refer to
Book SPM,
Hughes &
Cotterell –
Ch.3)

Step Wise - an overview

(Book SPM
Hughes & Cotterell – Ch.3)



Step Wise - an overview

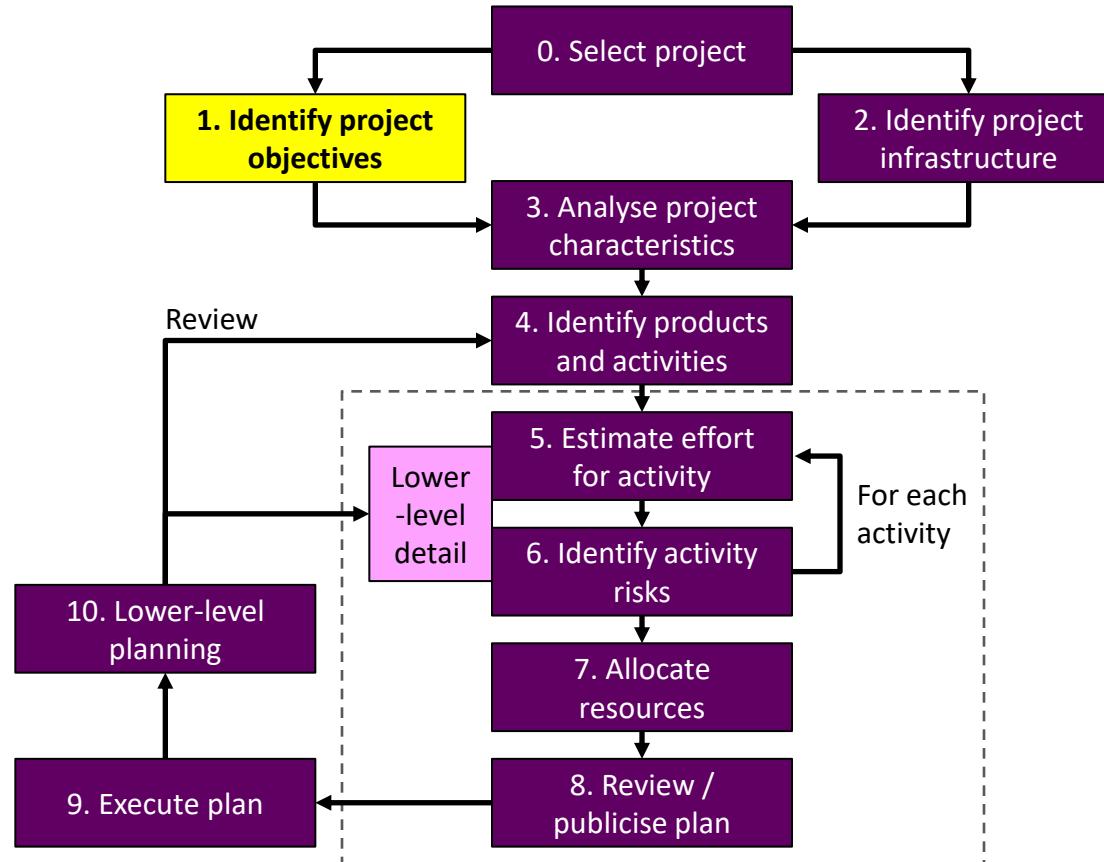


Project scenario: college payroll system

- College currently outsources payroll processing, which is costly and limits data analysis
- Decision made to bring payroll **in-house** by acquiring an **off-the-shelf** application - supported by a new internal payroll office and small software add-ons to integrate with timetabling data.
- About the decision alternatives:
 - In-house *versus* outsourcing (external/third party)
 - Off-the-shelf *versus* custom-developed

Assume you were hired as the project manager

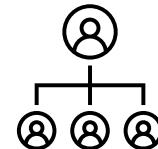
Step Wise - an overview



Step 1: identify project objectives

- 1.1 Identify objectives and measures of effectiveness
 - how do we know if we have succeeded?
 - From Unit 1: objectives must be **SMART**:
 - **S**pecific, **M**easurable, **A**chievable, **R**elevant, **T**ime-constrained

- 1.2 Establish a project authority



- who is the boss?

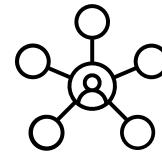
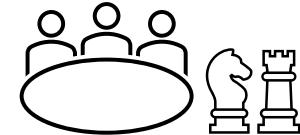
- 1.3 Identify all stakeholders in the project and their interests



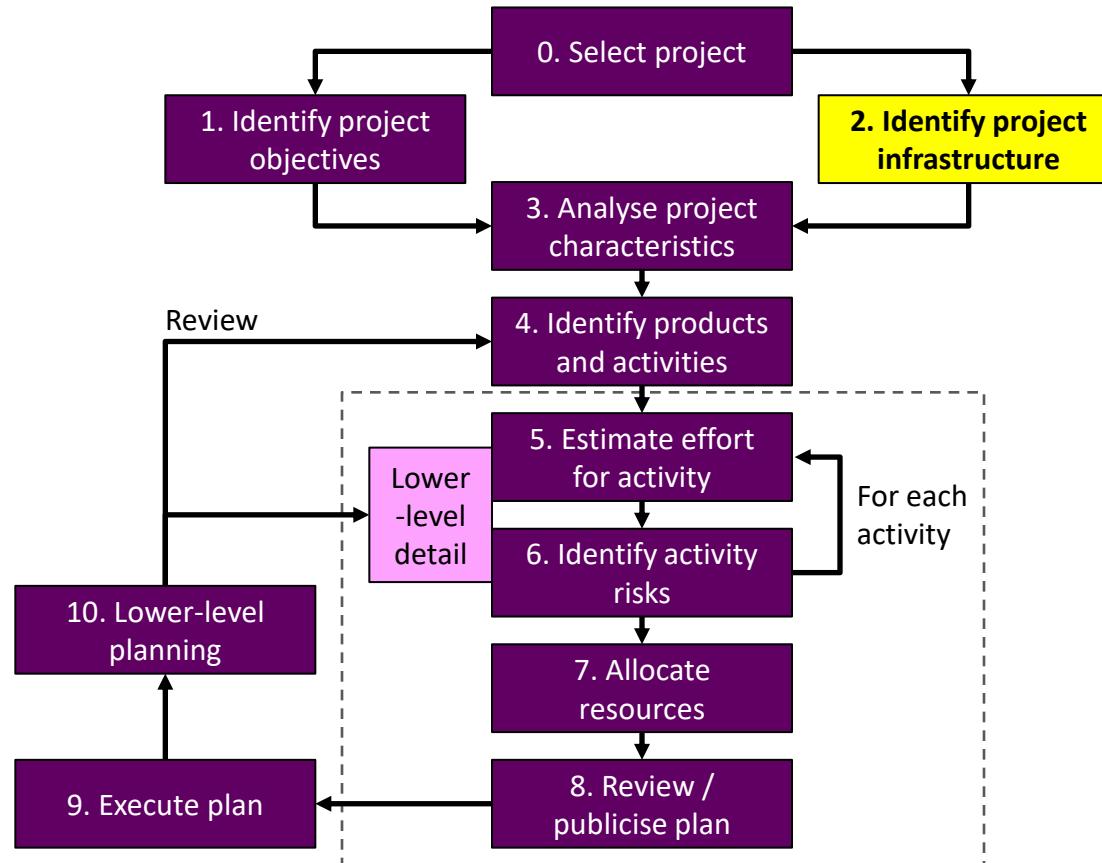
- who will be affected/involved in the project?

Step 1: identify project objectives

- 1.4 Modify objectives in the light of stakeholder analysis
 - do we need to do things to win over stakeholders?
- 1.5 Establish methods of communication with all parties
 - how do we keep in contact?



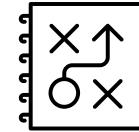
Step Wise - an overview



Step 2: identify project infrastructure

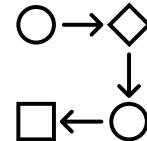
- 2.1 Establish link between project and any strategic plan

– why did they want the project?



- 2.2 Identify installation standards and procedures

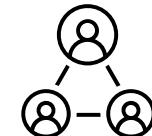
– what standards do we have to follow?



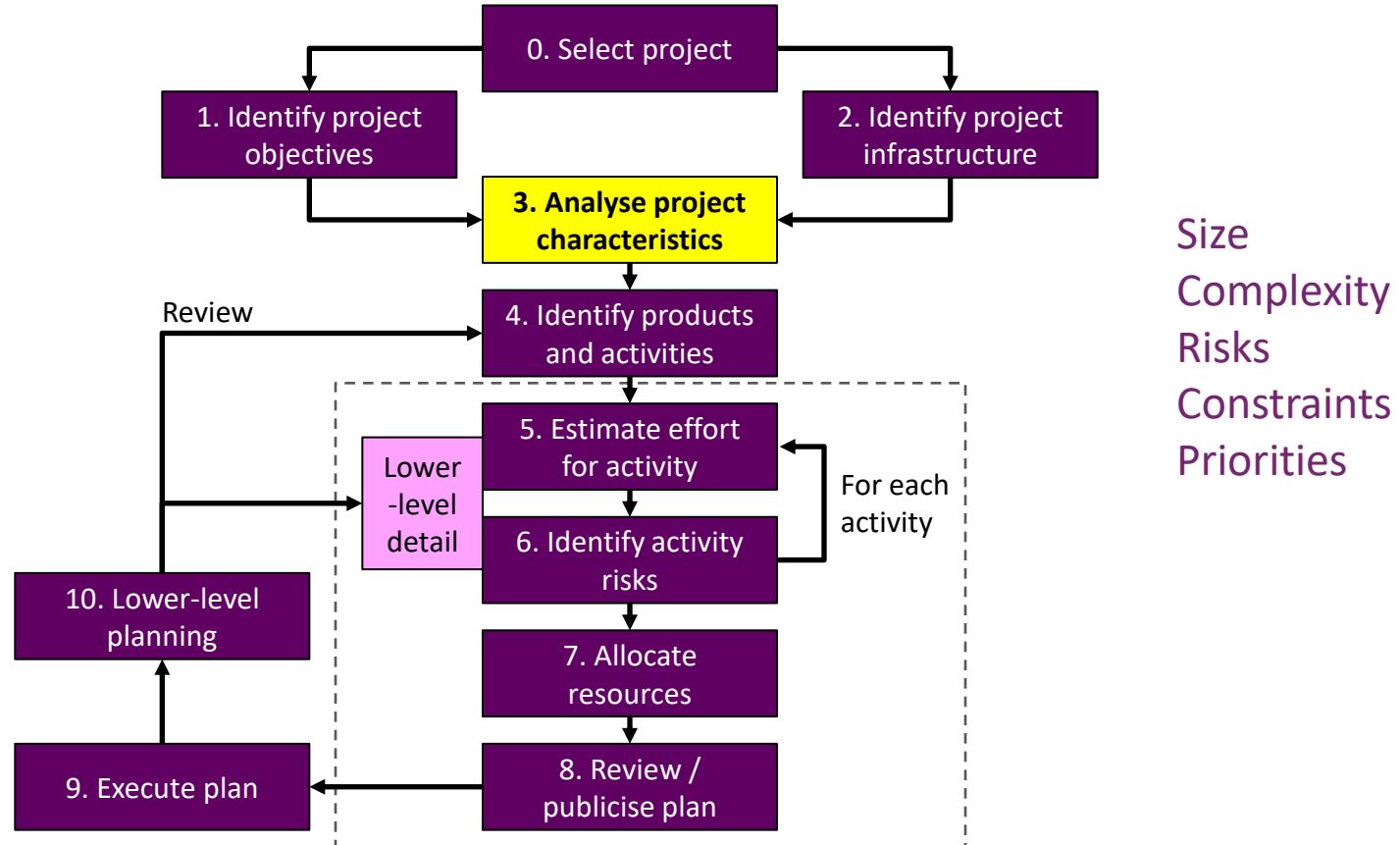
- 2.3. Identify project team organisation

– where do I fit in?

– members roles and responsibilities

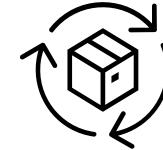
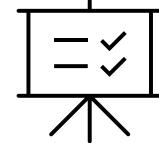


Step Wise - an overview



Step 3: analyse project characteristics

- 3.1 Distinguish the project as either **objective-based** or **product-based**.



- Is there more than one way of achieving success?

Step 3: analyse project characteristics

- 3.1 Distinguish the project as either **objective-based** or **product-based**.
 - Is there more than one way of achieving success?
- 3.2 Analyse other **project characteristics** (e.g., quality-based)
 - what is different about this project?



Step 3: analyse project characteristics

- 3.3 Identify high level **project risks**

- what could go wrong?
- what can we do to stop it?

What can be a high-risk scenario in software development projects?

Step 3: analyse project characteristics

- 3.3 Identify high level project risks
 - what could go wrong?
 - what can we do to stop it?
- 3.4 Consider **user requirements** concerning implementation

Functionalities
Usability
Performance
Integration with other systems
Security, privacy, etc.

Step 3: analyse project characteristics

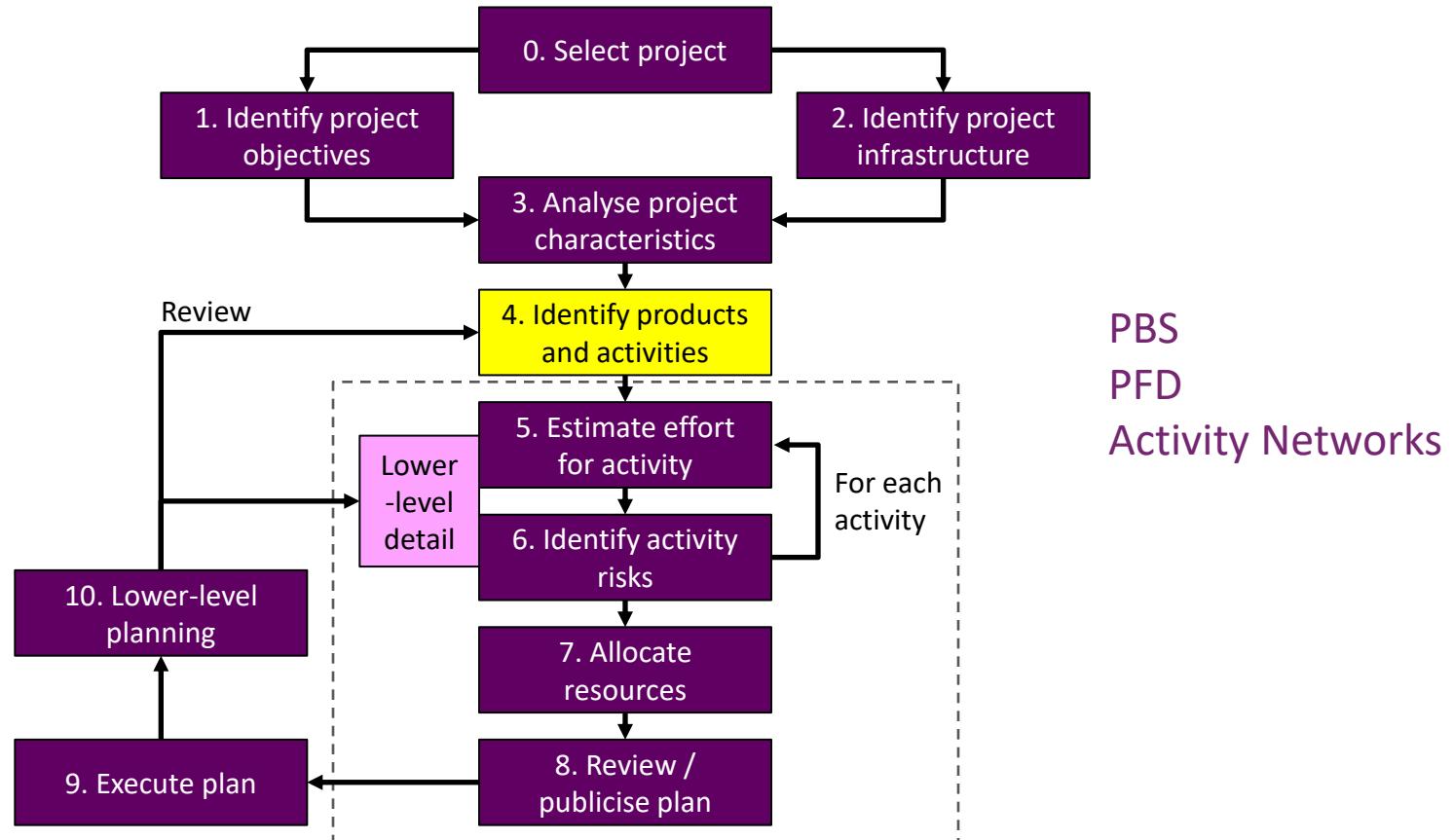
- 3.3 Identify high level project risks
 - what could go wrong?
 - what can we do to stop it?
- 3.4 Consider user requirements concerning implementation
- 3.5 Select **development methodology** and **life cycle** approach:
 - Waterfall? Incremental? Agile? etc.

Complexity
Changeability
Delivery

Step 3: analyse project characteristics

- 3.3 Identify high level project risks
 - what could go wrong?
 - what can we do to stop it?
- 3.4 Consider user requirements concerning implementation
- 3.5 Select development methodology and life cycle approach:
 - Waterfall? Incremental? Agile? etc.
- 3.6 Review overall **resource estimates**
 - does all this increase the cost?

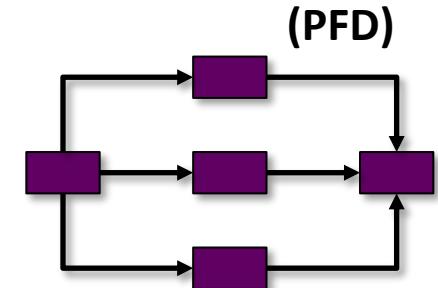
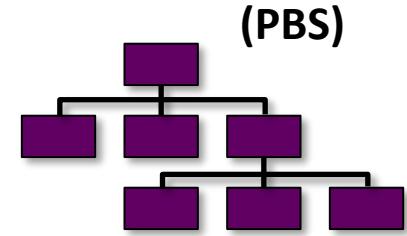
Step Wise - an overview



Step 4: identify products and activities

- 4.1 Identify and describe project **products**
 - what do we have to produce in terms of both customer deliverables and intermediate artefacts?
 - Typically, a product hierarchy that can be documented in a Product Breakdown Structure (PBS)

- 4.2 Document generic **product flows**
 - Some products need one or several other products to exist before they can be created
 - E.g., specification must exist before design, and design before code
 - **Product Flow Diagrams (PFDs) are used to represent these relationships**



Step 4: identify products and activities

- 4.3 Recognise **product instances**

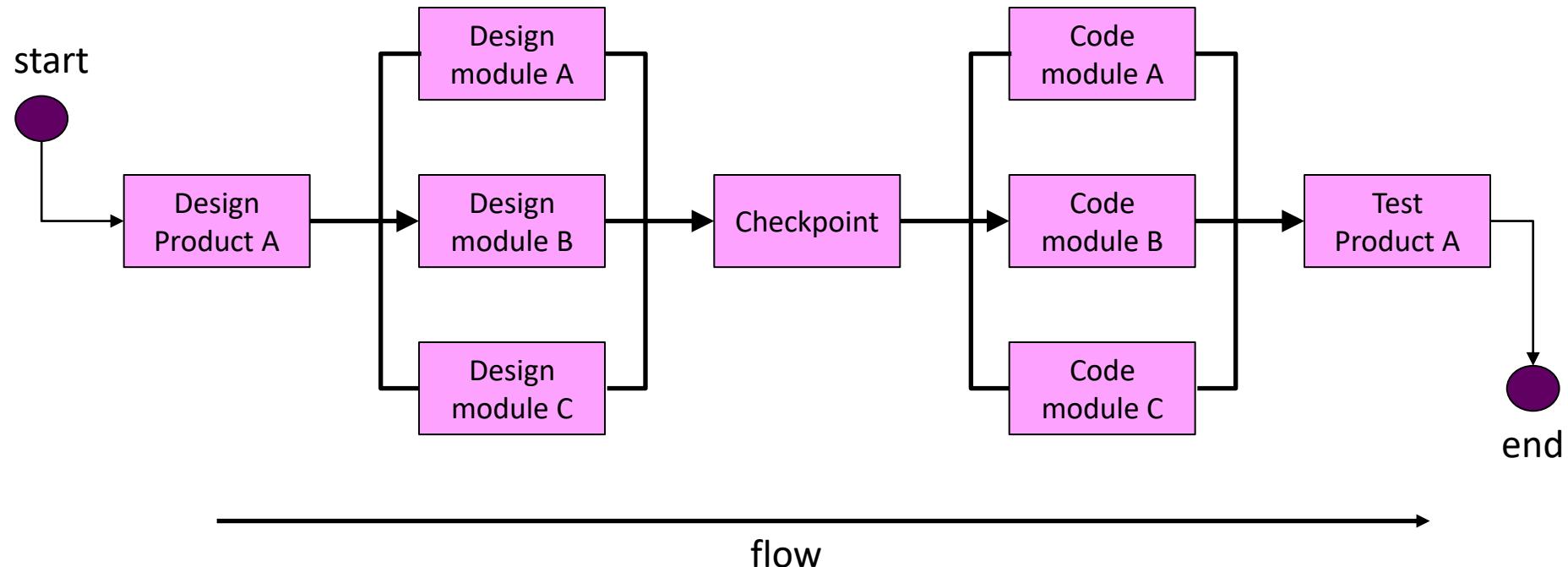
- The PBS (Product Breakdown Structure) and PFD will probably have identified generic products (e.g. ‘software modules’) for which specific instances can be identified (e.g., ‘module A’, ‘module B’, etc.)
- In many cases this has to be left to later, more detailed, planning

- 4.4 Produce ideal **activity network**

- **Identify the activities** needed to create each product in the PFD (Product Flow Diagrams)
 - More than one activity might be needed to create a single product
 - Hint: Identify activities by verb + noun, avoiding ‘produce X’
- **Draw up activity network**

Step 4: identify products and activities

An activity network: activities order and flow (arrows)



Step 4: identify products and activities

- 4.3 Recognise **product instances**

- The PBS (Product Breakdown Structure) and PFD will probably have identified generic products (e.g. 'software modules') for which specific instances can be identified (e.g., 'module A', 'module B', etc.)
 - In many cases this has to be left to later, more detailed, planning

- 4.4 Produce ideal **activity network**

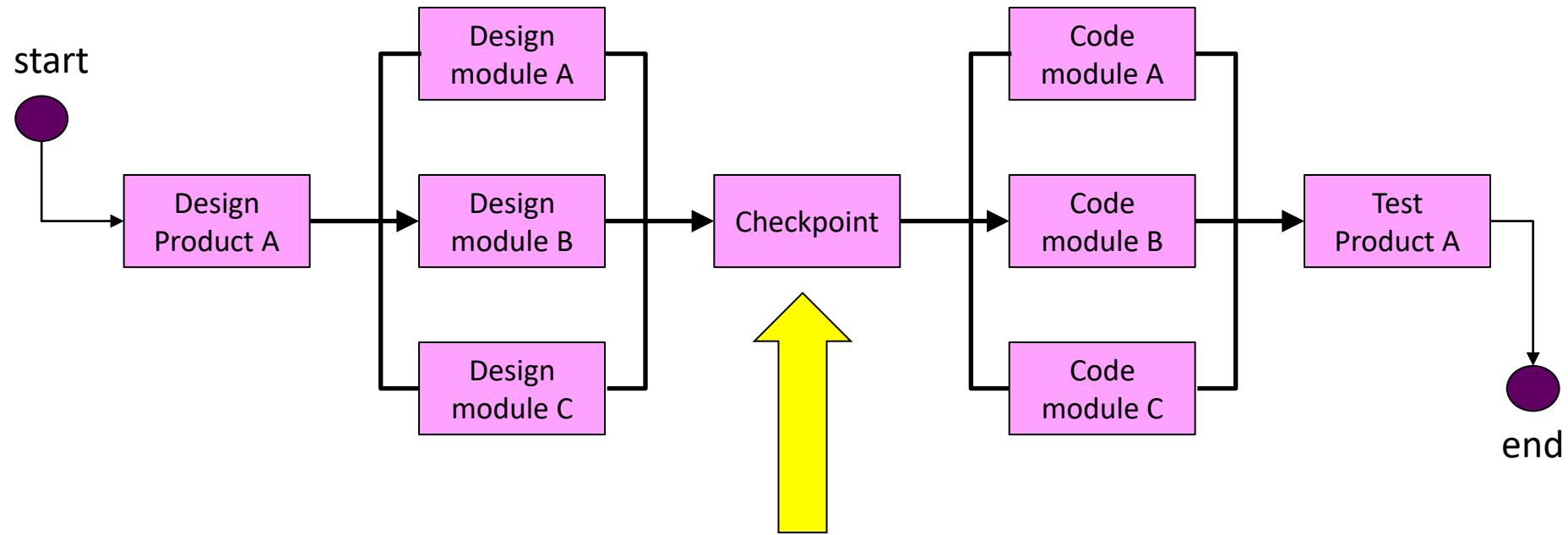
- Identify the activities needed to create each product in the PFD (Product Flow Diagrams)
 - More than one activity might be needed to create a single product
 - Hint: Identify activities by verb + noun, avoiding 'produce X'
 - Draw up activity network

- 4.5 Add **stages & checkpoints/milestones**

- Ensure that intermediate products are compatible, and project can proceed without major incompatibility risks

Step 4: identify products and activities

An activity network: activities order and flow (arrows)



Step 4: identify products and activities

- **Products:**
- The result of an activity
 - Could be (among other things)
 - physical thing (installed pc)
 - a document (logical data structure)
 - a person (trained user)
 - a new version of an old product (updated software)

Step 4: identify products and activities

- **Products:**
- The result of an activity
 - Could be (among other things)
 - physical thing (installed pc)
 - a document (logical data structure)
 - a person (trained user)
 - a new version of an old product (updated software)
- The following are normally NOT products:
 - activities (e.g. training)
 - events (e.g. interviews completed)
 - resources and actors (e.g. software developer)

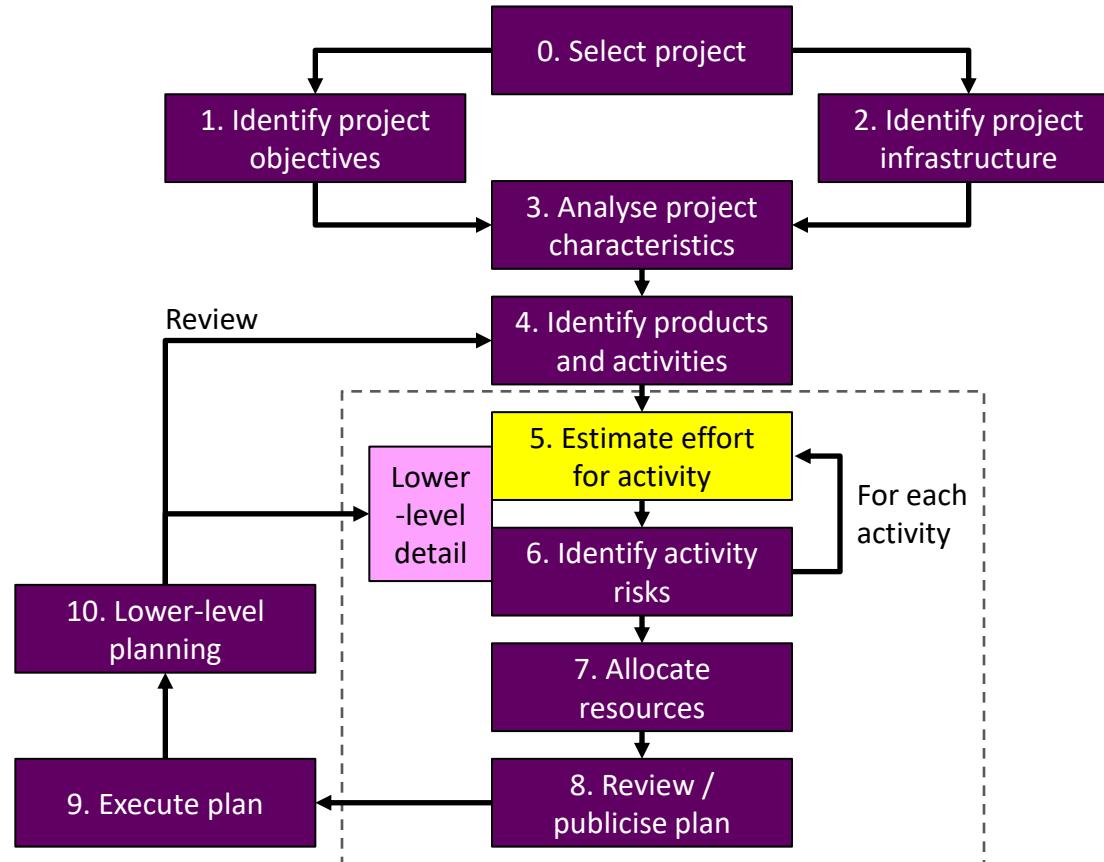
Step 4: identify products and activities

- Consider a final-year project whose objective is to develop a web application that Aston students can use to advertise items for sale
- What are some of the main project products?

Step 4: identify products and activities

- Consider a final-year project whose objective is to develop a web application that Aston students can use to advertise items for sale
- What are some of the main project products?
- **Possible solution includes:**
 - Project definition form
 - Project objectives
 - Preliminary work plan
 - Progress report
 - Final report
 - Requirements document
 - Project diary
 - Design iteration 1
 - Design iteration 2

Step Wise - an overview



Step 5: estimate effort for each activity

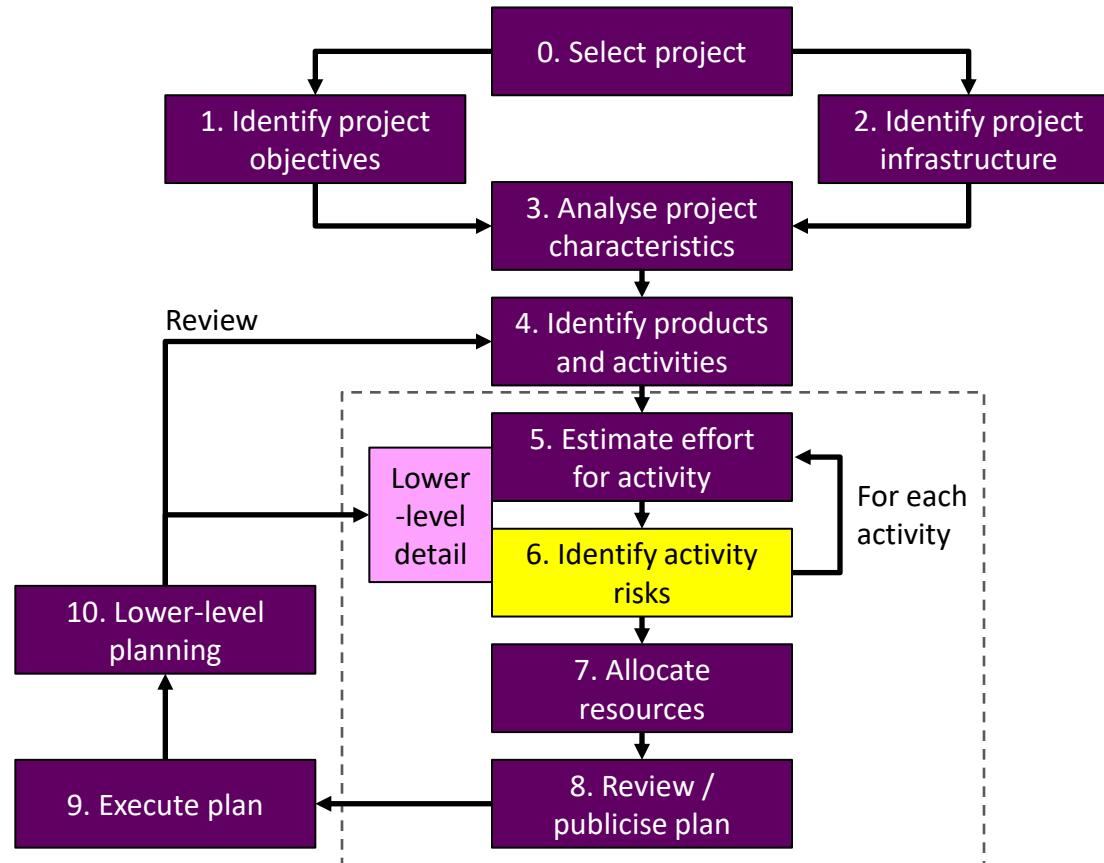
- **5.1 Carry out bottom-up estimates**

- Distinguish carefully between effort and elapsed time
- Effort = number of staff-days, staff-weeks, etc. required to complete a task
- Elapsed time = calendar time between task start time and task end time
- Example: If 2 people work on the same task for 5 days without any interruption, then the effort is 10 staff-days and the elapsed time is 5 days.

- **5.2. Revise plan to create controllable activities**

- break up very long activities into a series of smaller, easier to monitor & control ones
- bundle up very short activities

Step Wise - an overview

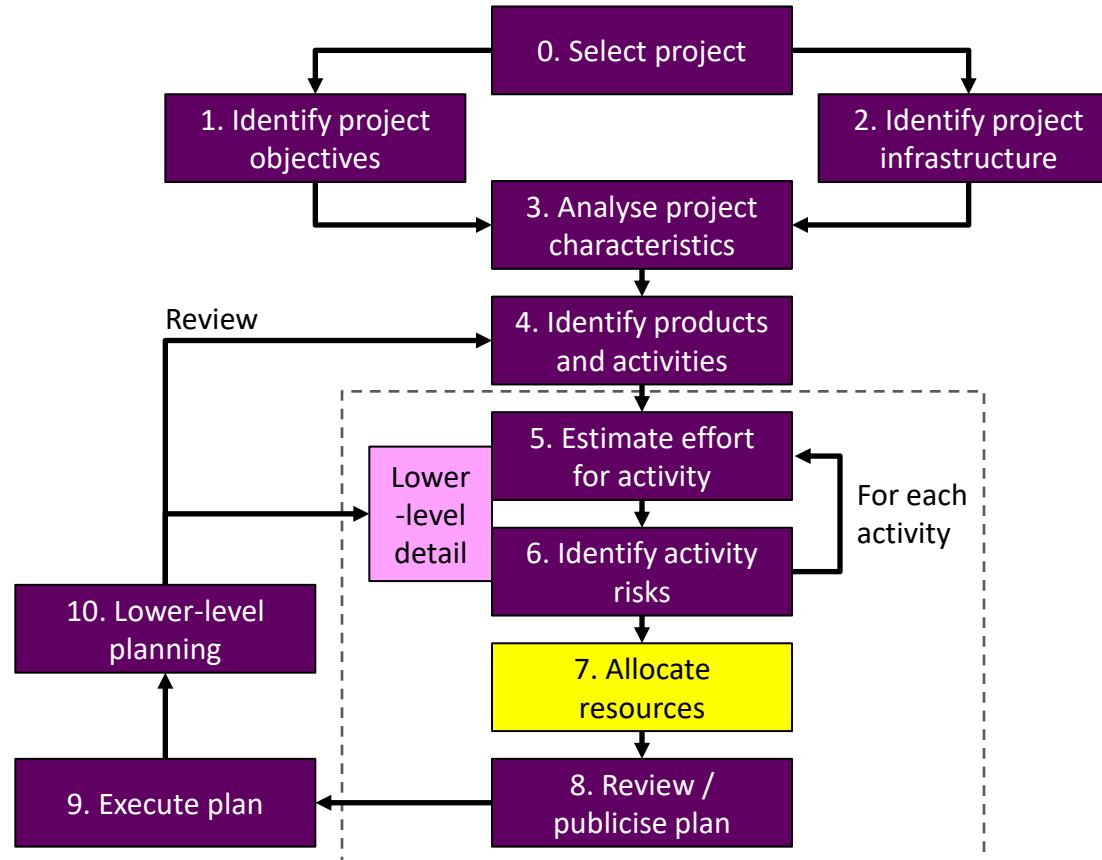


Step 6: identify activity risks

(to be studied in more detail later in the module)

- 6.1. Identify and quantify risks for activities
 - Estimate damage if risk occurs (measured in time lost or money wasted)
 - Estimate likelihood of risk occurring
- 6.2. Plan risk reduction and contingency measures
 - **risk reduction:** activity to eliminate/reduce likelihood of risk occurring
 - **contingency:** action to take if risk does occur in order to reduce impact

Step Wise - an overview

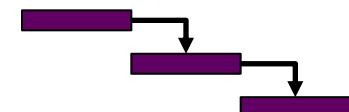


Step 7: allocate resources

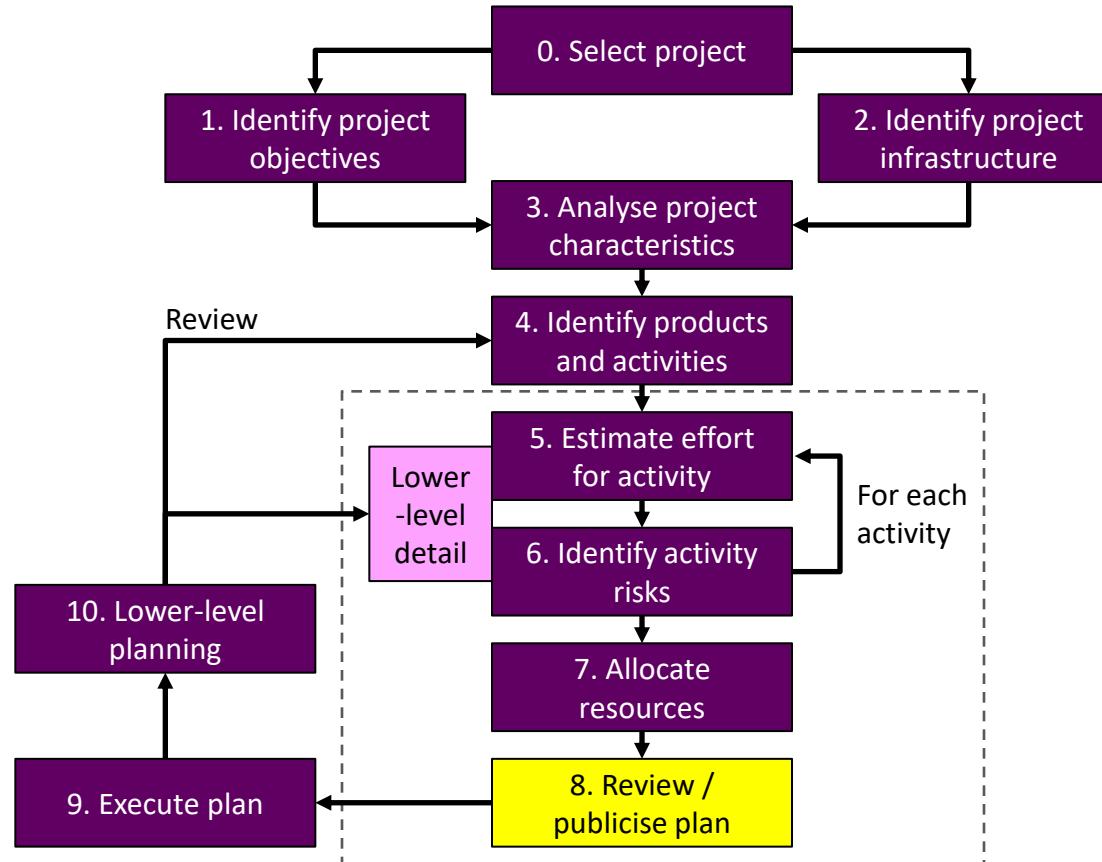
(to be studied in more detail later in the module)

- 7.1 Identify and allocate resources to activities
 - Record type of staff needed for each activity
 - Identify staff available and allocate to tasks provisionally
- 7.2 Revise plans and estimates to take into account resource constraints
 - Staff not being available until a later date
 - Staff needed for other, non-project activities

*The result of these steps is typically a **Gantt chart** (devised in the 1910s by American management consultant Henry Gantt)*



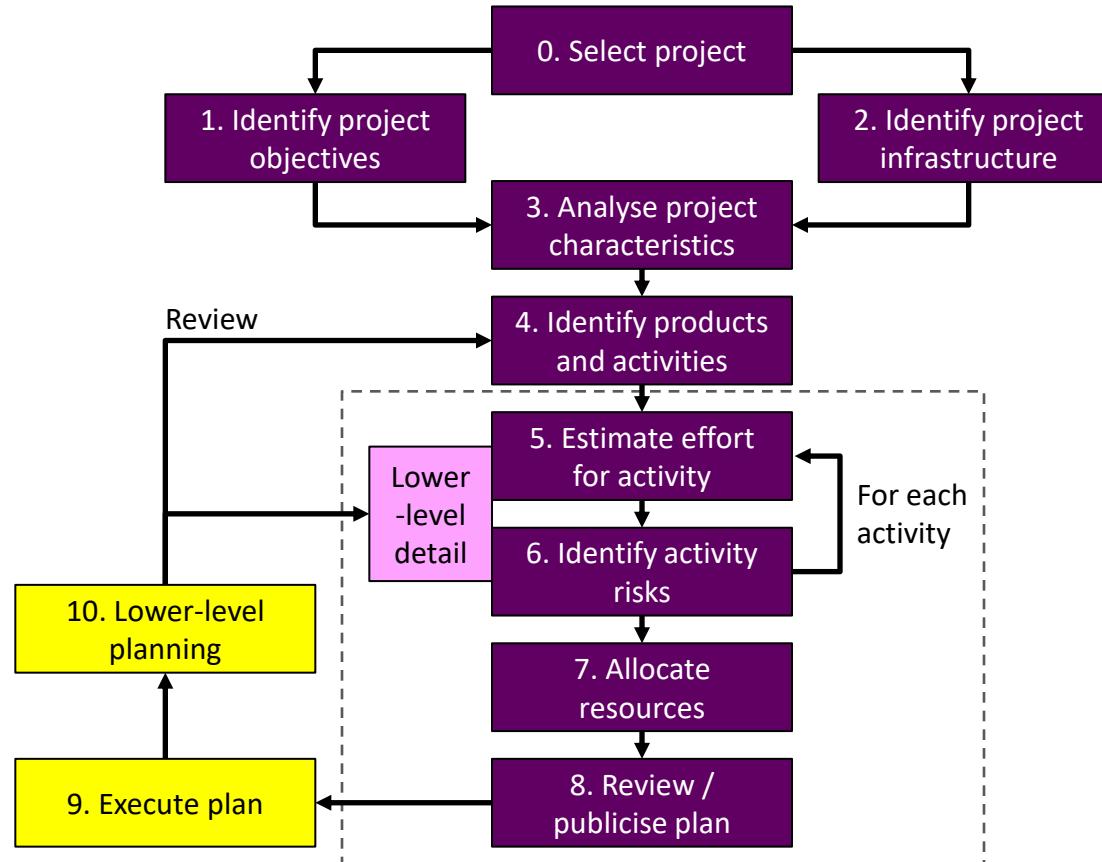
Step Wise - an overview



Step 8: review / publicise plan

- 8.1 Review quality aspects of project plan
 - Ensure that quality criteria are defined for all activities, i.e., criteria that need to be fulfilled to sign off an activity as completed
 - Helps identify and correct problems with individual activities at an early stage
- 8.2 Document plan and obtain agreement
 - Ensure that all parties involved understand and “sign up” to the commitments required of them in the plan

Step Wise - an overview



Steps 9 & 10

- Step 9: execute plan
- Step 10: lower-level planning
- Detailed plans are required and feasible for the current activities
- Detailed plans for later project stages are often not possible
 - Estimates of effort, start dates, etc. are associated with confidence intervals that are too large

Outline

- Frameworks for software project planning
 - Motivation
 - Step Wise project planning
 - **PRINCE2 project planning**
- Selection of software project approaches
- Effort estimation for software projects
- Activity planning and resource allocation

Overview of PRINCE2

- **PRINCE2** acronym: **P**rojects **I**N **C**ontrolled **E**nvironments, version **2**
- Standardised set of process-based project management procedures, originally intended for ICT projects
- Devised under the sponsorship of UK's Office of Government Commerce, and used primarily in the UK
- 1996 revision of earlier 1989 PRINCE framework – recently in 2017, emphasis on agility

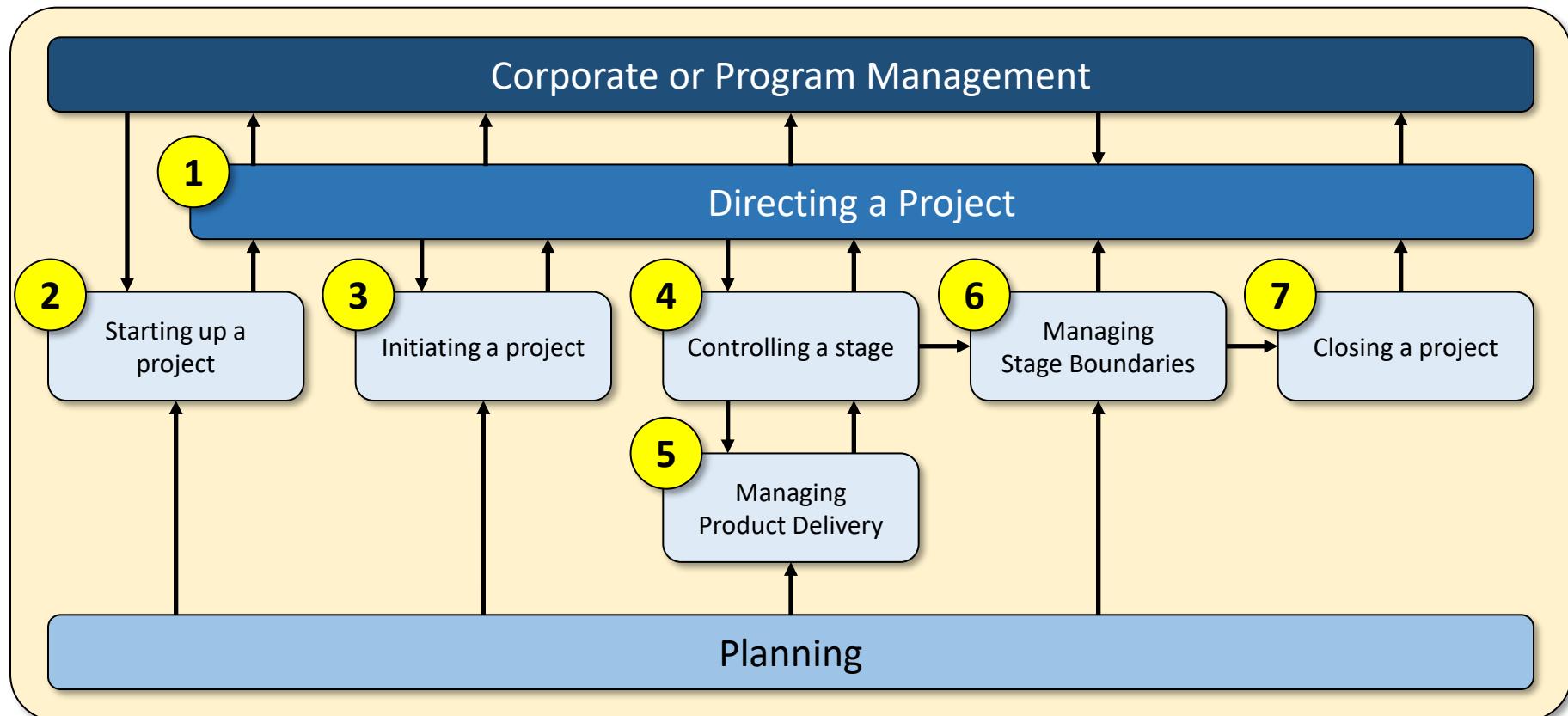
Overview of PRINCE2

- Large projects are divided into subsets of project activities organised into stages
 - Stages are managed as a sequence of individual units
 - Project Board authorises project manager to execute a stage only after the plans for that stage are approved
 - The end of a stage is a decision point when the Project Board reviews progress to date and decides that project remains feasible and is worth continuing
- Project planning activities are placed into seven processes

PRINCE2 processes

1. **Directing a Project** covers activities by the Project Board
2. **Starting up a Project** comprises activities concerned with reaching a position where detailed project planning can begin
3. **Initiating the Project** refers to activities associated with producing an overall plan for the project
4. **Controlling a Stage** includes the activities carried out by the project manager once a stage has been authorised
5. **Managing Product Delivery** covers activities that involve communication between the project manager and team managers
6. **Managing Stage Boundaries** includes activities associated with the creation of next stage plans & updates to overall project plan
7. **Closing the Project** includes end-of-project activities

PRINCE2 processes



Comparison to Step Wise

- Both PRINCE2 and Step Wise focus on **how the project is organised and controlled** — defining roles, stages, and planning activities.
- Both are originally **product-based** approaches
- PRINCE2 is process-based, while Step Wise places more emphasis on techniques used in project planning
 - Techniques similar to those employed by Step Wise can be used in PRINCE2, but they are not prescribed by the framework
- Often also compared to PMBOK - *Project Management Body of Knowledge* (PMI, PMP certification)

What's next...

- Frameworks for software project planning (Week 4 lecture)
 - Step Wise vs Prince 2 (further reading and knowledge check quiz)
 - This week's tutorial (Week 4) – Planning activities
 - Introduction to activities and deliverables (products)
- Next week (Week 5 lecture)
 - Selection of software development approaches
 - Activity networks (theory and practice)
 - Critical path analysis (CPA)
 - Resource allocation considerations



Aston University

BIRMINGHAM UK

Software Project Management

Unit 3: Software Project Planning (1)

Thais Webber
Richard Lee





Aston University

BIRMINGHAM UK

Software Project Management

Unit 3: Software Project Planning (2)

Thais Webber
Richard Lee



Outline

- Frameworks for software project planning
- Selection of software project approaches
 - Analysing other project characteristics
 - Development process models
- Effort estimation for software projects
- Activity planning and resource allocation

Selection of software project approaches

- Look at risks and uncertainties, e.g.
 - Are requirements well understood?
 - Are technologies to be used well understood?
- Look at the type of application being built, e.g.
 - Information system? Embedded system?
 - Safety-critical software? Mobile application?
- Approaches with heavy **structure** versus Approaches with pressure about **speed** of delivery?
- Software development **process** that best fit the needs?

Structured approaches

- Also called heavyweight approaches
- Step-by-step methods where each step and intermediate product is carefully defined
- Emphasis on getting quality right first time
- Example: use of UML (i.e., Unified Modelling Language)

Agile methods

- Emphasis on speed of delivery rather than documentation
- RAD – Rapid application development emphasised use of quickly developed prototypes
- SCRUM (Unit 4)

Outline

- Frameworks for software project planning
- Selection of software project approaches
 - Analysing other project characteristics
 - **Development process models**
- Effort estimation for software projects
- Activity planning and resource allocation

Choice of (development) process models

- Several widely used development process **models**
 - ‘**waterfall**’ model /sequential / also known as ‘one-shot’, ‘once-through’
 - **prototyping** / evolutionary development model / ‘try-feedback-refine’, ‘learn by doing’
 - **incremental** delivery model /‘build in pieces’
 - **agile** methods such as extreme programming and SCRUM / ‘iterative-collaborative-adaptive’

Waterfall versus agile

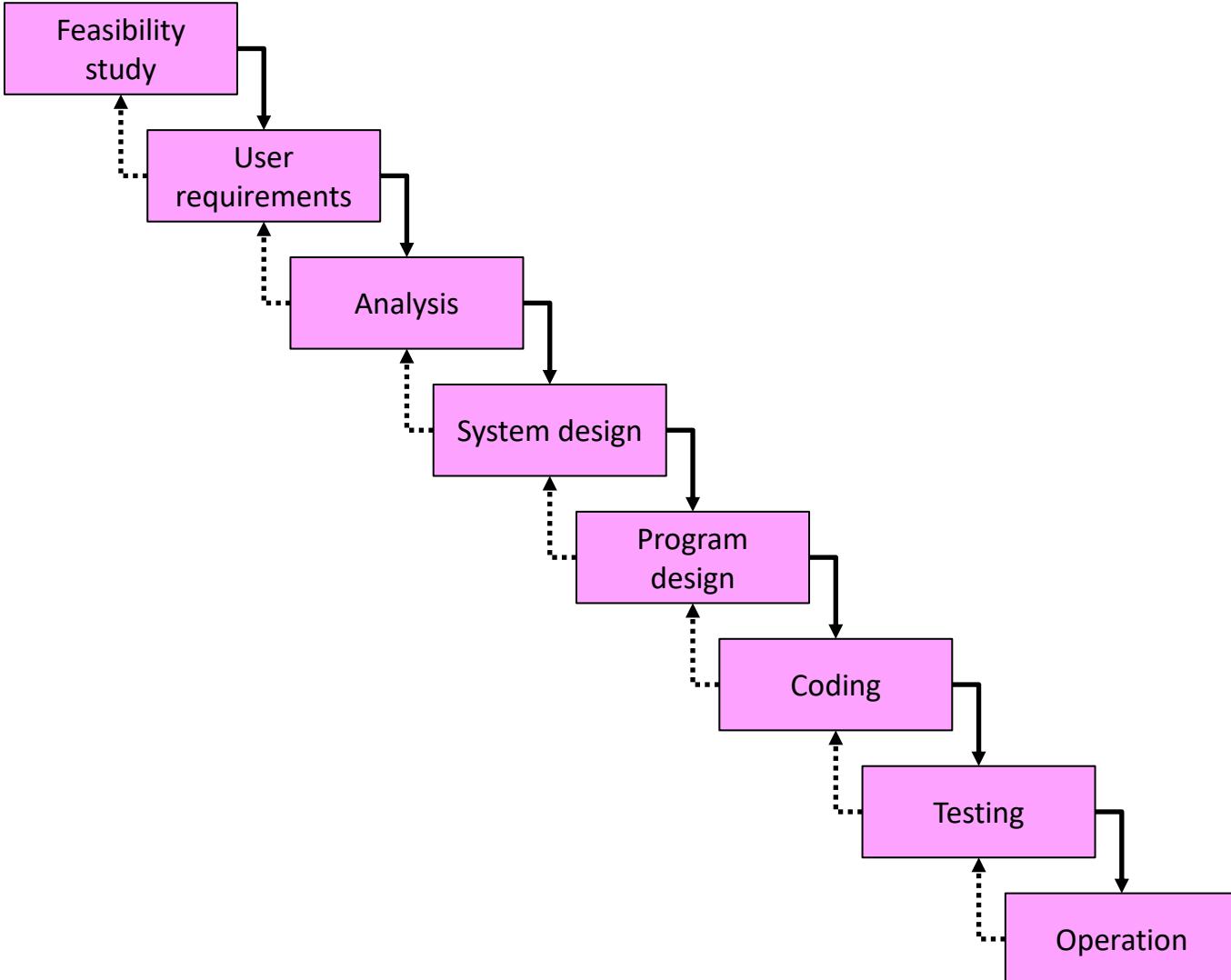
Predictive approach

- "Plan the work, work the plan"
- Everything must be planned
- More control and visibility

Empiric approach

- "Fail fast, fail safe"
- Based on experience
- More flexibility and learning

The waterfall model



- The 'classical' model
- Imposes structure
- Every stage signed off
- Limited scope for iteration

Works well when end-user requirements are clearly defined and stable - no change throughout the project

Waterfall advantages

- Simple and easy to understand and use
- Easy to manage due to the rigidity of the model; each phase has specific deliverables and a review process
- Phases are processed and completed one at a time
- Works well for smaller projects where requirements are very well-understood
- Clearly-defined stages
- Well-understood milestones
- Easy-to-arrange tasks
- Process and results are well-documented

Waterfall disadvantages

- No working software is produced until late during the life cycle
- High amounts of risk and uncertainty
- Poor model for long and ongoing projects
- Not suitable for the projects where requirements are at a moderate to high risk of changing; risk and uncertainty are high with this process model
- It is difficult to measure progress within stages
- Cannot accommodate changing requirements
- Adjusting scope during the life cycle can end a project
- Integration is done as a big-bang at the very end, which doesn't allow identifying any technological or business bottleneck or challenges early

Projects suited for agile delivery

- The most appropriate projects for agile are ones with aggressive deadlines, high degree of complexity, and high degree of novelty (uniqueness) to them

- Novelty?



- Urgency



- Complexity



Focus is on delivering the
highest business value in the
shortest time

Agile methods tackling disadvantages

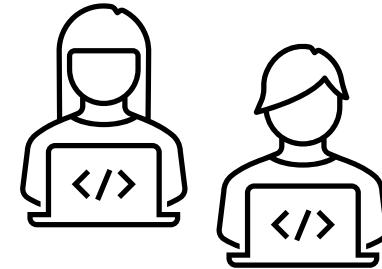
- Agile aims to address several disadvantages of structured development methods:
 - **large amounts of documentation** which can be largely unread
 - need to keep **documentation up to date**
 - **communication stifled** by division into specialist groups and need to follow procedures
 - **user exclusion** from the decision process
 - **long lead times** to deliver anything
 - etc.

Extreme programming (XP)

A type of agile development process

- Characteristics:

- Increments of one to three weeks
- Customer(end users) can suggest improvement at any point
- Elimination of distinction between design and building of software
- Code developed to meet current needs only
- Frequent refactoring to keep code structured
- Developers work in pairs
- Test cases and expected results devised *before* software design
- After testing of increment, test cases added to a consolidated set of test cases – reused as part of continuous integration



Frequent refactoring
Pair programming
Test-driven development (TDD)
Continuous integration

Limitations of extreme programming (XP)

- Reliance on availability of **high-quality developers** (e.g. experience, good technical skills)
- **Dependence on personal knowledge** – after development, knowledge of software may decay making future development difficult
- **Rationale for decisions may be lost**, e.g., which test case checks a particular requirement
- **Reuse of existing code less likely**

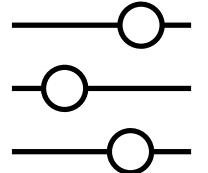
Prototyping and evolutionary methods

- Main types
 - ‘**throw away**’ prototypes – used to learn about an area of uncertainty
 - **evolutionary prototypes** – the prototype is developed and modified until it becomes the operational system
- What is being prototyped?
 - human-computer interface – front end
 - elements/subsets of functionality – back end

Prototype terms – key points

• Prototype

- a model for acquiring requirements
- a working model built to learn how a work system could operate
- like physical models in many engineering applications



• Throwaway Prototype

- designed to be discarded
- suitable when the final design is unclear; useful for comparing alternatives

• Evolutionary Prototype

- designed to be adapted for normal use
- suitable when the final design is clear
- built using the intended platform

Reasons for prototyping

- Learning by doing
- Improved communication
- Improved user involvement
- A feedback loop is established
- Reduces the need for documentation
- Reduces maintenance costs due to changes before the application goes live
- Prototype can be used for producing expected results

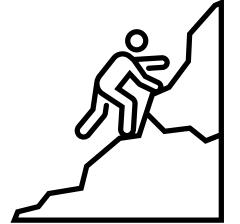
Dangers of prototyping

- Users may misunderstand the role of the prototype
- Their expectations may get too high
- Lack of project control and standards
- Additional expense of building prototype
- Focus on usable interface could be at expense of machine efficiency



Incremental delivery

- The application to be delivered is broken down into a number of components called **lots** that provide **immediate value** to the customers
- Each component is developed as a separate **increment**



Incremental - pros and cons

- Several important advantages:
 - feedback from early stages used in developing later stages
 - easier to cope with changing requirements
 - user gets some benefits earlier
 - project may be put aside temporarily
- BUT there are some possible disadvantages:
 - loss of economy of scale – some costs will be repeated
 - software breakage – later increments may change earlier increments

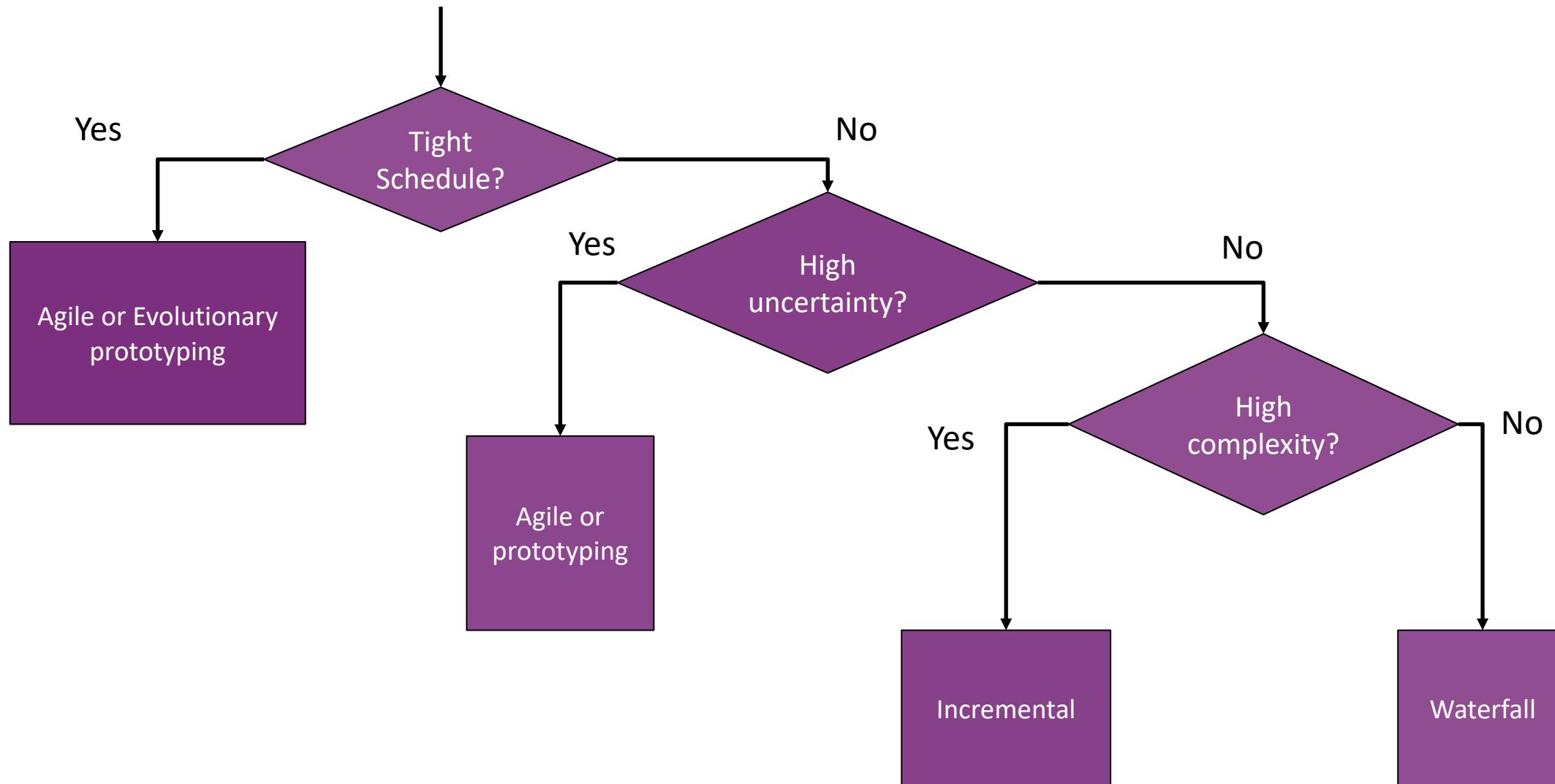
Typical increment plan

- Each lot should deliver some benefits to the user
- Some lots will be dependent on others, hence there is a natural ordering of these lots

Choosing a model

- If uncertainty is high:
 - Agile or prototyping
- If complexity is high but uncertainty is not:
 - Incremental delivery
- If uncertainty and complexity are both low:
 - Waterfall
- If schedule is tight:
 - Agile or evolutionary prototype

Choosing a model for SW development

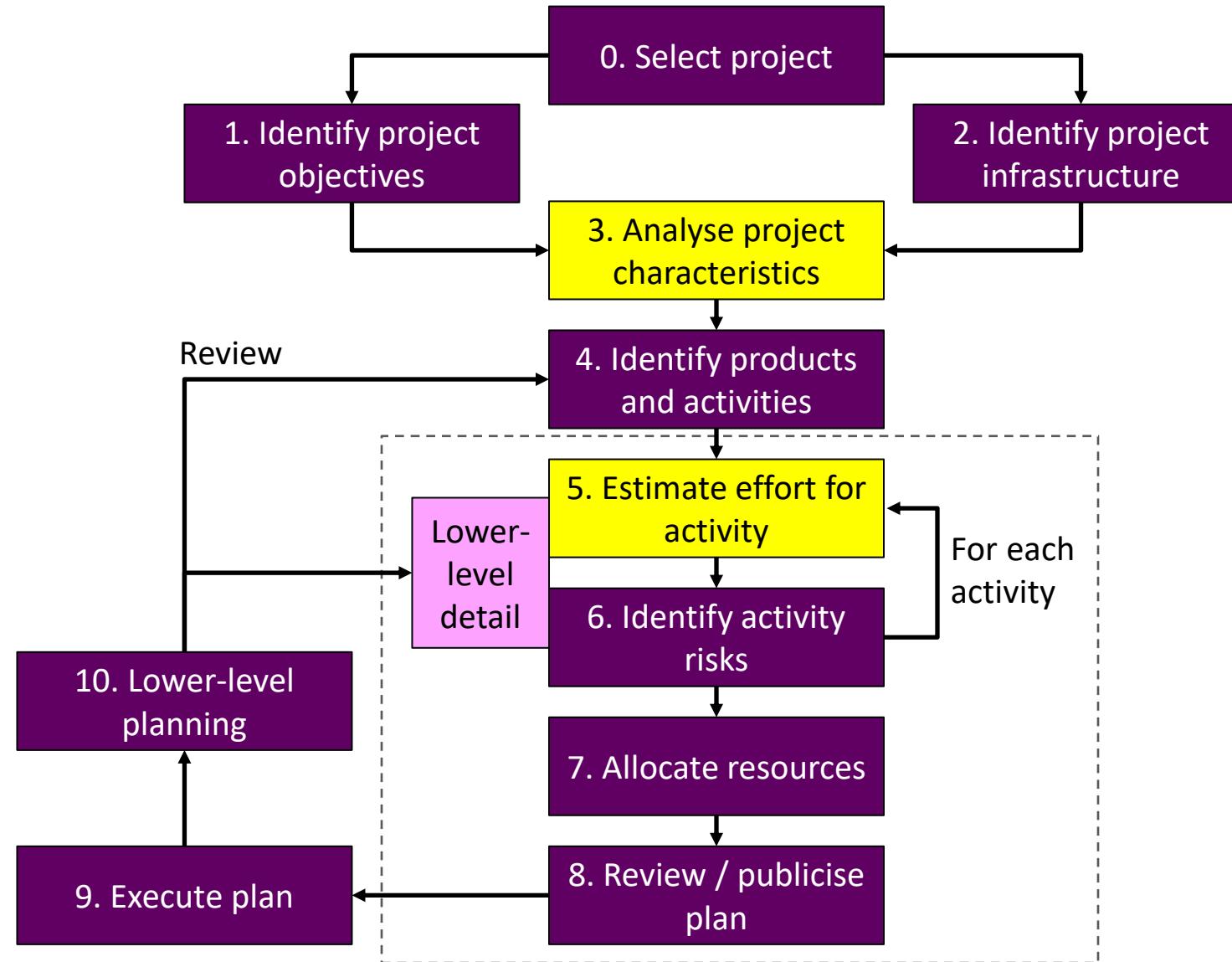


Outline

- Frameworks for software project planning
- Selection of software project approaches
- **Effort estimation for software projects**
- Activity planning and resource allocation

Steps 3 and 5 of Step Wise - Effort

***Effort**
time,
resources,
and cost



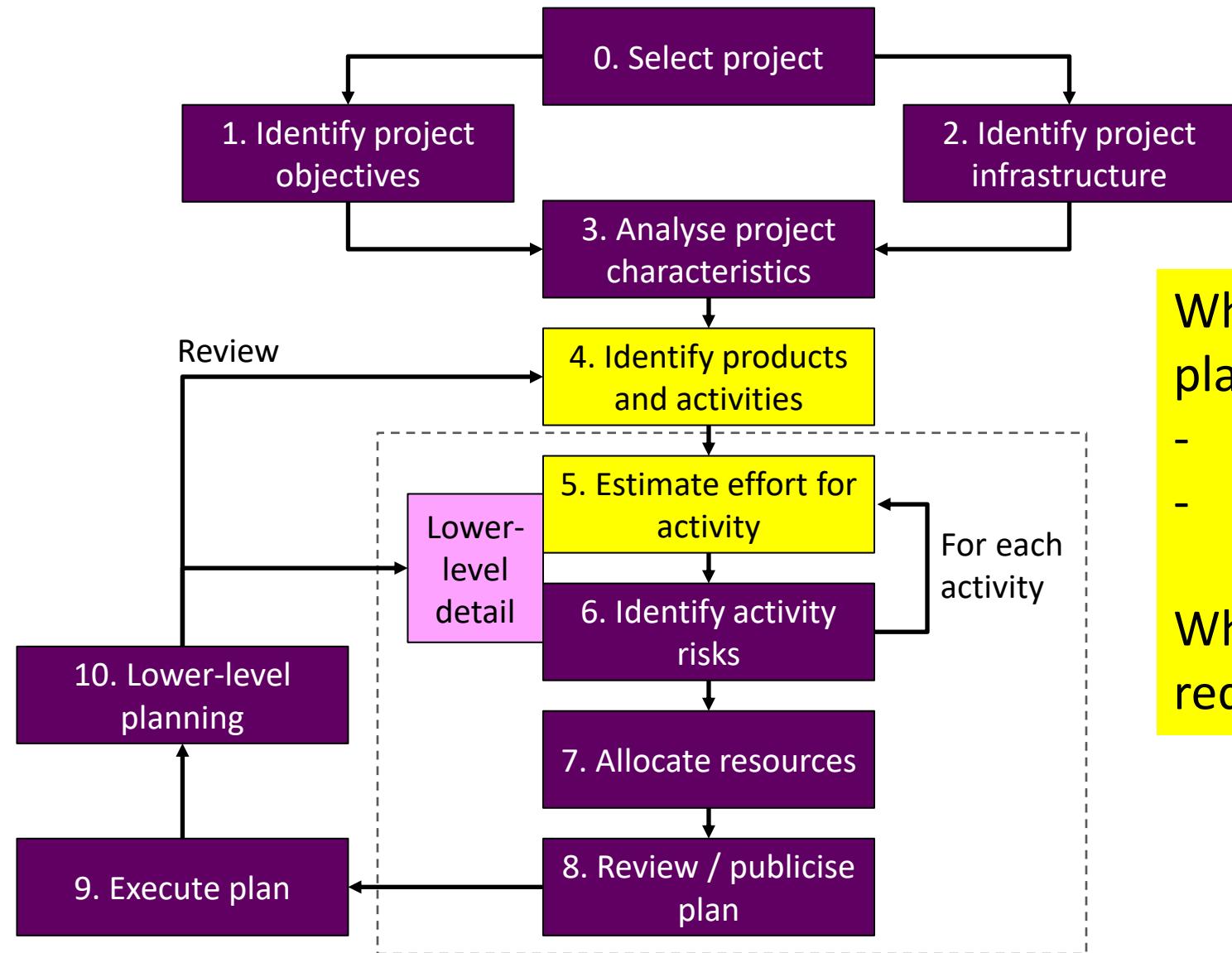
Goal:
accurate and
realistic estimates

- Size
- Dependencies
- Skill levels

Outline

- Frameworks for software project planning
- Selection of software project approaches
- Effort estimation for software projects
- Activity planning and resource allocation
 - Activity planning
 - Resource allocation

Steps 4 and 5 of Step Wise



What needs to be planned?

- Scope
 - Tasks

What is the effort required?

Planning your project

- **Objective:** produce a schedule that indicates start and completion times for each project activity
 - Ensuring that appropriate resources will be available when required
 - Avoiding resource overload
 - Derivation of a timed cash flow forecast
 - Project re-planning to correct drift from the target
- This enables:
 - The WHEN/WHO for each activity
 - Resources ready when needed
 - When money will be needed?
 - Schedule allows adjusting the plan when needed

Approaches (3)

- **Activity-based**

- Draw-up a Work Breakdown Structure listing the work items (activities) needed based on an analysis of similar past products

- **Product-based***

- Used in **PRINCE2** and **Step Wise**
 - **List** the deliverable and intermediate products of project – **Product Breakdown Structure (PBS)**
 - **Identify the order** in which products have to be created – **Product Flow Diagram (PFD)**
 - **Work out the activities** needed to create the products – **e.g. creating Activity Networks**

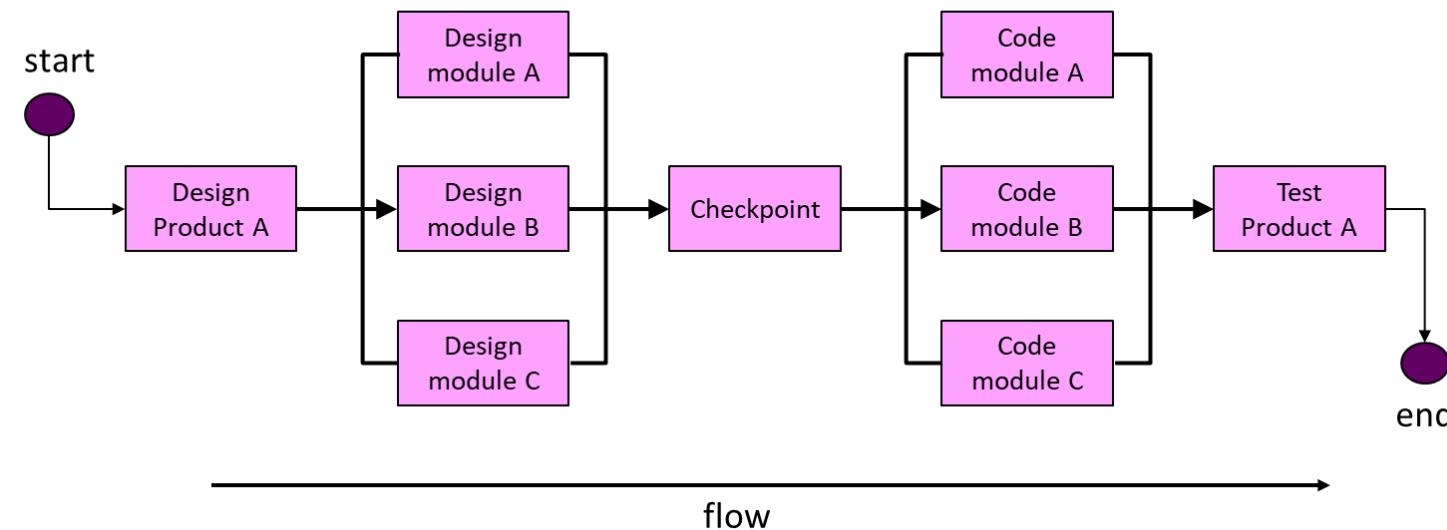
- **Hybrid**

- start from a simple list of final deliverables (which are product specific) and apply the **activity-based approach to each deliverable**

Activity networks

- These networks help us to:
 - Assess the feasibility of the planned project completion date
 - Identify when resources will need to be deployed to activities
 - Calculate when costs will be incurred
 - Globally, co-ordinate and motivate of the project team

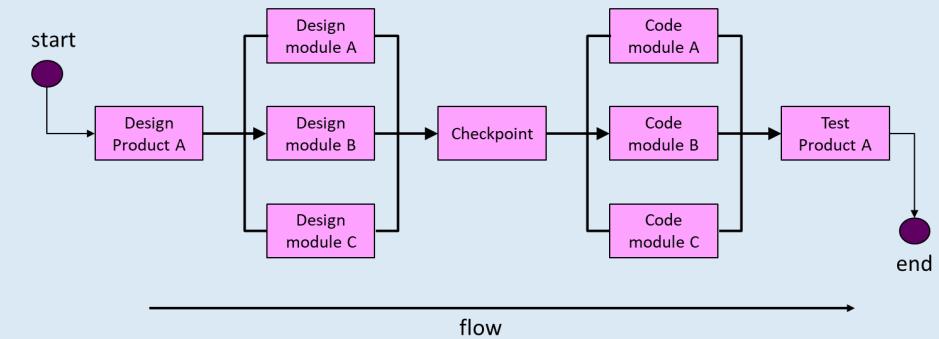
An activity network: activities order and flow (arrows)



Activity networks

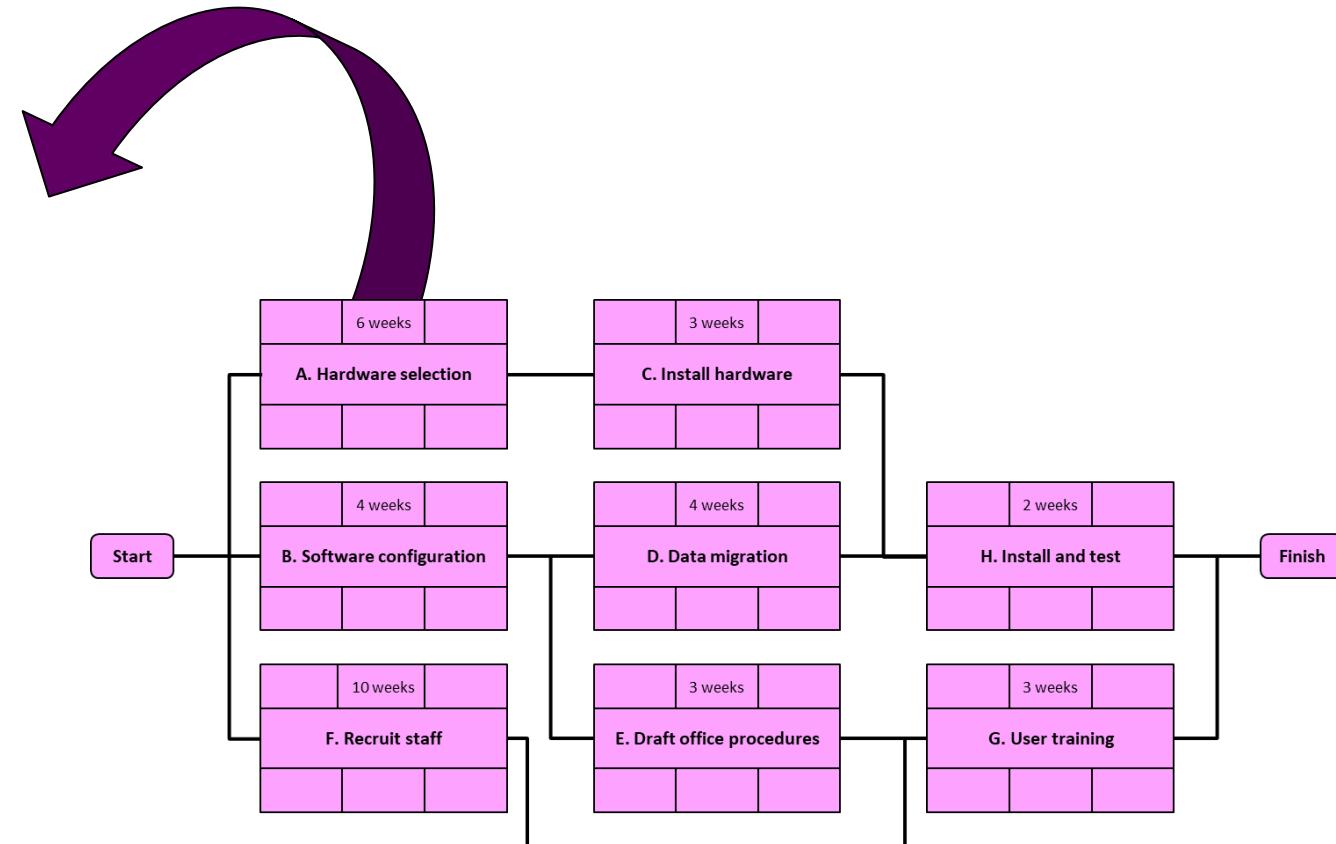
- These help us to:
 - Assess the feasibility of the planned project completion date
 - Identify when resources will need to be deployed to activities
 - Calculate when costs will be incurred
 - Globally, co-ordinate and motivate of the project team
- Assumptions:
 - Each project is composed of a number of activities
 - Can start when at least one activity is ready to start
 - Is completed when all activities are completed

An activity network: activities order and flow (arrows)



Labelling convention

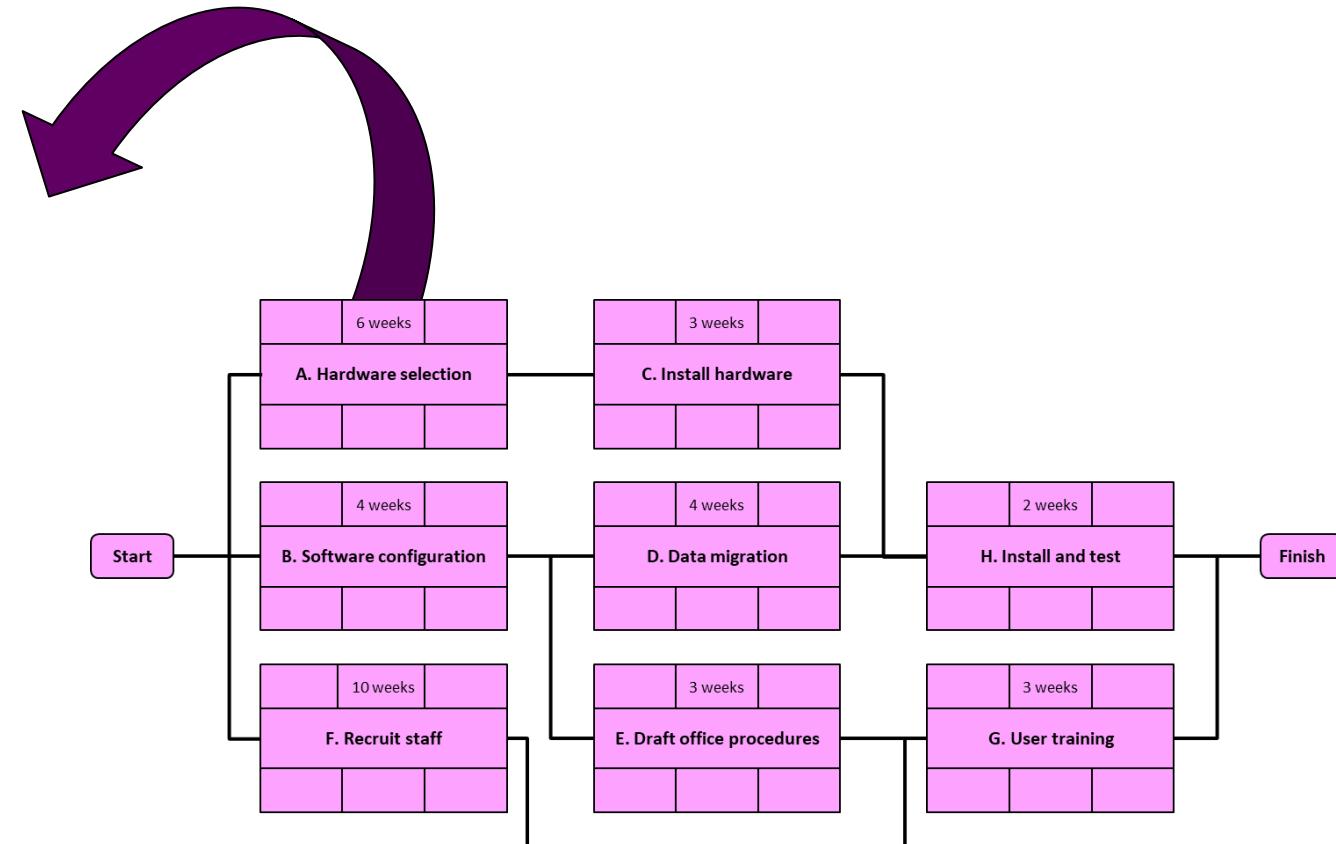
| Earliest start | Duration | Earliest finish |
|--------------------------------------|----------|-----------------|
| Activity label, activity description | | |
| Latest start | Float | Latest finish |



Labelling convention

| Earliest start | Duration | Earliest finish |
|---|----------|-----------------|
| Activity label, activity description | | |
| Latest start | Float | Latest finish |

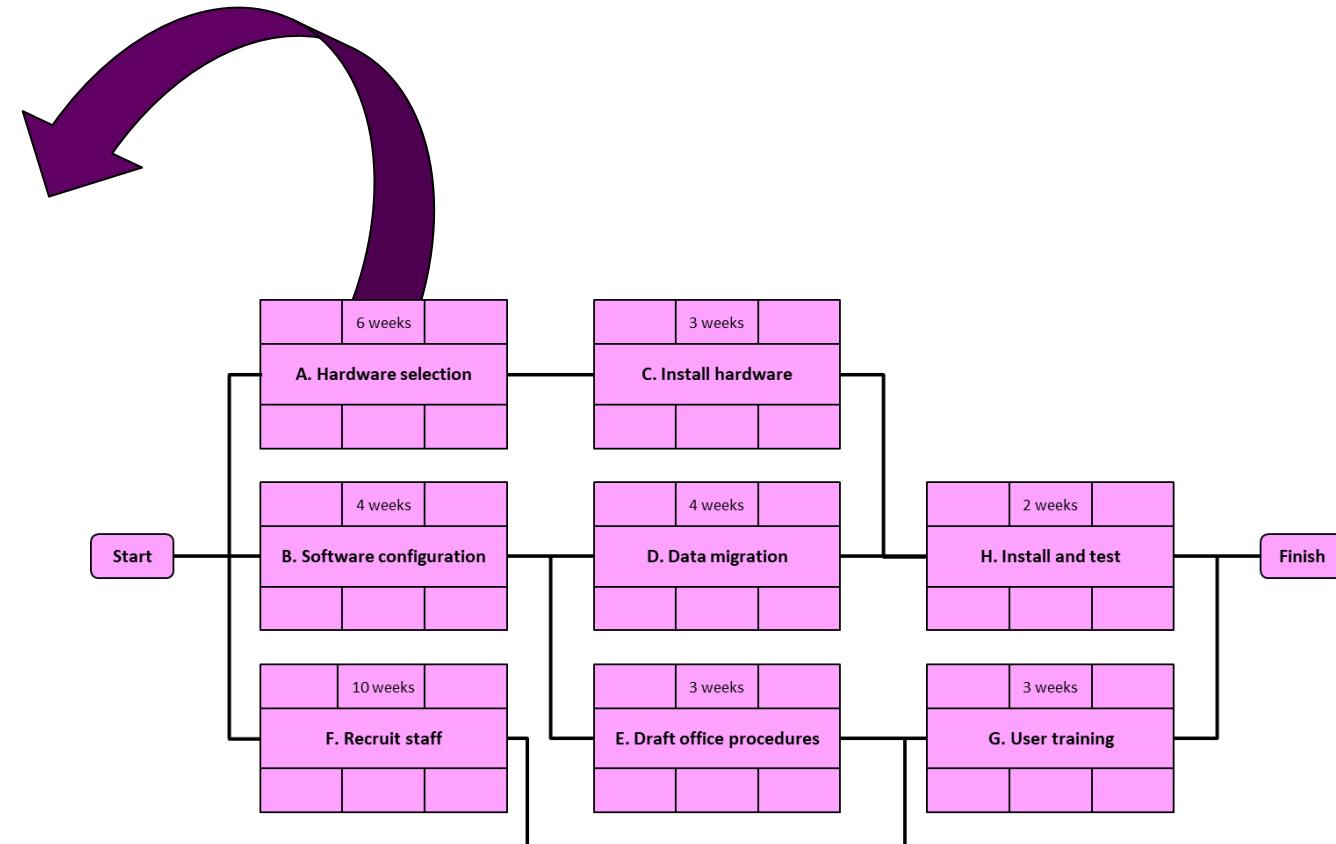
Identify **what** the activity is



Labelling convention

| Earliest start | Duration | Earliest finish |
|--------------------------------------|----------|-----------------|
| Activity label, activity description | | |
| Latest start | Float | Latest finish |

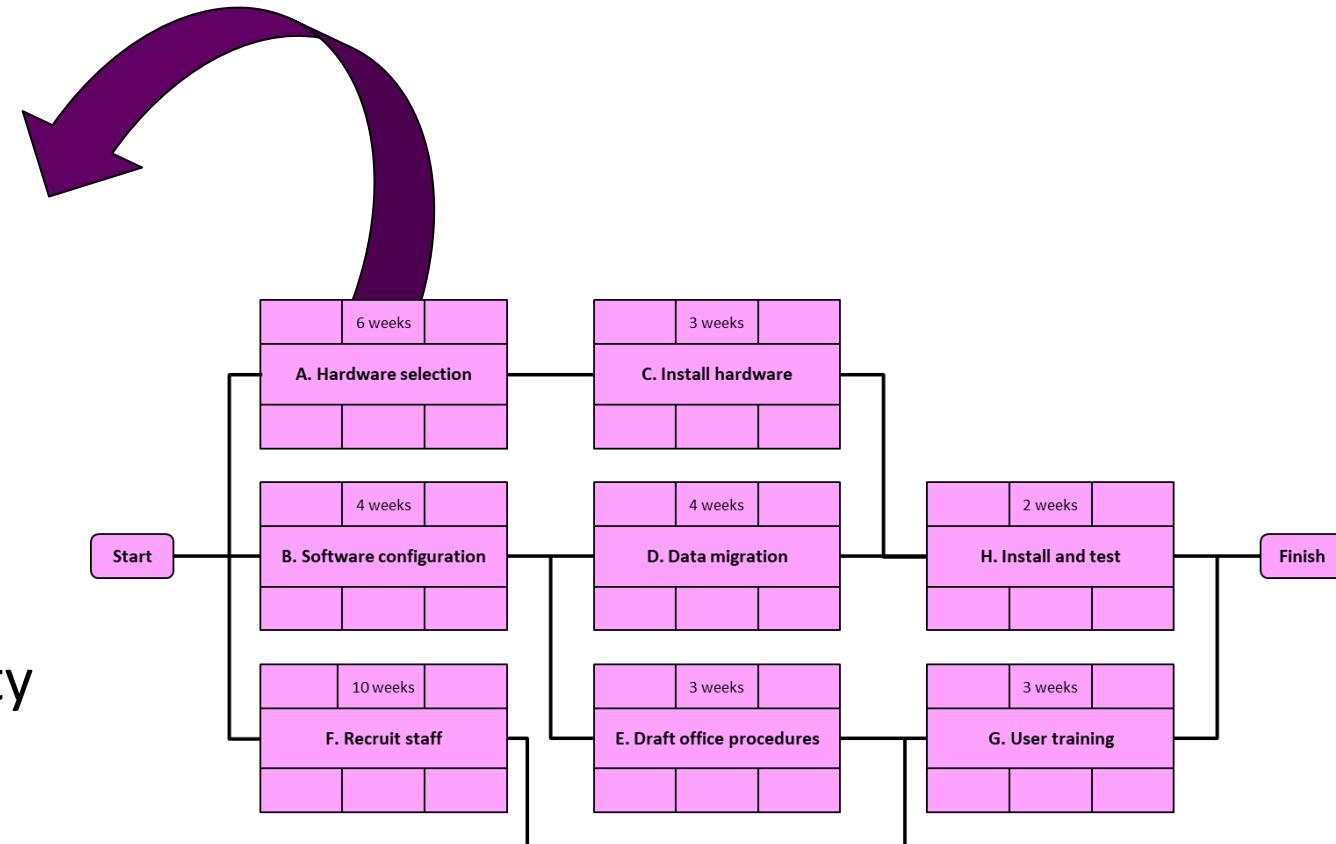
earliest time the activity can begin after the required tasks before it are finished



Labelling convention

| Earliest start | Duration | Earliest finish |
|---|-----------------|-----------------|
| Activity label, activity description | | |
| Latest start | Float | Latest finish |

The **time required to complete the activity**

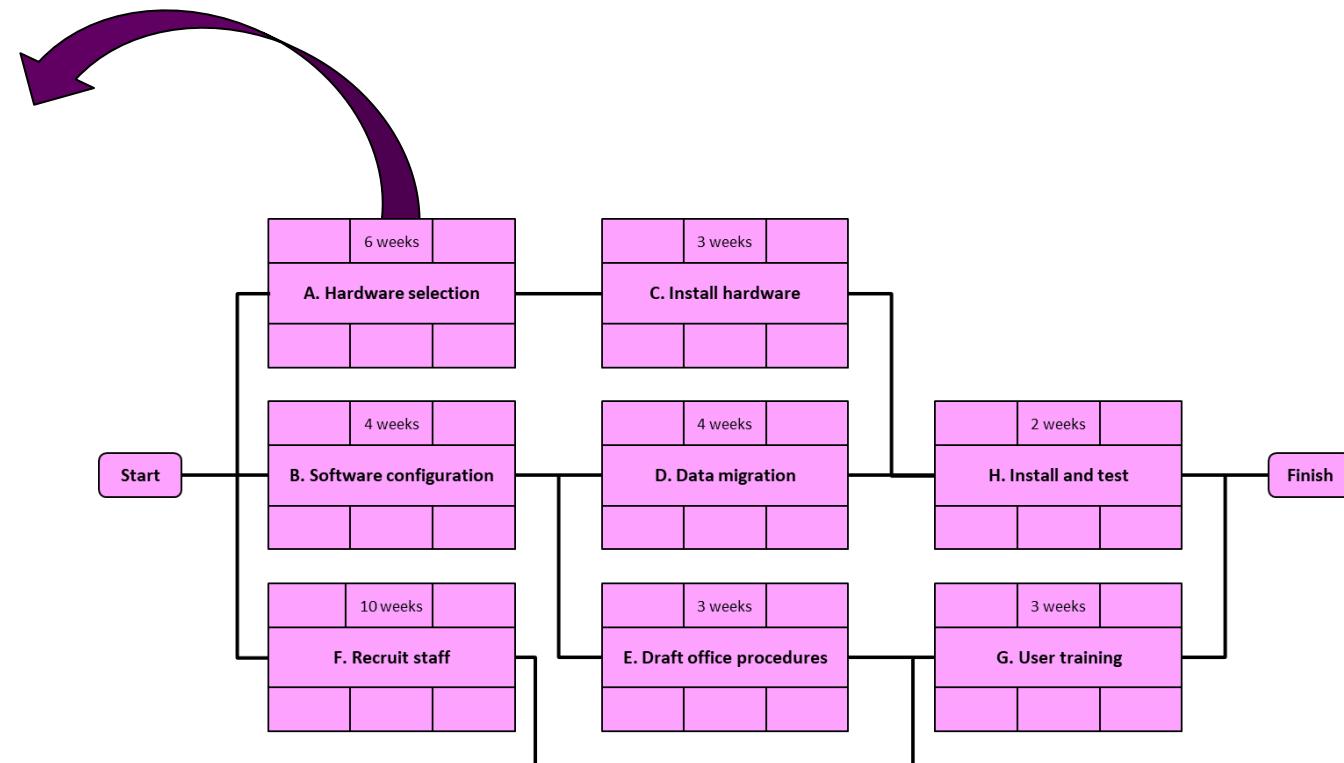


Labelling convention

| Earliest start | Duration | Earliest finish |
|---|----------|-----------------|
| Activity label, activity description | | |
| Latest start | Float | Latest finish |

The **earliest time to finish the activity**

**Earliest finish =
Earliest Start + Duration**

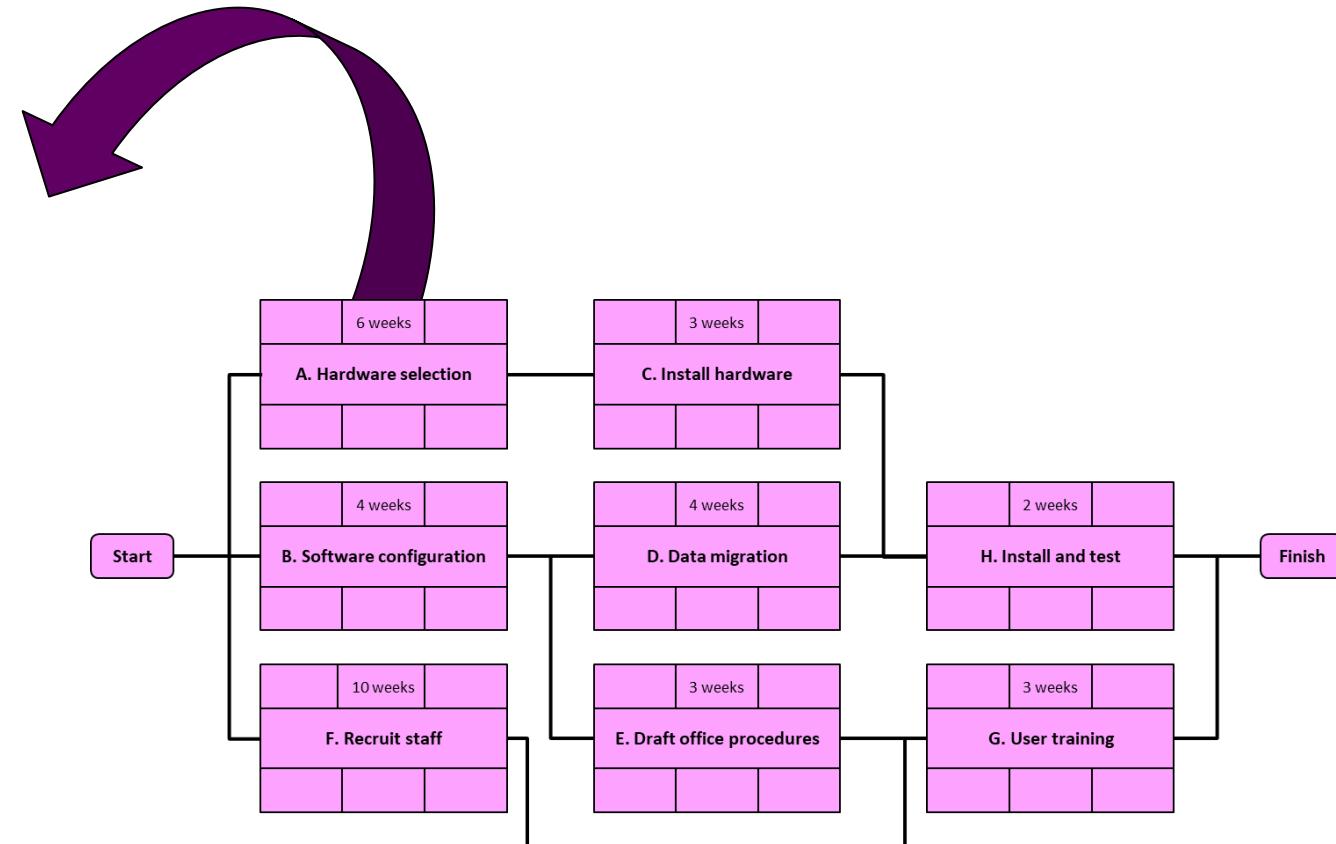


Labelling convention

| Earliest start | Duration | Earliest finish |
|---|----------|-----------------|
| Week 0 | 10 weeks | Week 10 |
| Activity label, activity description | | |
| Latest start | Float | Latest finish |

The earliest time to finish the activity

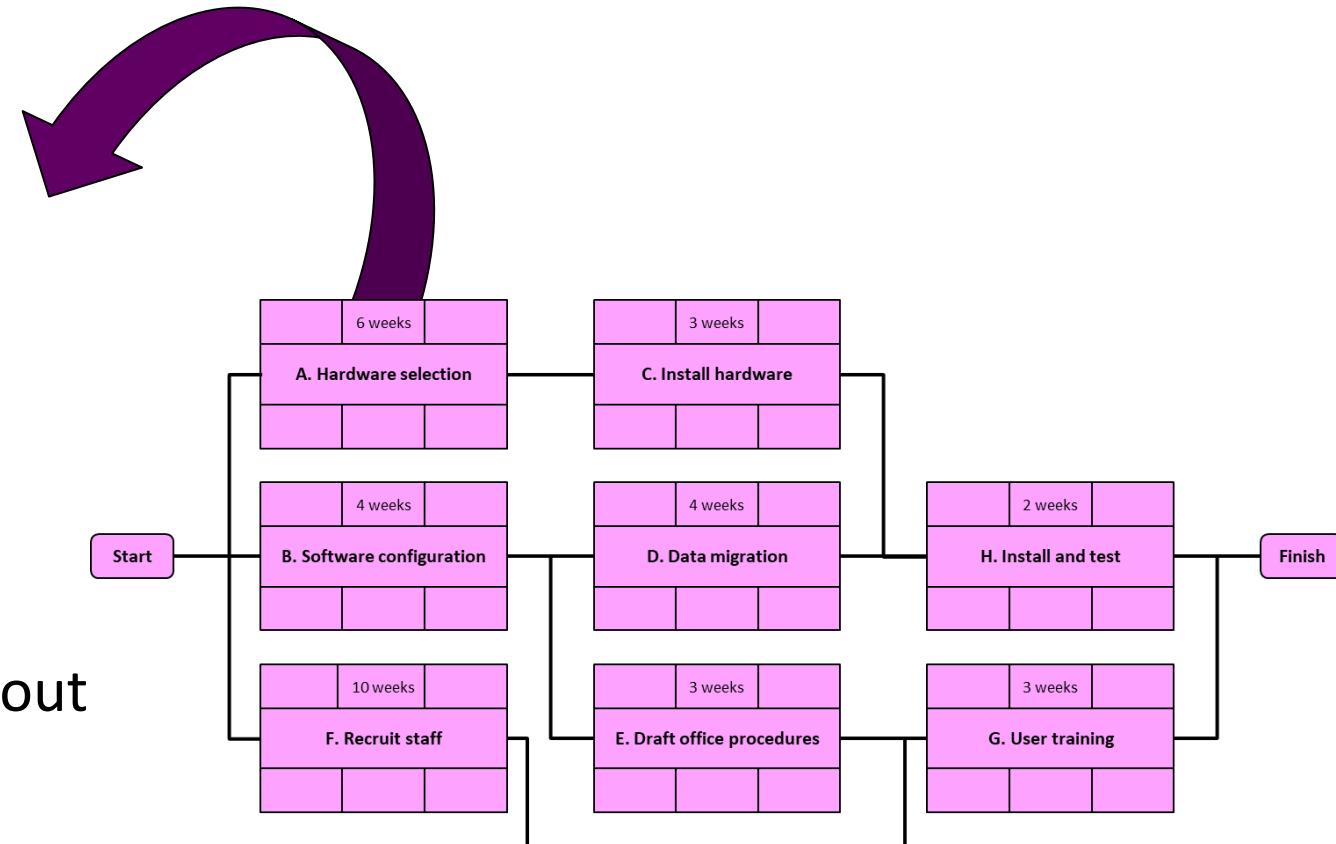
Earliest finish = Earliest Start + Duration



Labelling convention

| Earliest start | Duration | Earliest finish |
|---|----------|-----------------|
| Activity label, activity description | | |
| Latest start | Float | Latest finish |

The **latest time** an activity **can finish** without impacting the overall project timeline

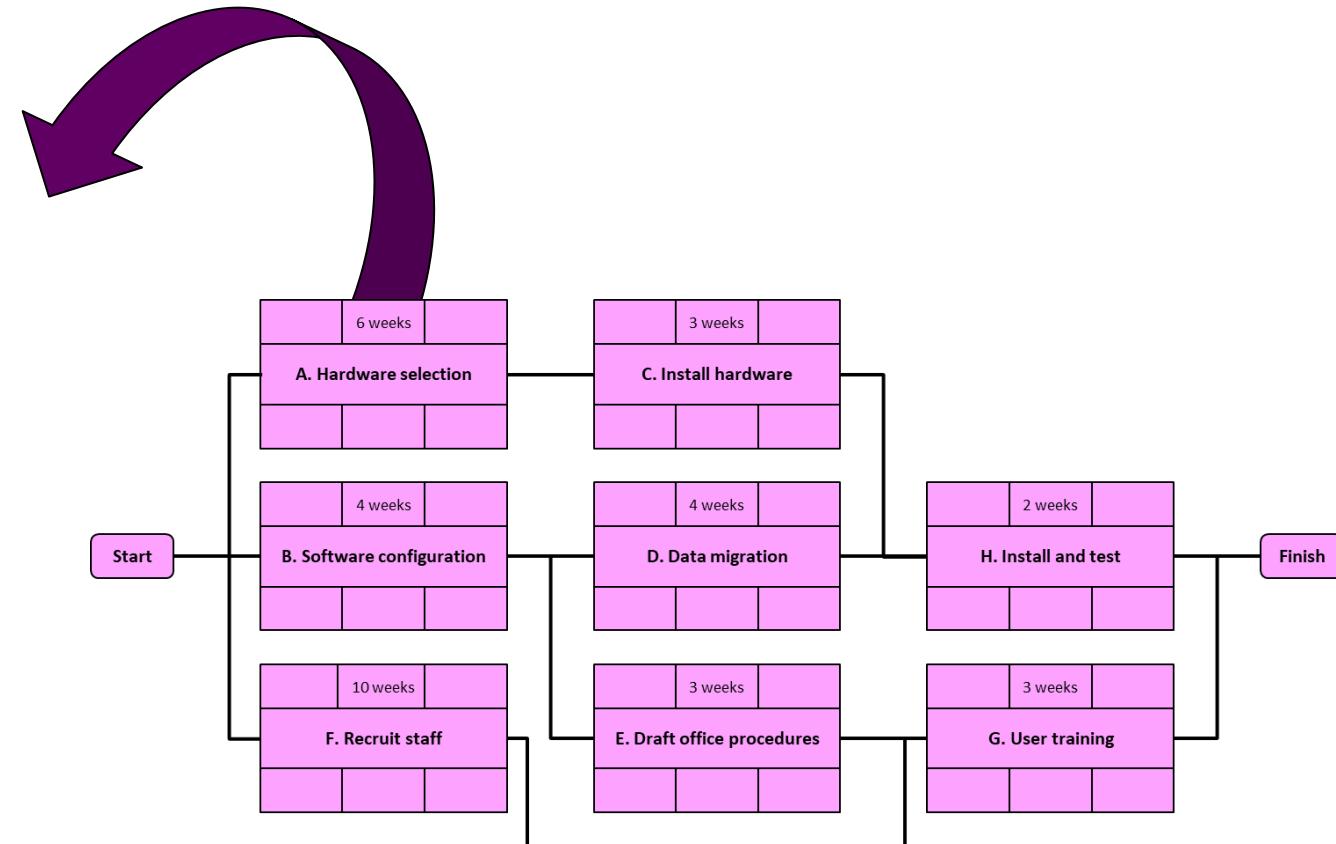


Labelling convention

| Earliest start | Duration | Earliest finish |
|---|----------|-----------------|
| Latest start | Float | Latest finish |
| Activity label, activity description | | |

The **latest time** an activity **can begin** without delaying the project

Latest start =
Latest finish - Duration

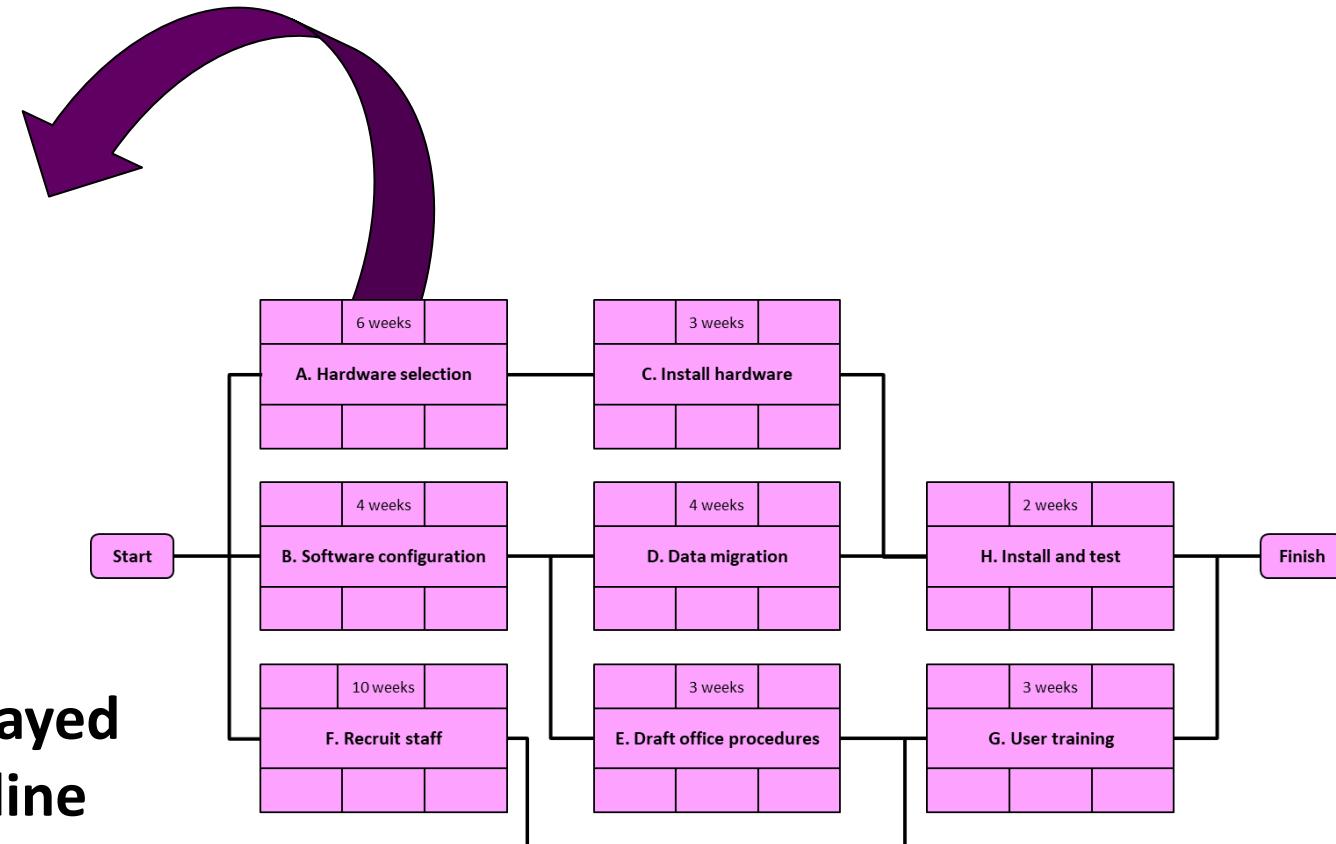


Labelling convention

| Earliest start | Duration | Earliest finish |
|---|----------|-----------------|
| Activity label, activity description | | |
| Latest start | Float | Latest finish |

The **amount of time** an activity can be **delayed** without affecting the overall project **timeline**

Float is a calculated value
Float = latest finish - earliest finish

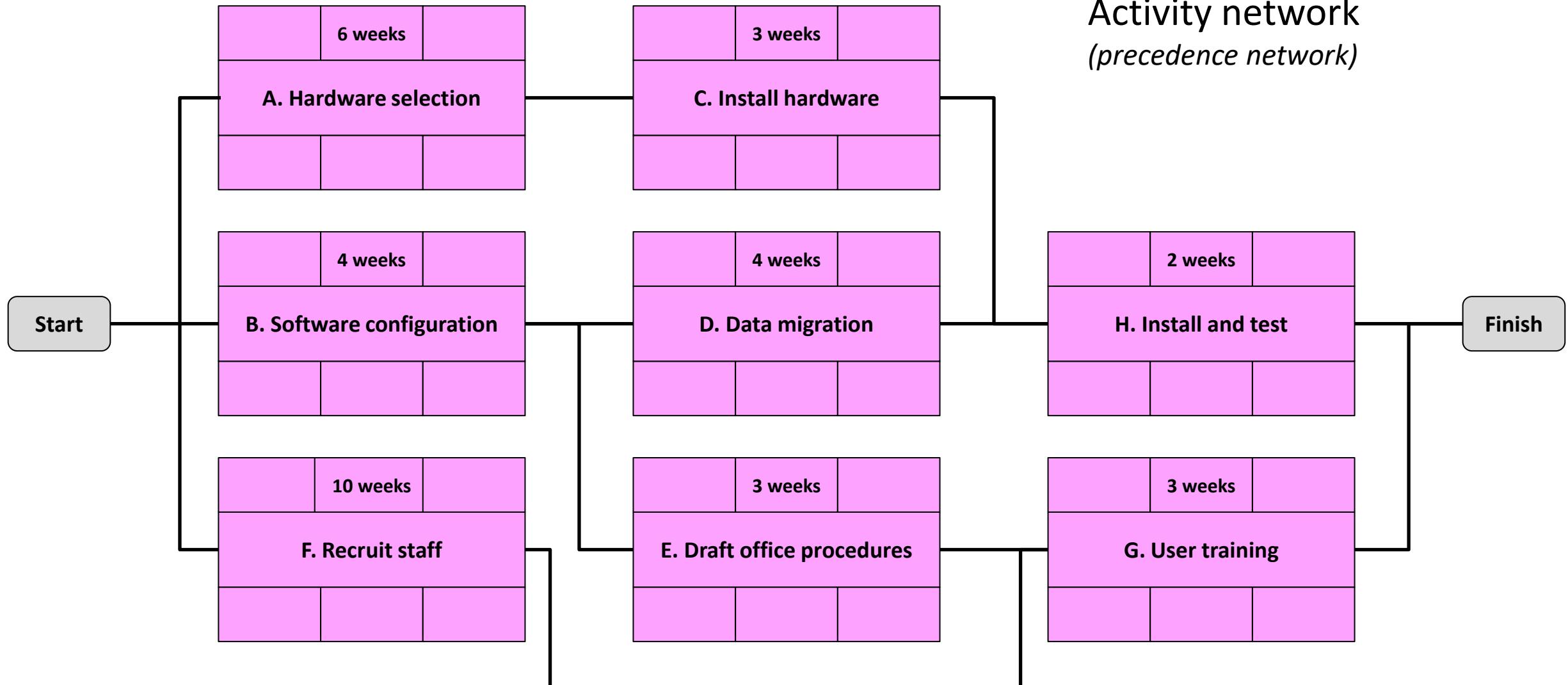


Example activity network

Project specification with estimated activity durations and dependencies

| Activity | Description | Duration (Weeks) | Precedence requirements |
|----------|-------------------------|------------------|-------------------------|
| A | Hardware selection | 6 | - |
| B | System configuration | 4 | - |
| C | Install hardware | 3 | A |
| D | Data migration | 4 | B |
| E | Draft office procedures | 3 | B |
| F | Recruit staff | 10 | - |
| G | User training | 3 | E, F |
| H | Install and test system | 2 | C, D |

Example activity network



Example activity network

Forward pass (left to right)

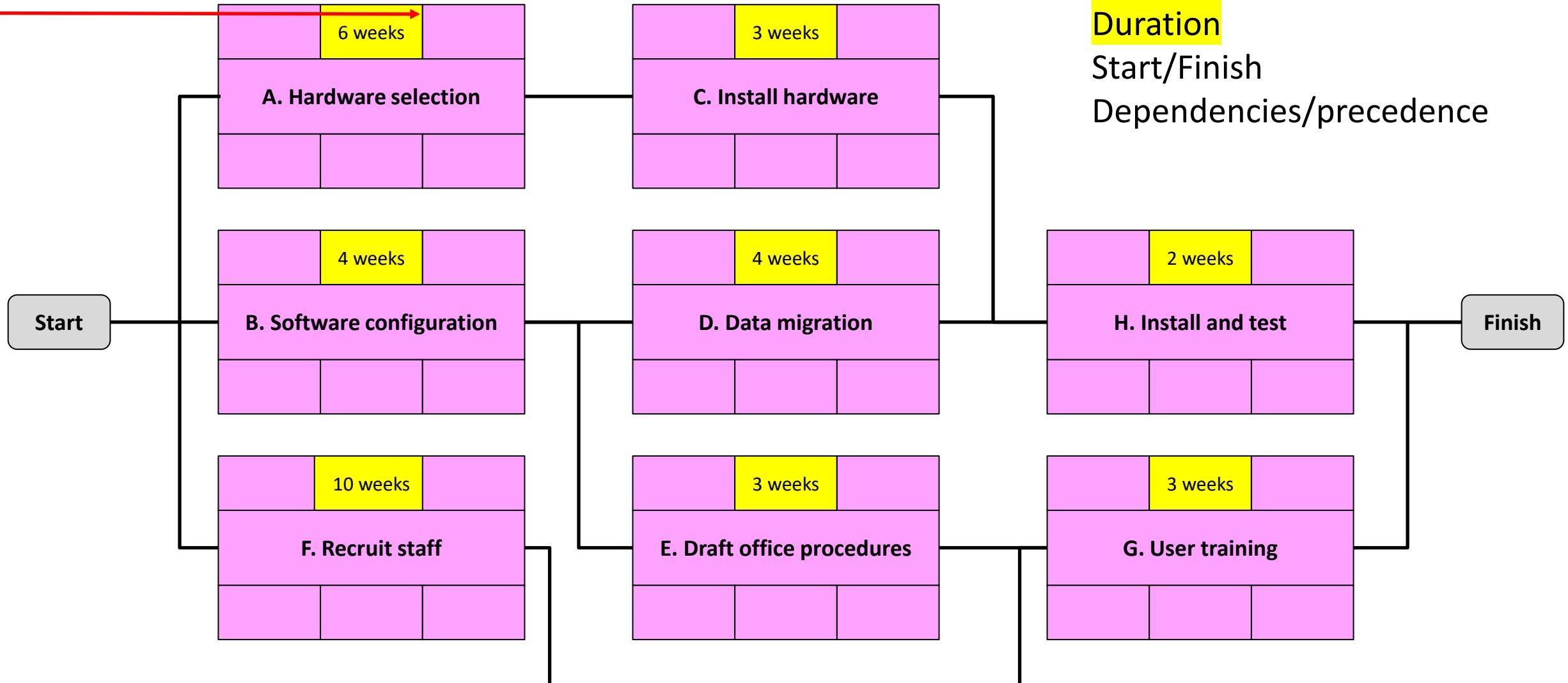
earliest start earliest finish

Activity Code. Description.

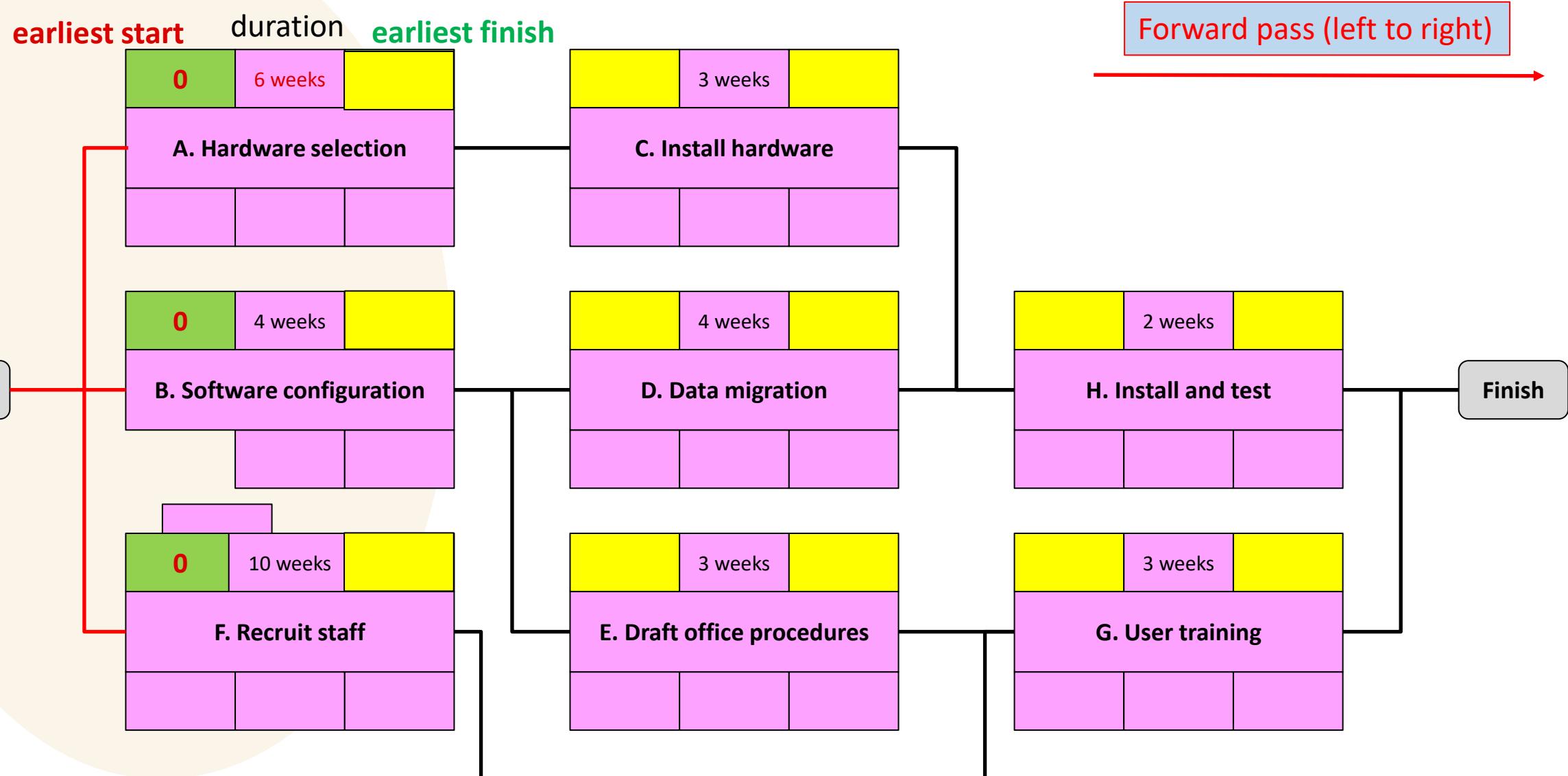
Duration

Start/Finish

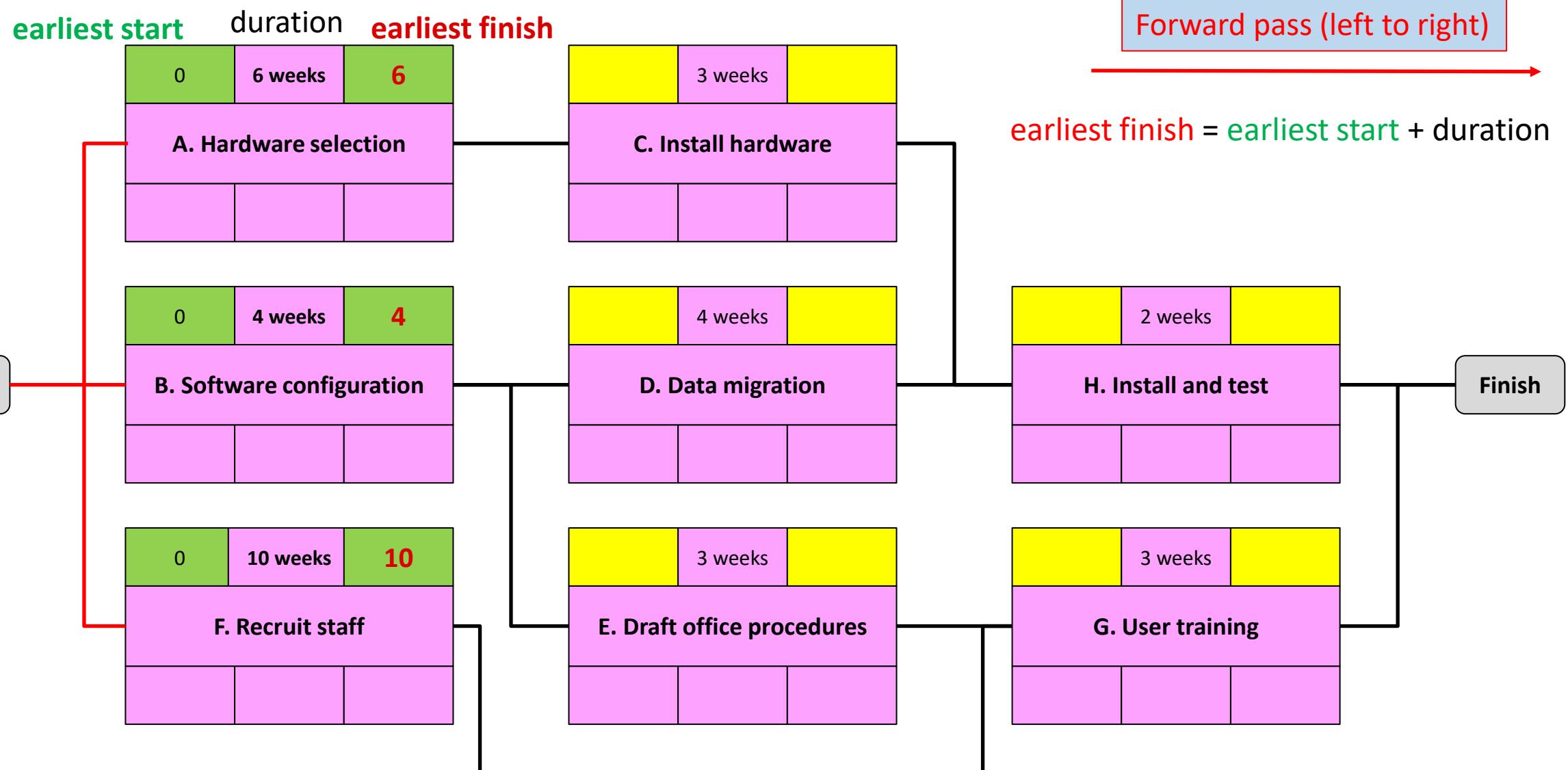
Dependencies/precedence



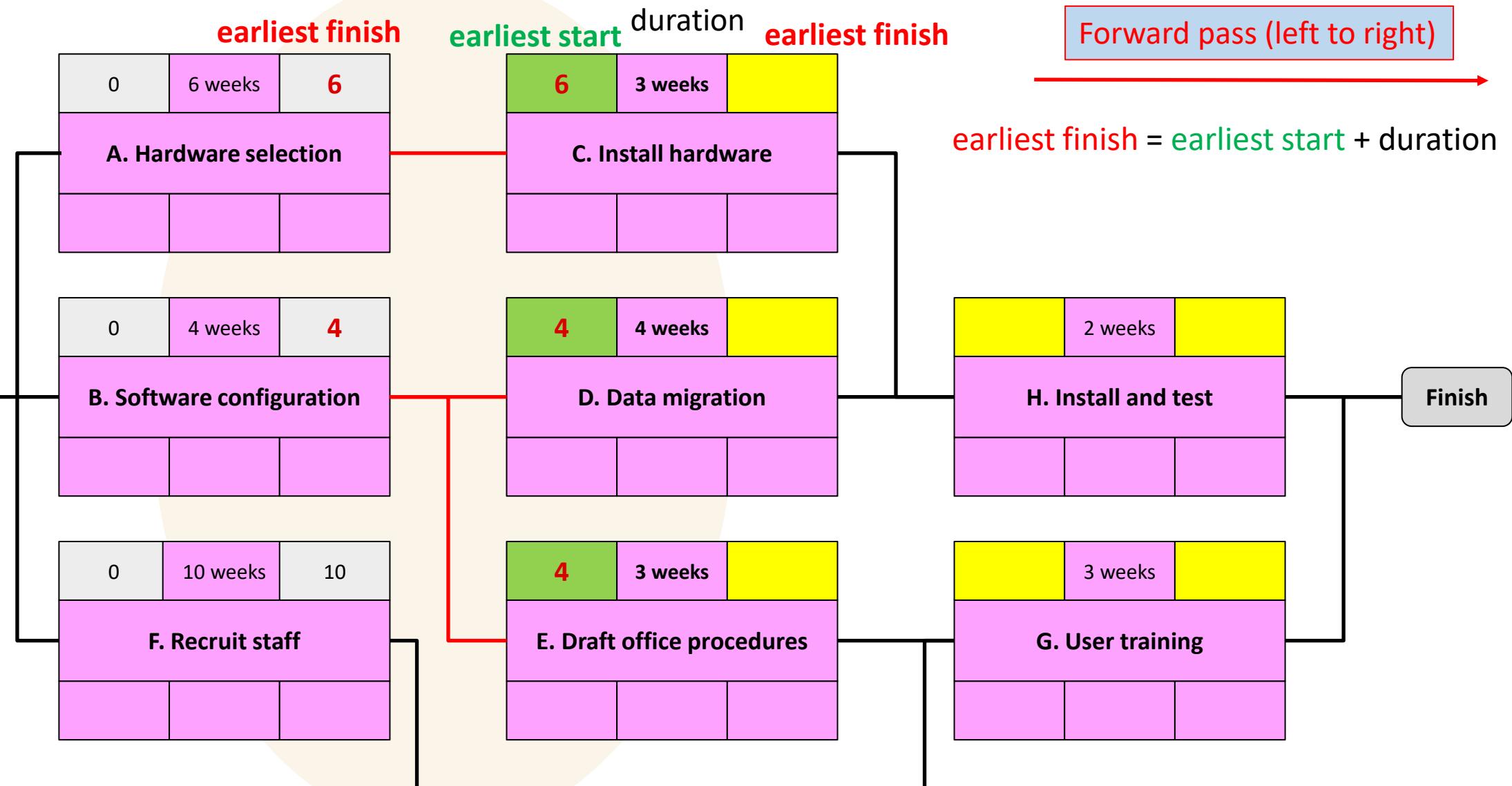
Earliest start/finish calculated



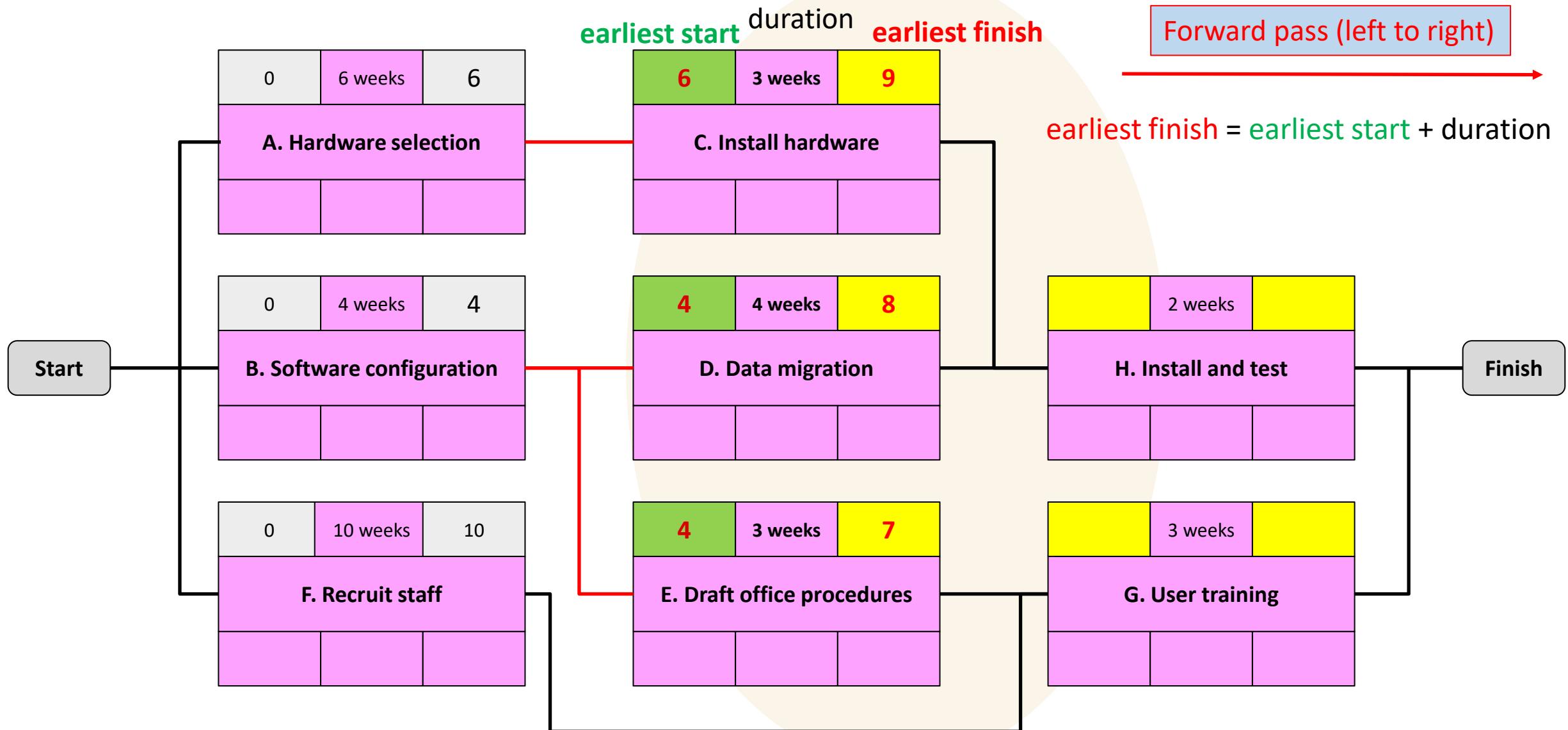
Earliest start/finish calculated



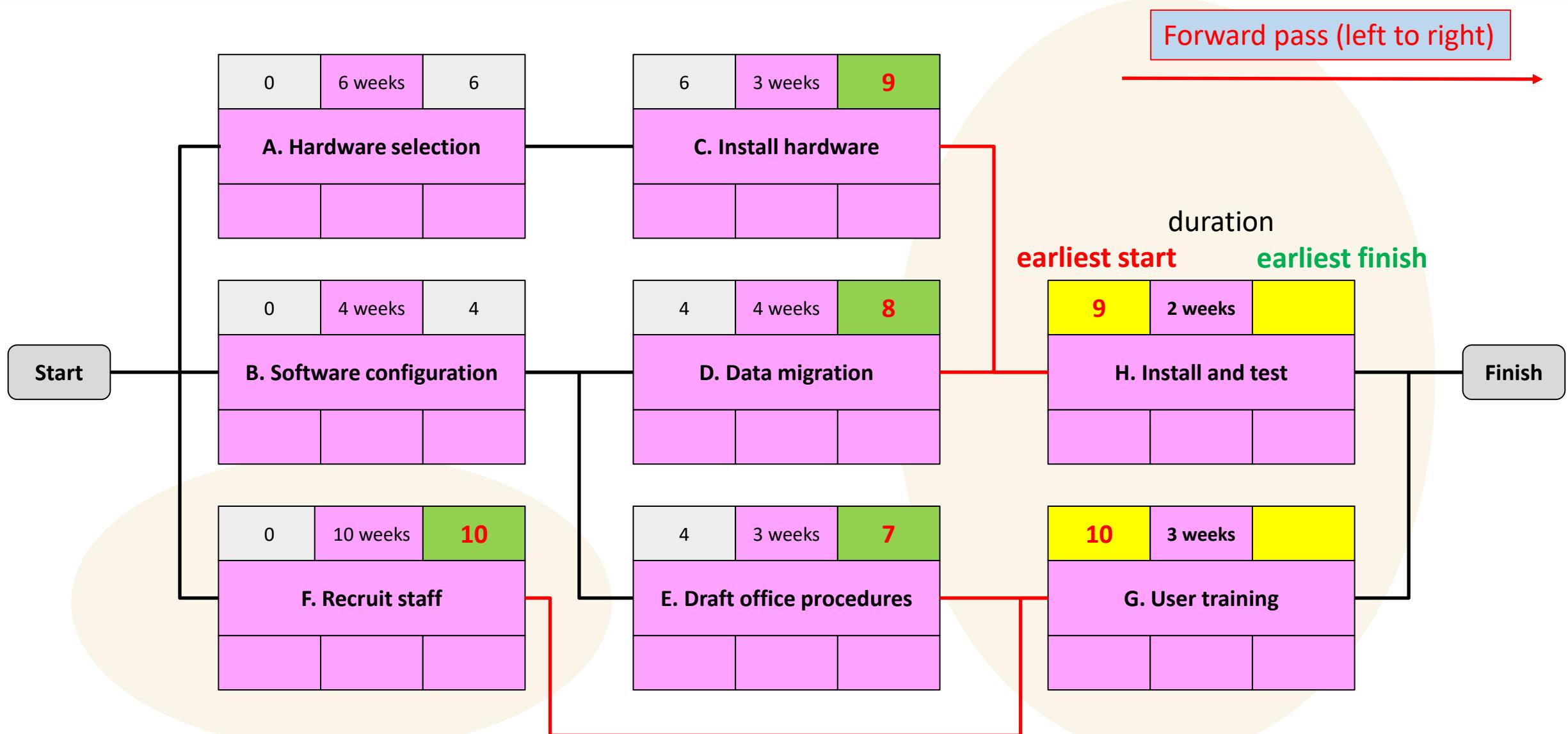
Earliest start/finish calculated



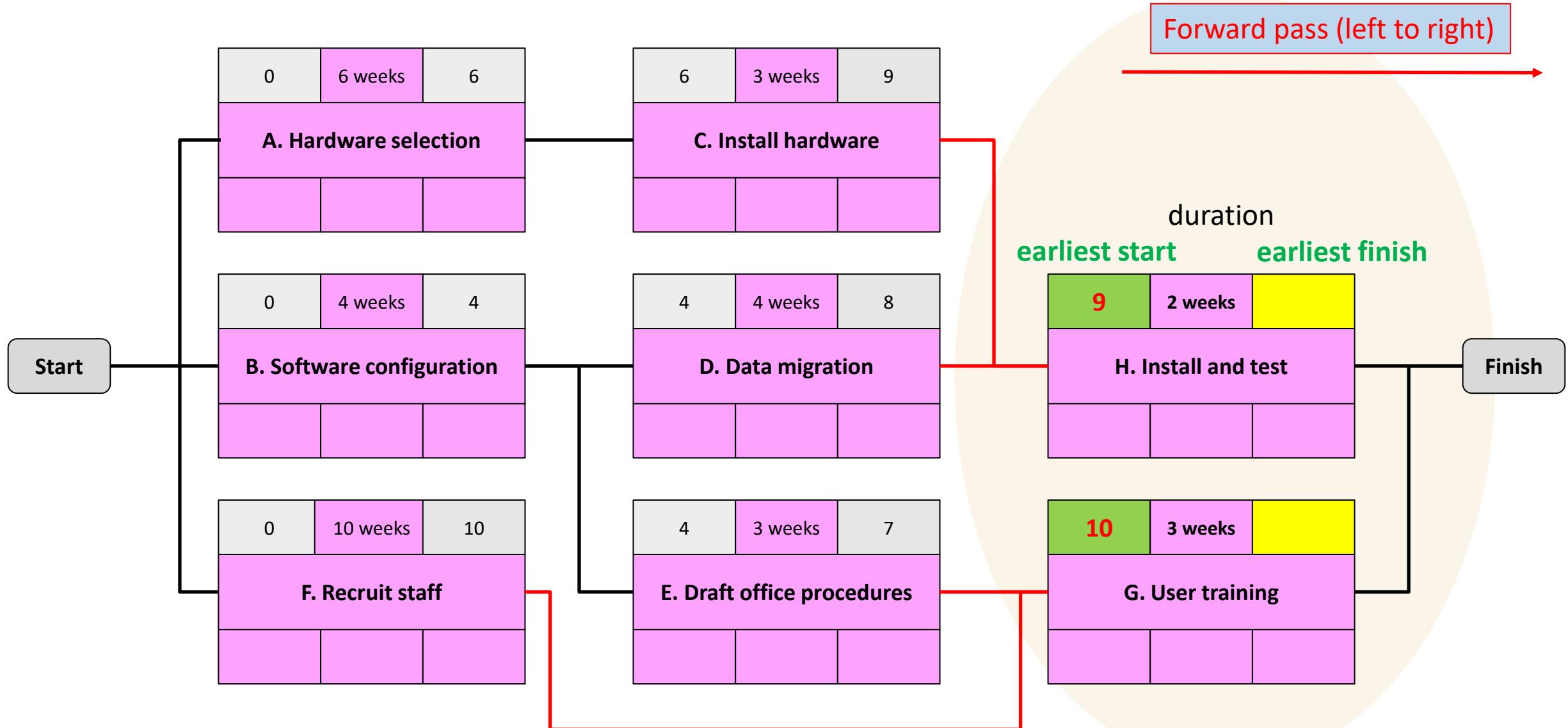
Earliest start/finish calculated



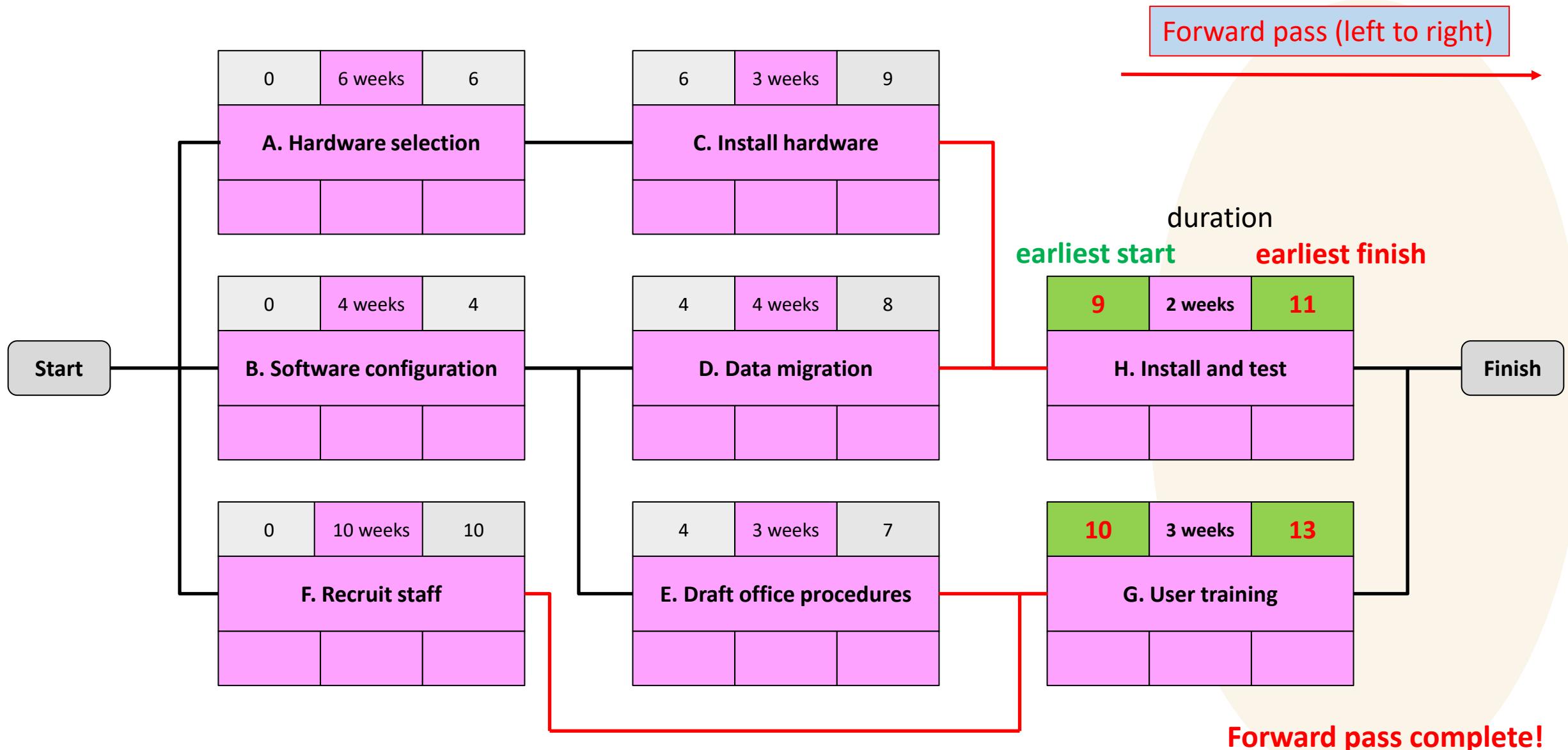
Earliest start/finish calculated



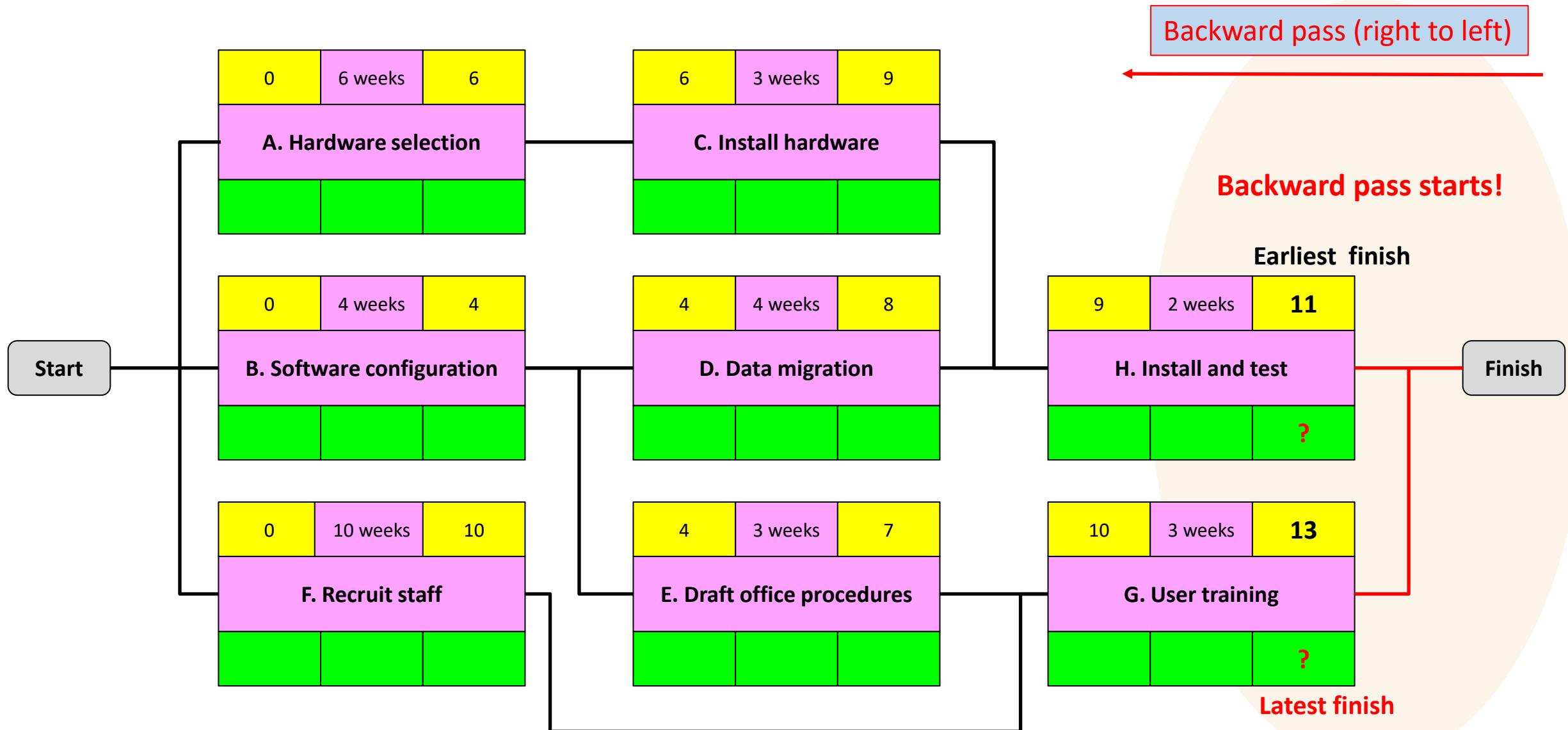
Earliest start/finish calculated



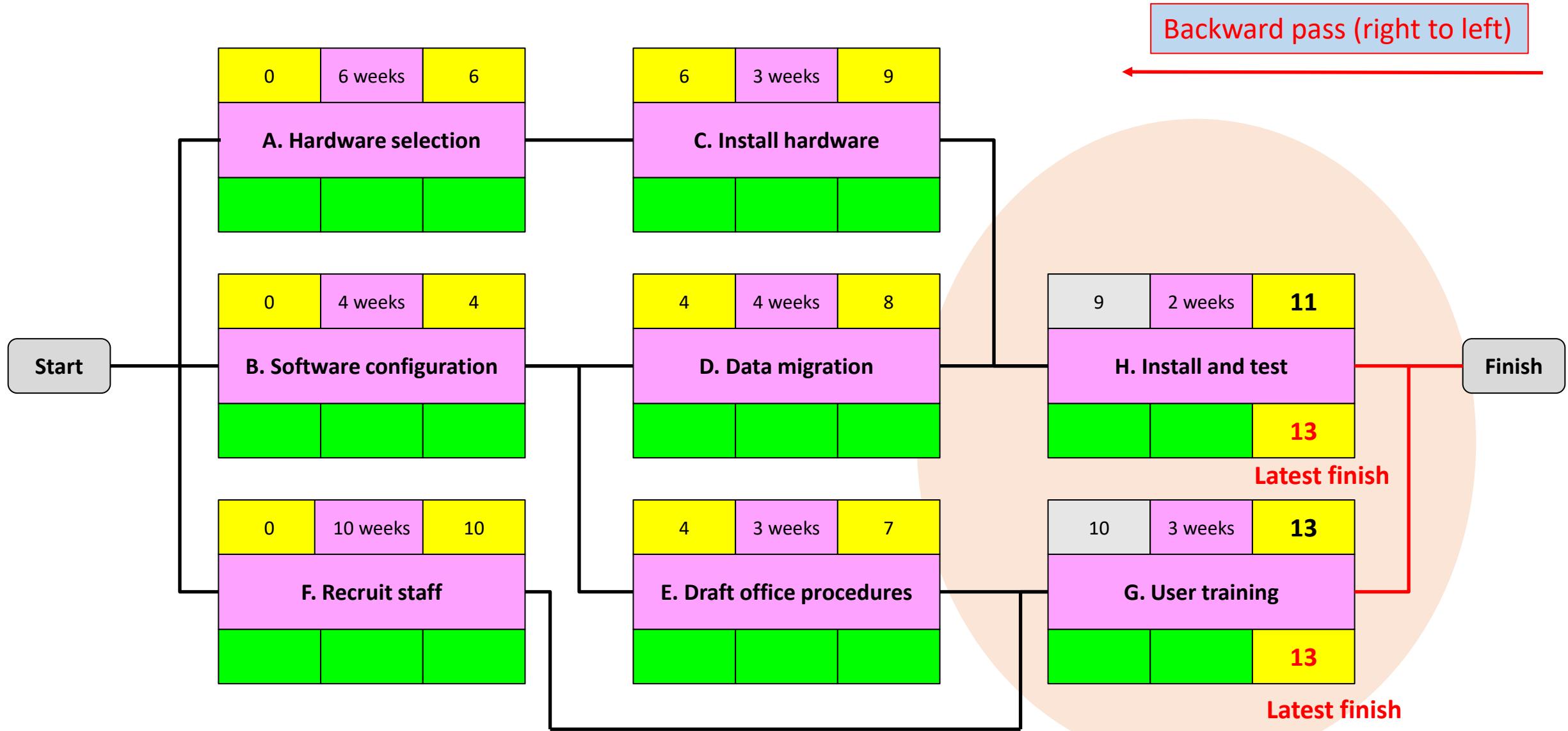
Earliest start/finish calculated



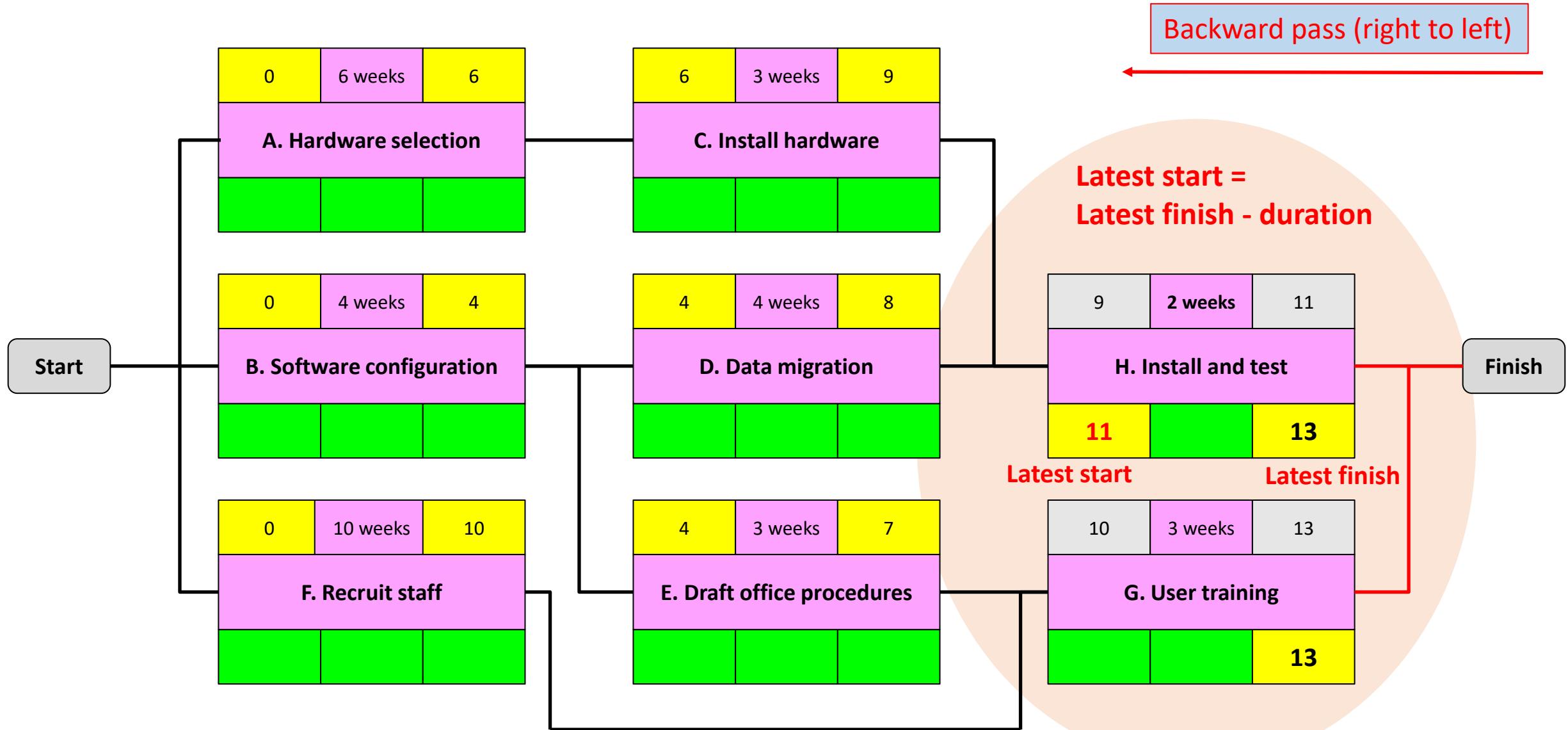
Latest start/finish and float calculated



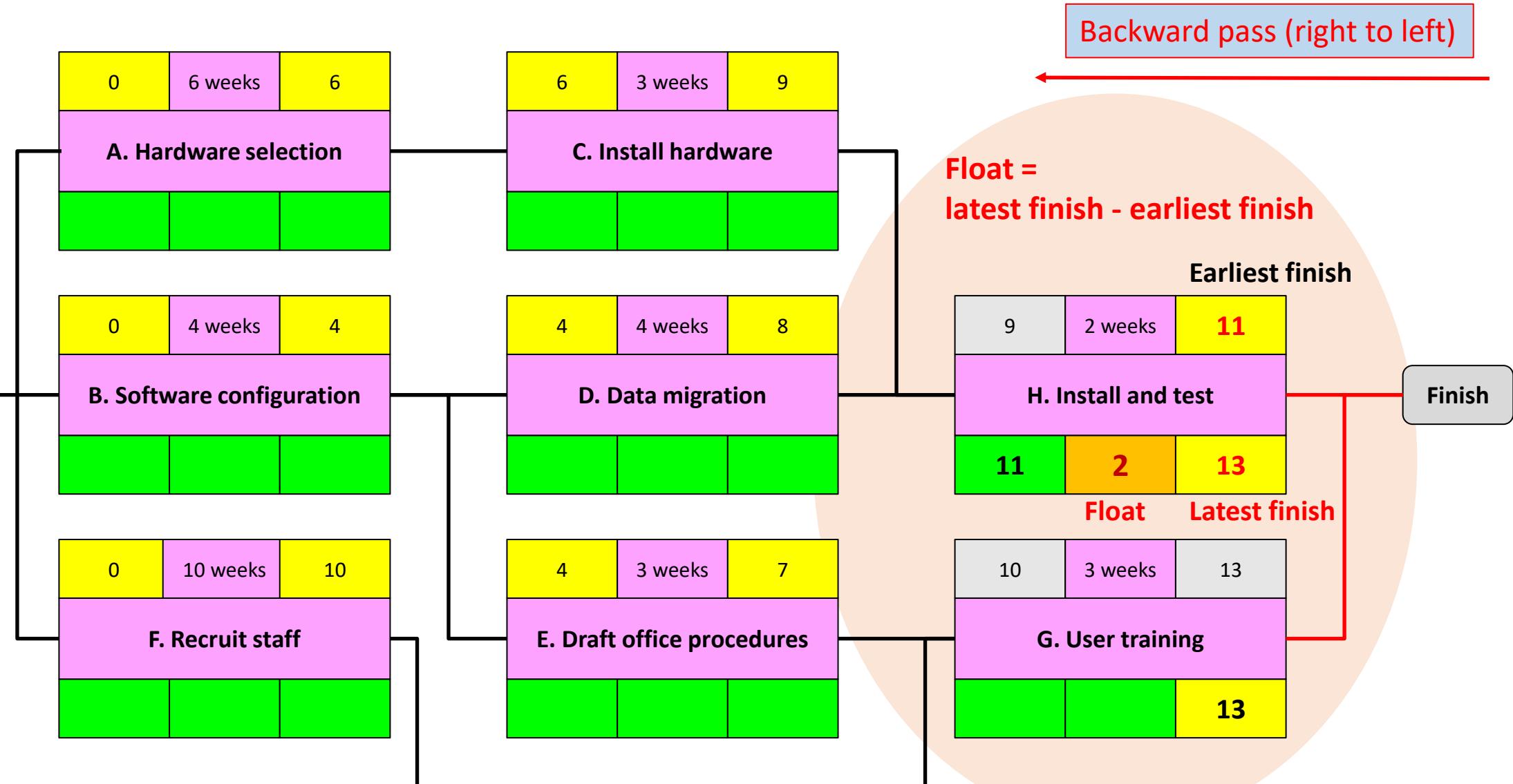
Latest start/finish and float calculated



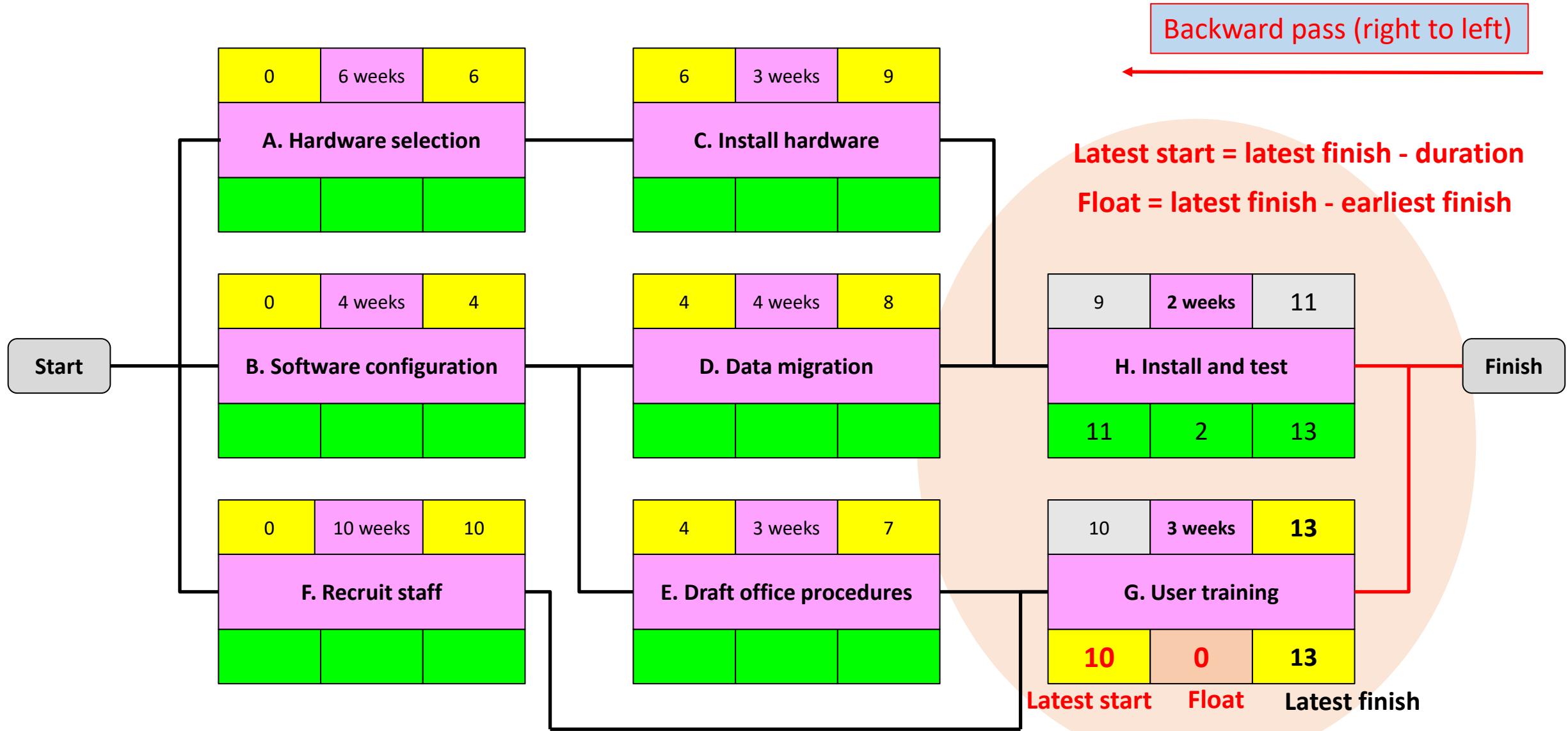
Latest start/finish and float calculated



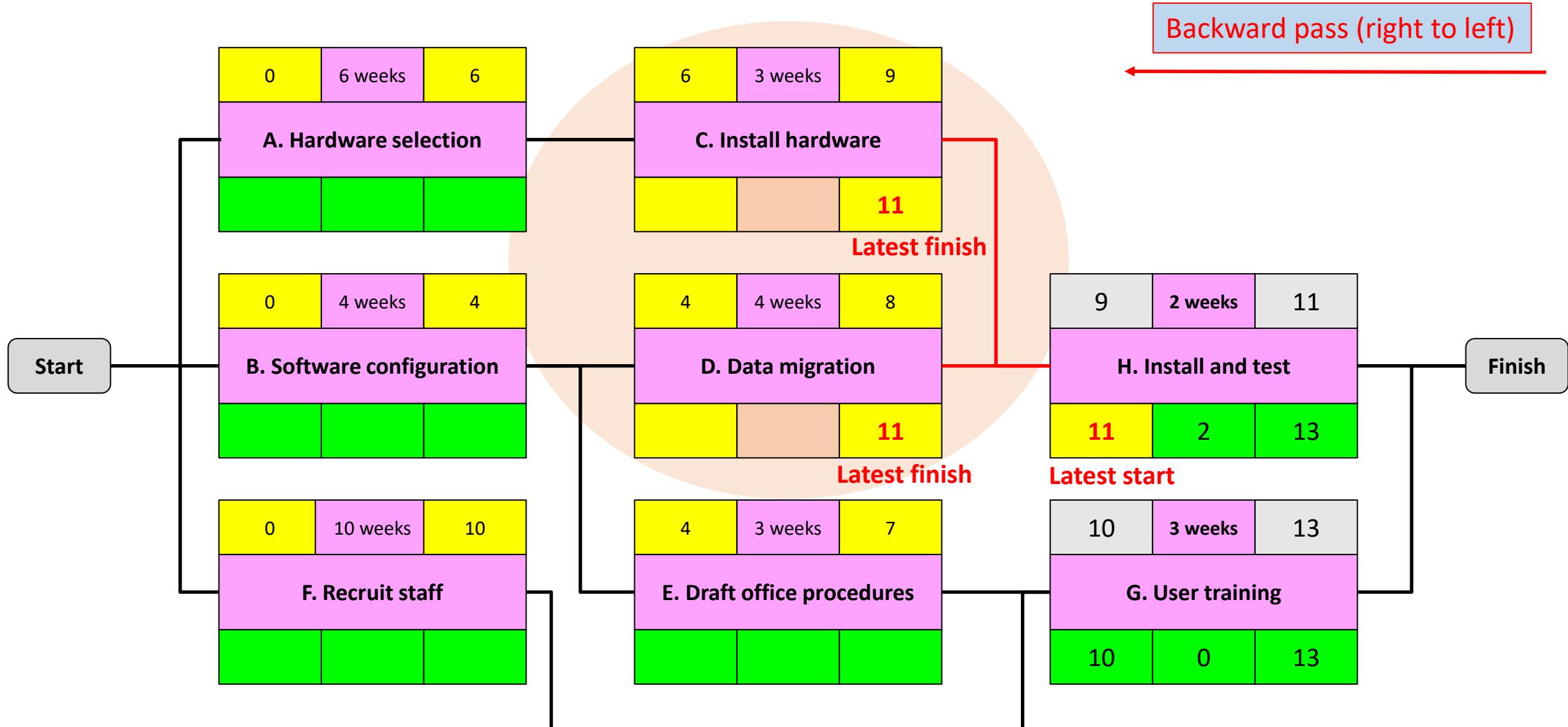
Latest start/finish and float calculated



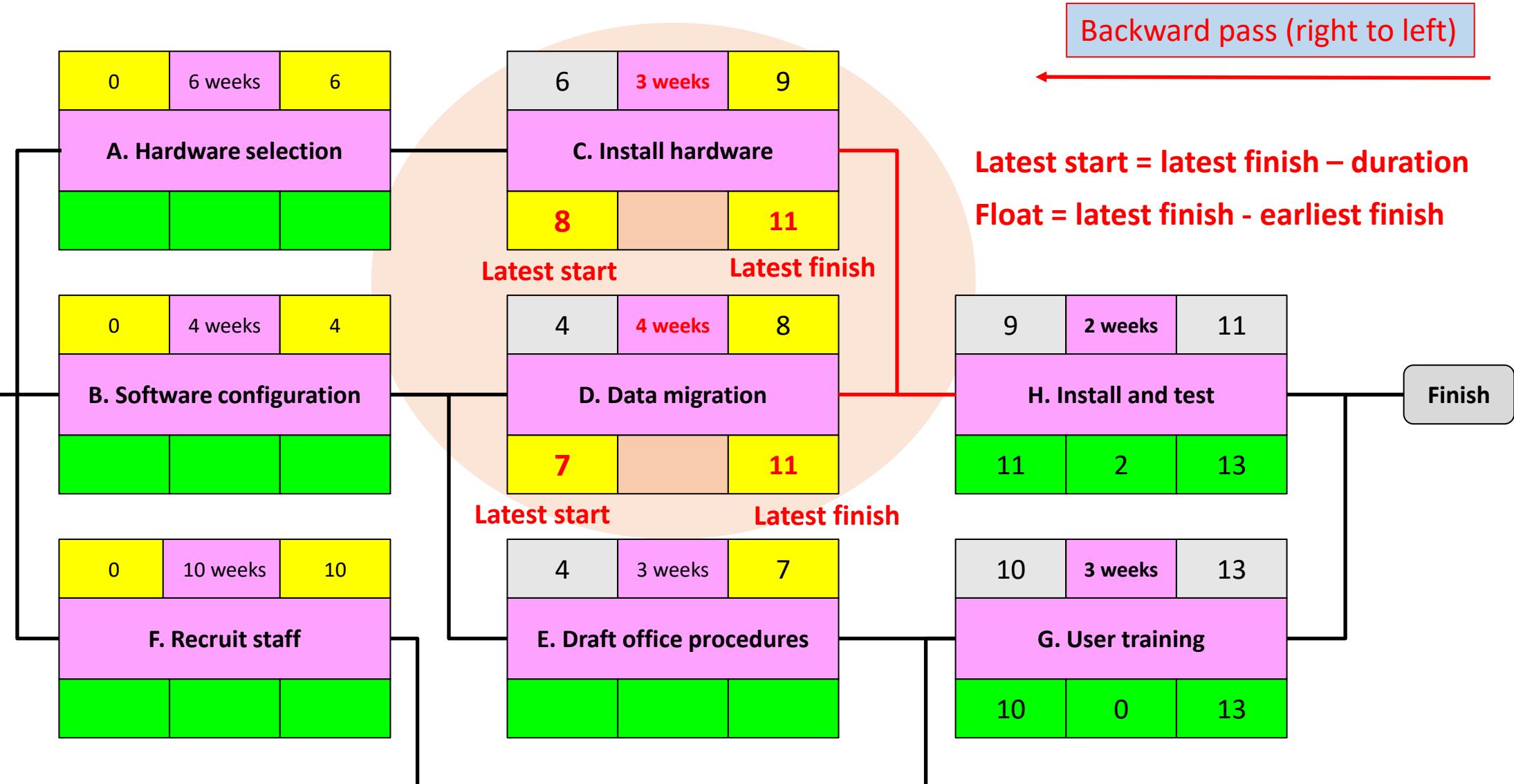
Latest start/finish and float calculated



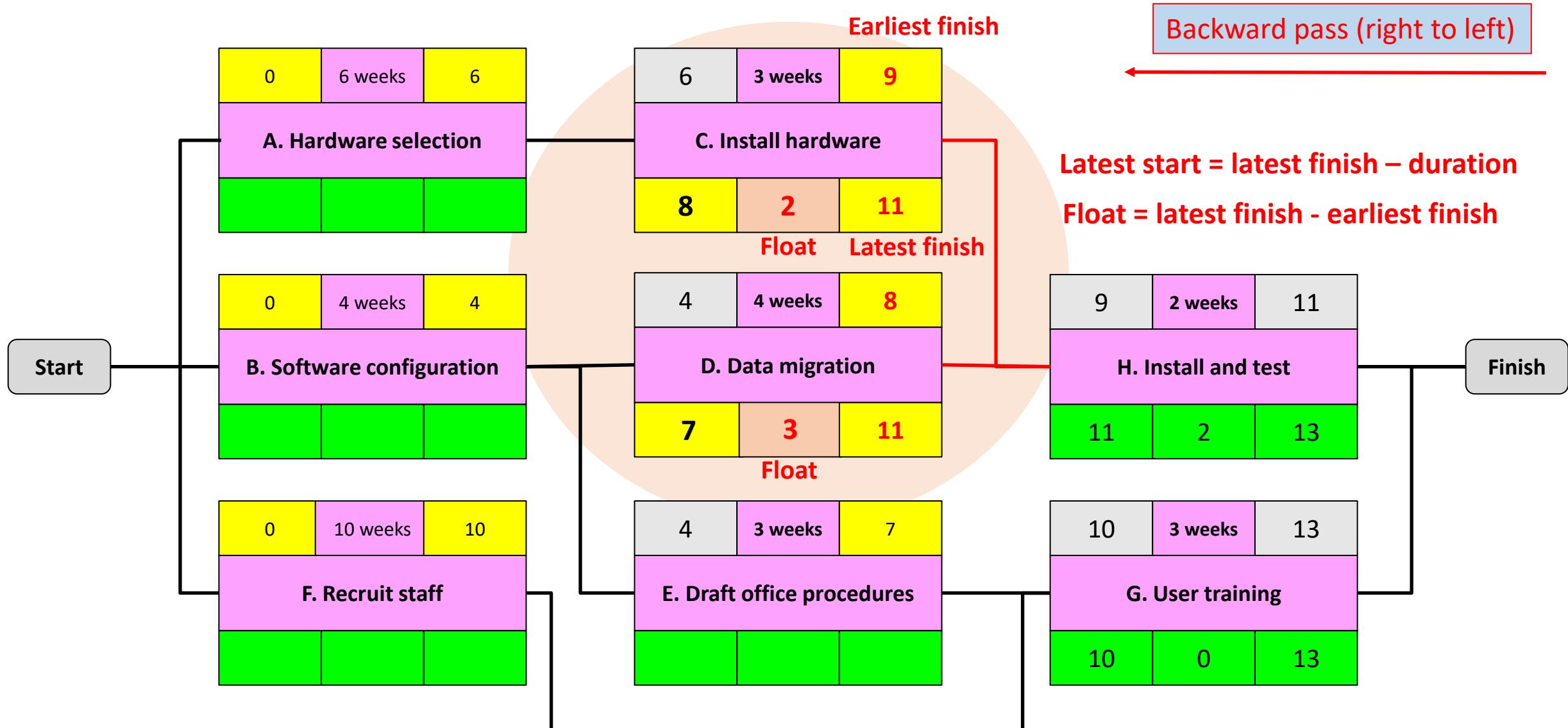
Latest start/finish and float calculated



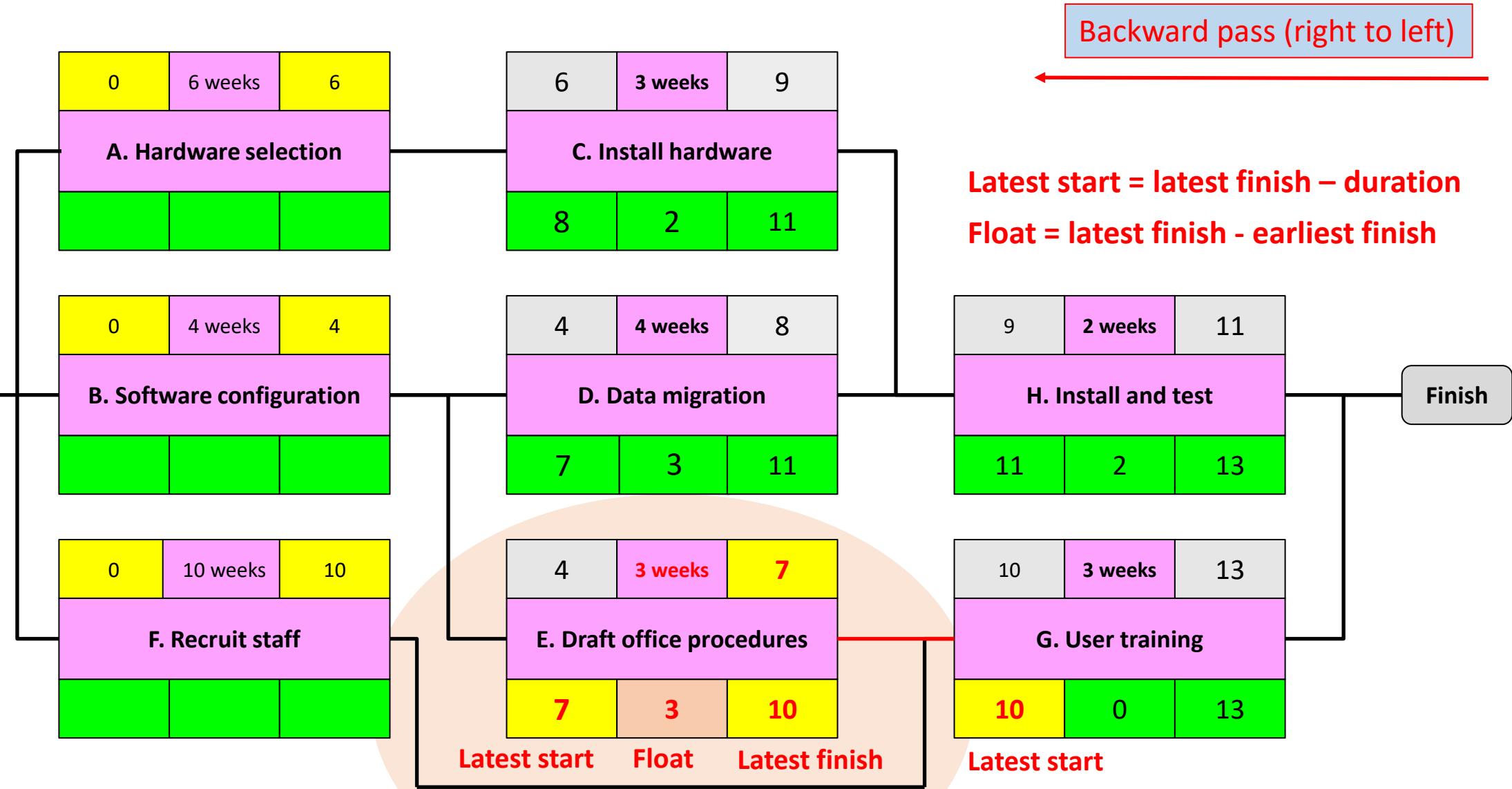
Latest start/finish and float calculated



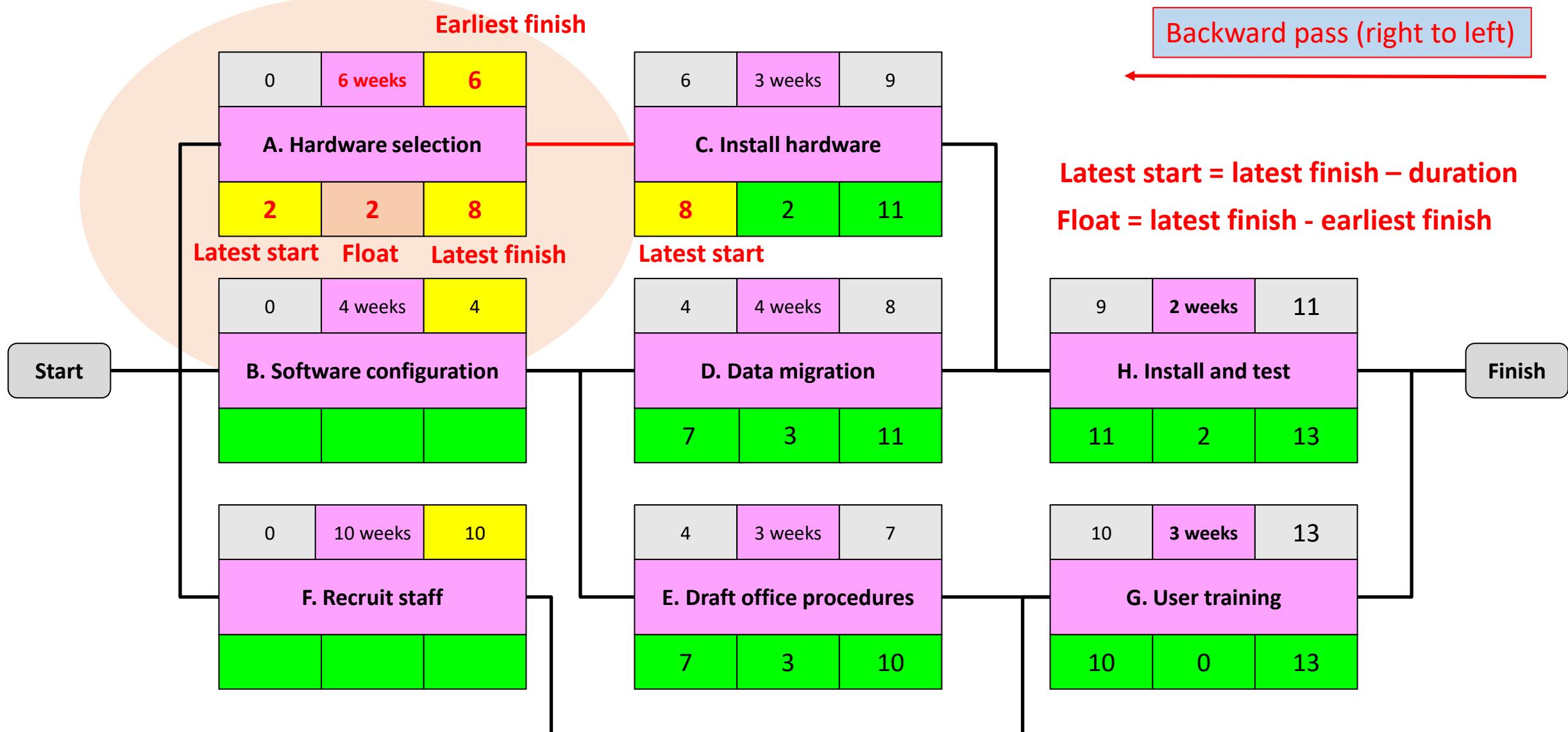
Latest start/finish and float calculated



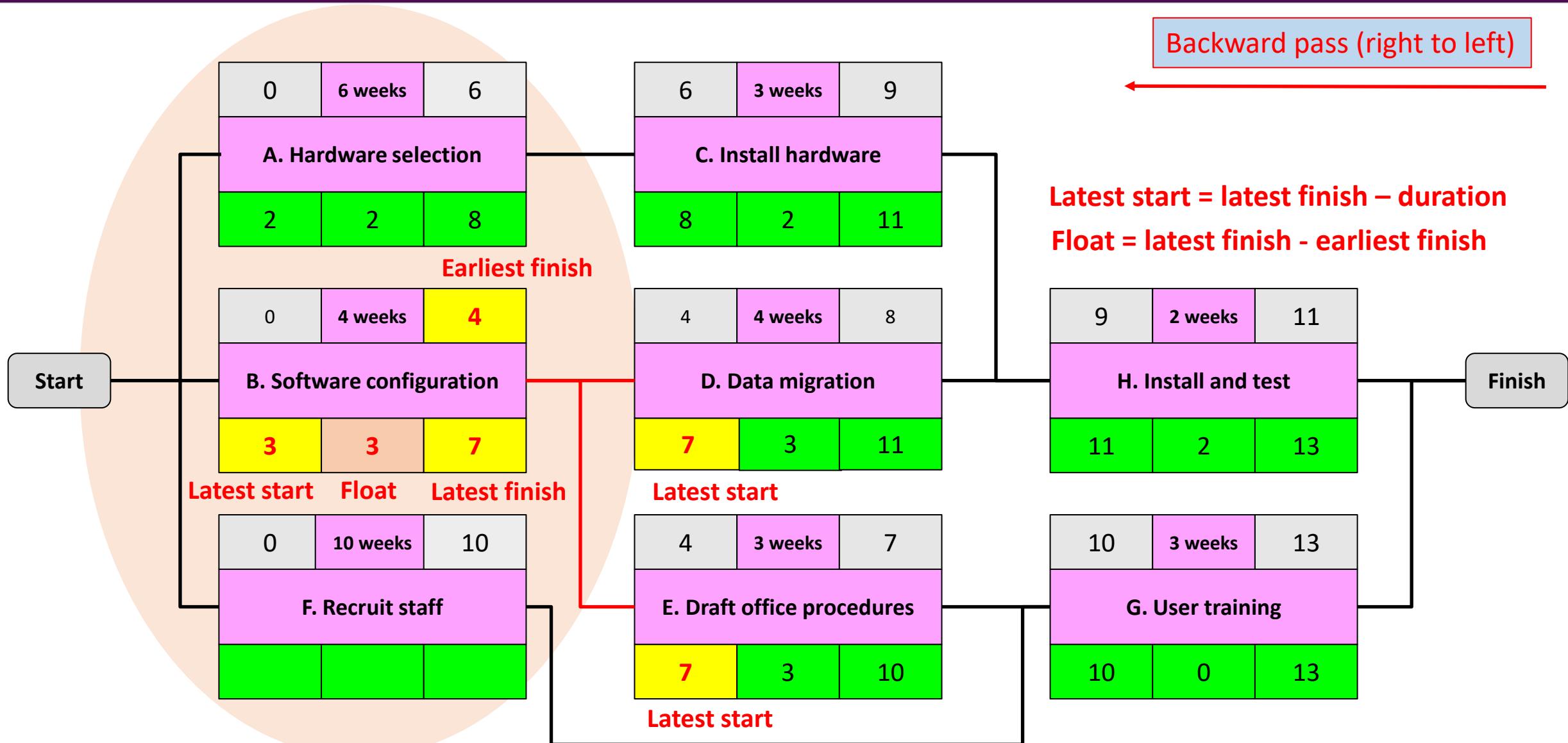
Latest start/finish and float calculated



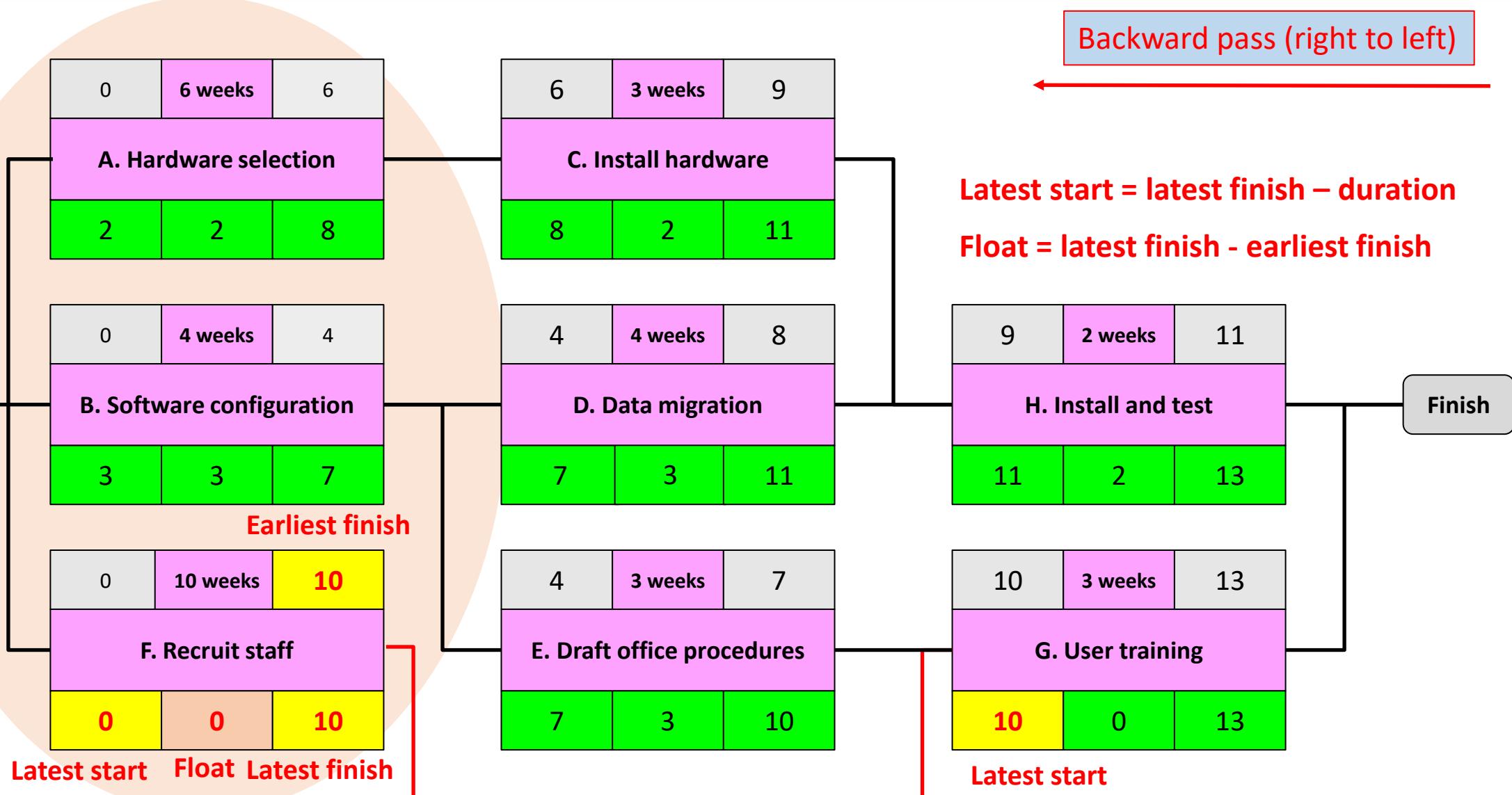
Latest start/finish and float calculated



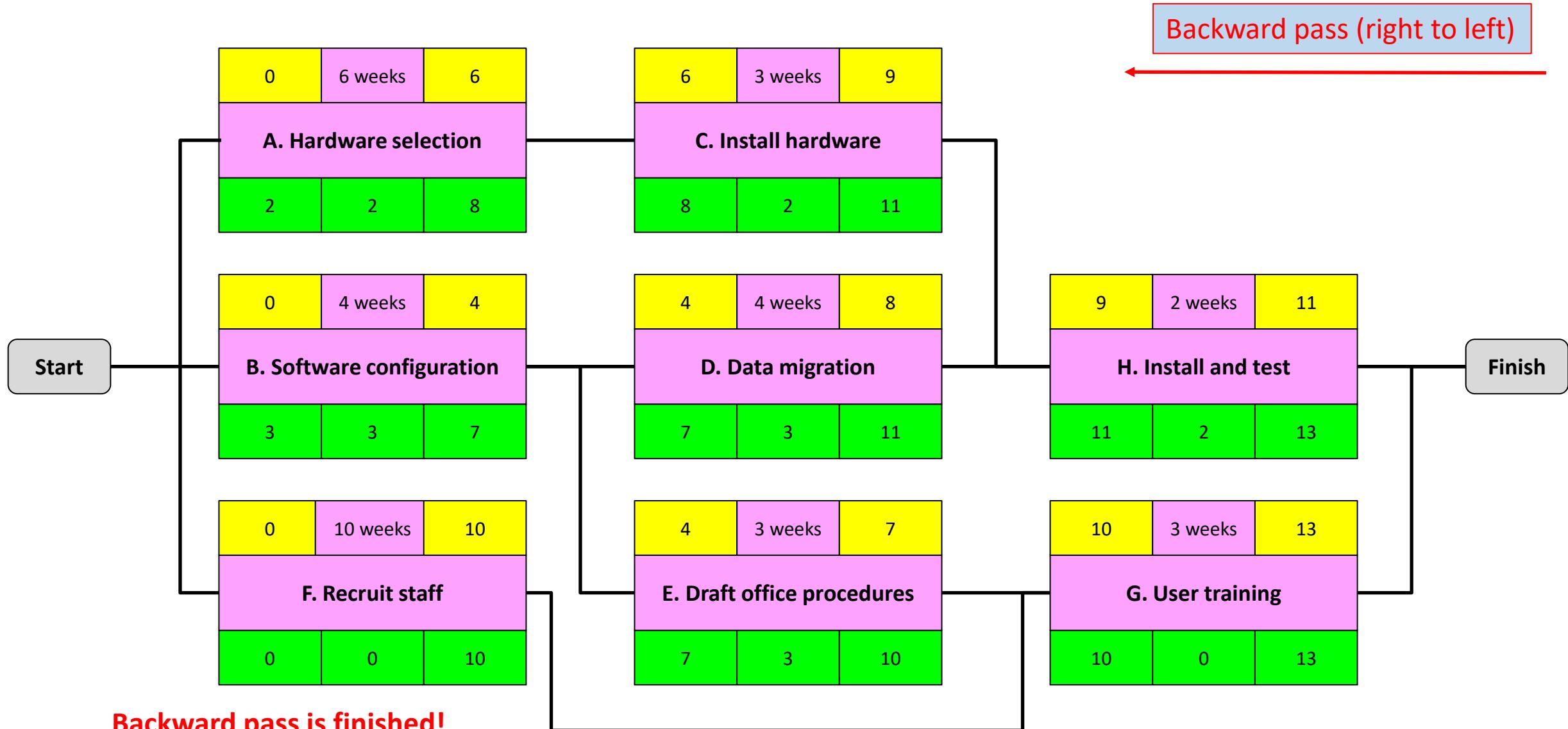
Latest start/finish and float calculated



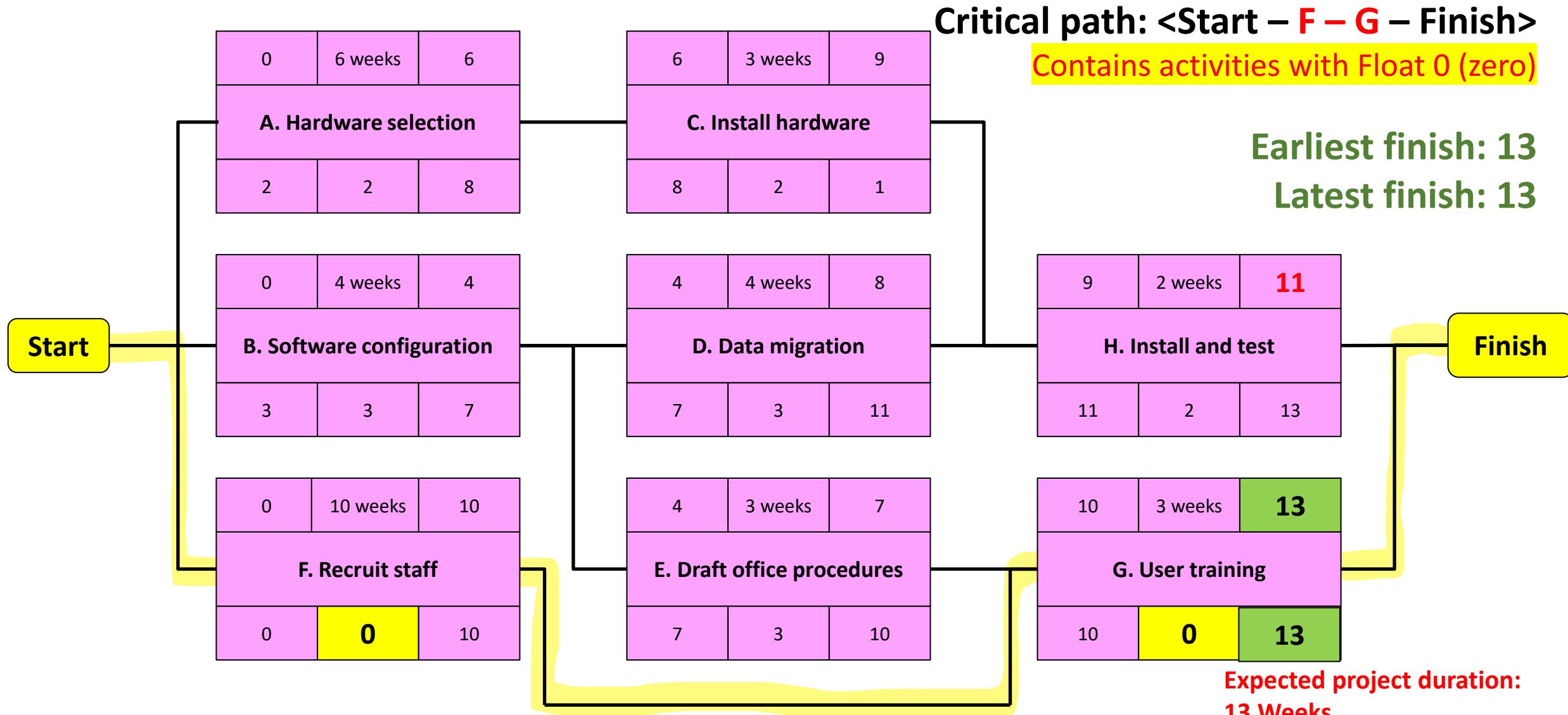
Latest start/finish and float calculated



Latest start/finish and float calculated



The Critical Path Analysis (CPA)



Activities

- Must have clearly defined start and end-points
- Must have resource requirements that can be forecasted: these are assumed to be constant throughout the project
- Must have a duration that can be forecasted
- May be dependent on other activities being completed first (precedence networks)

Rules for constructing activity networks

- Only one start and one end node – to avoid confusion
- Activities have durations – they do in real-world projects
- Time moves from left to right – convention to aid readability
- Loops are not allowed – to enable analysis
 - Finite loops specifying iterations can be “unfolded”

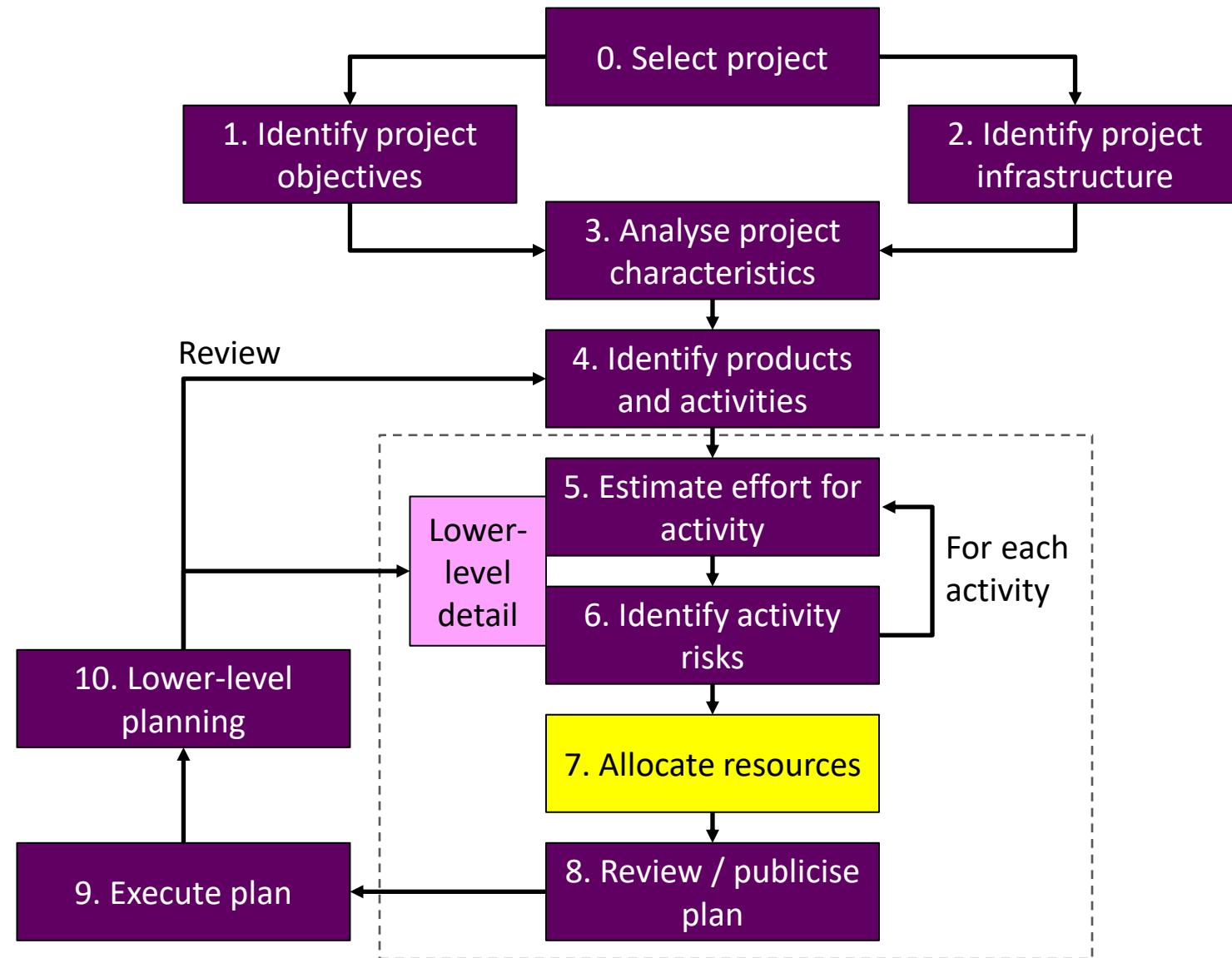
Steps to reduce the Critical Path

- Eliminate tasks on the critical path
- Re-plan serial paths to be in parallel
- Overlap sequential tasks
- Shorten the duration of critical path tasks
- Shorten early tasks
- Shorten longest tasks
- Shorten easiest tasks
- Shorten tasks that cost the least to speed up

Outline

- Frameworks for software project planning
- Selection of software project approaches
- Effort estimation for software projects
- Activity planning and resource allocation
 - Activity planning
 - **Resource allocation**

Steps 4 and 5 of step wise



Objectives of resource allocation

- Identification of the resources required for the project
- Making the demand for resources more even throughout the lifespan of the project
- Production of a work plan and resource schedule

Resources categories

- These include in general:

- Labour
- Equipment
- Materials
- Space
- **Services**
- Time
- Money

Examples resources:

- **Staff, team, developers, managers...**
- computers, hardware, software, tools, or frameworks
- raw materials, components, or consumables
- access to information and knowledge
- licenses, permits, or regulatory approvals
- communication tools
- basic utilities
- **suppliers, vendors, contractors, technical support ...**
- training materials
- risk assessment tools
- stakeholders support

Among others...

Resource allocation

- The project manager needs to:
 - Identify the resources needed for each activity and create a resource requirement list
 - Identify resource types (e.g. ‘VB programmers’ as opposed to ‘software developers’) – as individuals are interchangeable within the group
 - Allocate resource types to activities
- Resource clashes
 - Where the same resource is needed in more than one place at the same time
 - Can be resolved by:
 - Delaying one of the activities
 - Taking advantage of float to change start date
 - Delaying start of one activity until finish of the other activity that resource is being used on
 - puts back project completion
 - Can be resolved by moving resource from a non-critical activity
 - Can be resolved by bringing in additional resource - increases costs

Prioritising activities

- **Prioritising** is required when several activities are **competing for the same limited resource at the same time**
- Resources are allocated to activities in **decreasing priority order**. There are two main ways of doing this:
 - **Total float priority** – the activities with the smallest float have the highest priority; activities with the same float are processed in any order
 - **Ordered list priority** – this takes account of the duration of the activity as well as the float
- Typical priority list:





Aston University

BIRMINGHAM UK

Software Project Management

Unit 3: Software Project Planning (2)

Thais Webber
Richard Lee





Aston University

BIRMINGHAM UK

Software Project Management

Unit 4: Agile & Scrum (part 1)

Thais Webber
Richard Lee



Goals for this week

1. Describe the core ideas behind agile software development.
2. Begin to explain the structure, participants, artefacts, and events of the Scrum methodology.

What is agile development?

- Different focus:
 - Change is something to embrace; we determine what is best for the project as we go along
 - Instead of big-design-upfront and fixed deadlines/costs, focus on client-developer collaboration
 - Keep only the parts of the process that deliver value (no process for the sake of process)
- Examples of agile methodologies:
 - **Extreme Programming (XP)** (technical excellence)
 - **Scrum** (management, process)
 - **DSDM (*Dynamic Systems Development Method*)** (controlled agile option)
 - **Crystal** (there is no “one-size-fits-all” Agile method)
 - **Adaptive Software Development** – an evolution of RAD (focus learning)
 - **Pragmatic Programming** (developers encouraged to be flexible but disciplined, mindset)
 - **Kanban** (continuous flow agile, visualise and optimise)

Projects suited for agile delivery

- The most appropriate projects for agile are ones with aggressive deadlines, high degree of complexity, and high degree of novelty to them

- Novelty



- Urgency



- Complexity



Focus is on delivering the highest business value in the shortest time

Agile benefits

- Agility allows a better alignment between business objectives and IT
- Agility provides gains in terms of visibility, adaptability, business value, and risk reduction
- Costs reduction thanks to better effectiveness
- Time-to-market reduction
- Alignment with the market and business requirements
- Control over change cost
- Longer life software
- People aware of their responsibilities, more motivated
- Problems arise faster

...but success depends upon people rather than processes

Agile drawbacks

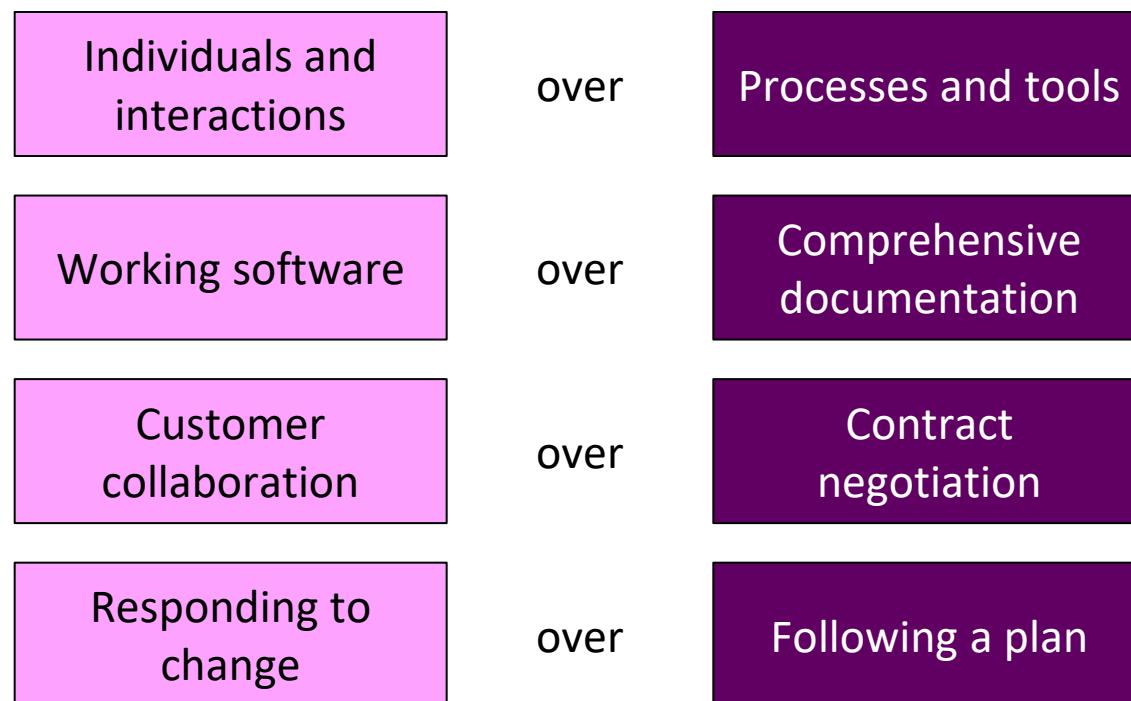
- Active user involvement and close collaboration is essential
- Requirements emerge and evolve
- Frequent delivery
- System structure tends to degrade as new increments are added
- Regular changes to the deliverable usually corrupt the structure unless time & money spent on refactoring

Quick poll vote – Agile discussion

1 – You prefer to deliver fast and tidy things up later?

2 – You would rather spend more time keeping everything clean from the start?

Agile Manifesto - values



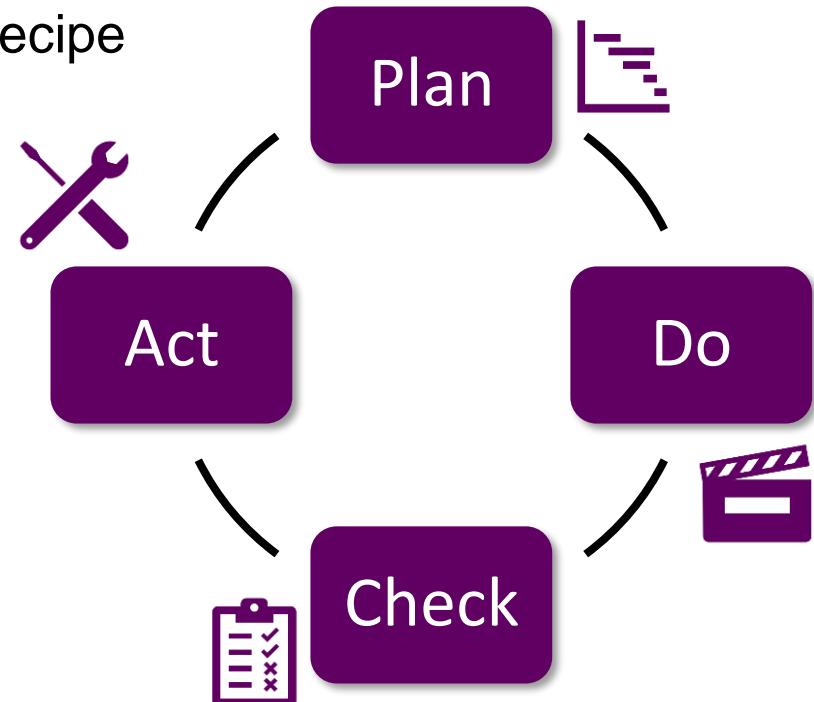
Other values

- Trust between:
 - Manager and employees
 - Colleagues
 - Customers and partners
- Transparency:
 - To identify and prevent obstacles
 - To be more effective
 - To facilitate collaboration
 - To adapt
 - **Not to watch or oversee**

Continuous improvement

- At the centre of empiric approaches
- Agility provides a framework, not a recipe
- Adapting practices according to:
 - the product we want to deliver
 - the skills and talents of the team
 - the conditions and the environment
- No agility without quality

(PDCA Cycle)

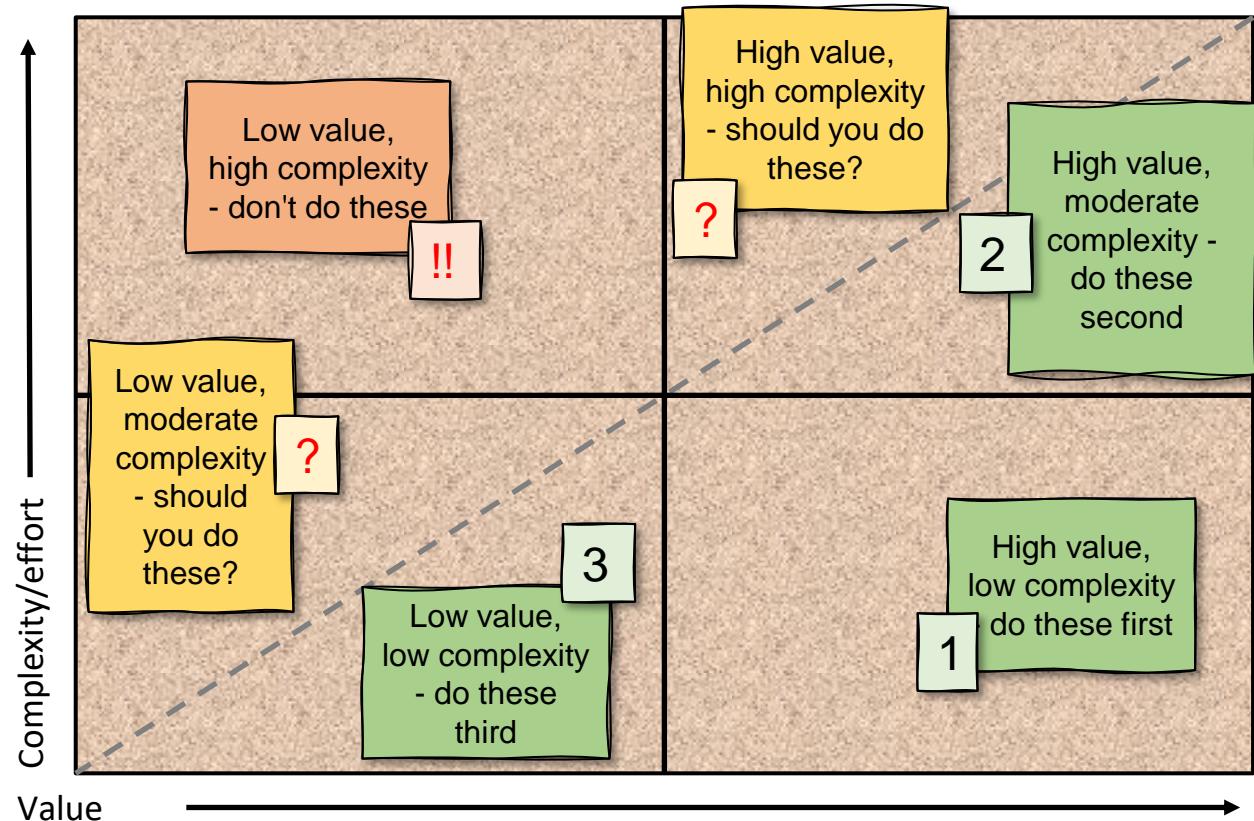


Sustainable pace

- Working at a steady and sustainable pace:
 - Promotes motivation
 - Increases effectiveness over time
 - Reduces the likelihood of risk
- Traditional project:
 - The closer we are to delivery, the greater the pressure
- Agile project
 - Pace and pressure are steady

System of prioritisation

- Visibility is central to agile:
 - Information shared with all
 - Problems arise easier
 - Actors feel responsible



- What is Scrum?

- Term originally comes from **rugby**
- Success in a scrum depends upon **working closely** together as a team

- The Scrum framework:

- Lightweight framework that promotes **lightweight, adaptive** solutions to complex problems
- Originated in **1995 (Sutherland and Schwaber)**
- Several books published by both

- Teamwork
- Coordination
- Trust

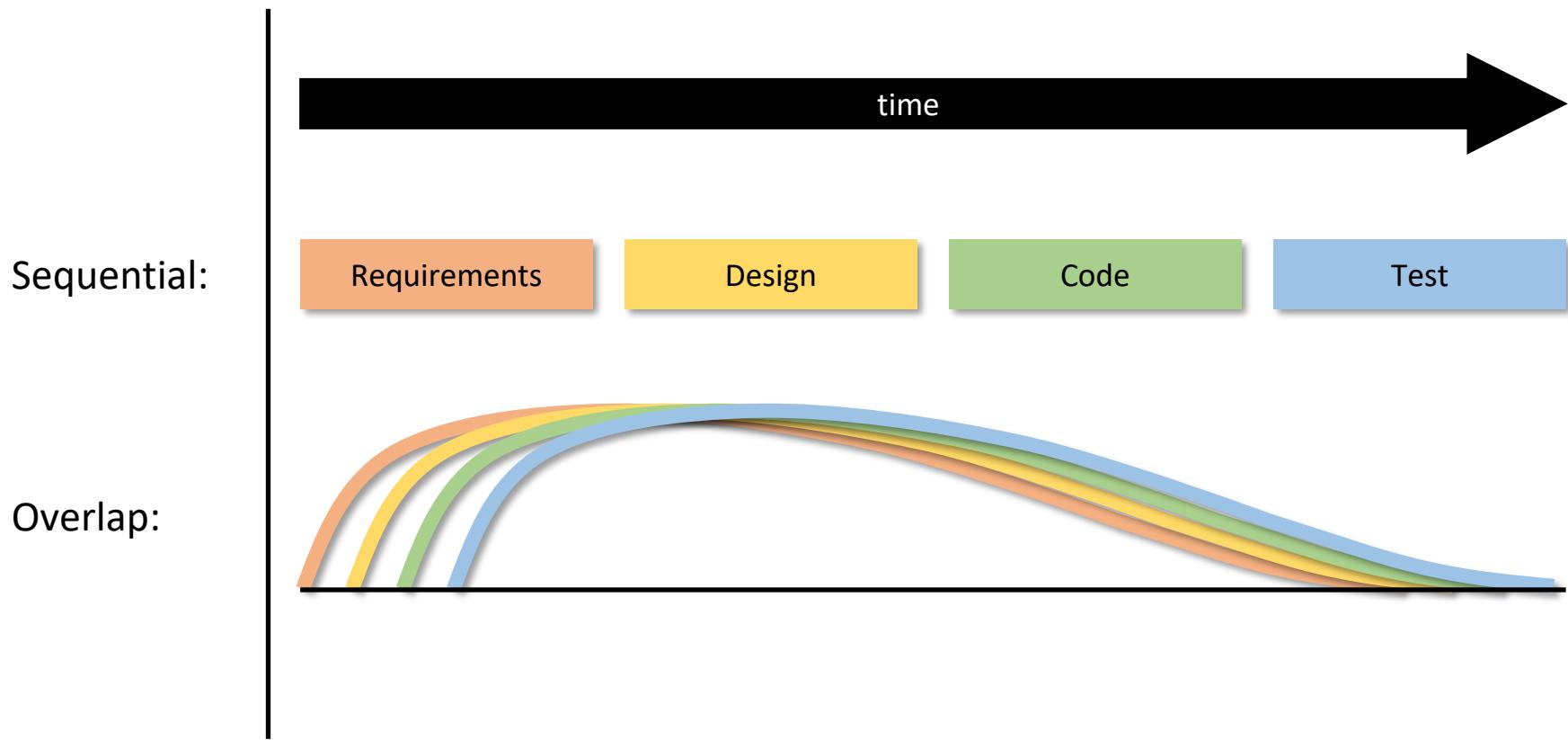
What is Scrum

- Is an agile, lightweight process
- Emphasis on collaboration
- Can manage and control software and product development
- Uses iterative, incremental practices
- Has a simple implementation
- Increases productivity
- Suited to complex product development
- Reduces time to benefits
- Easy to understand – hard to execute
- Embraces adaptive, empirical systems development

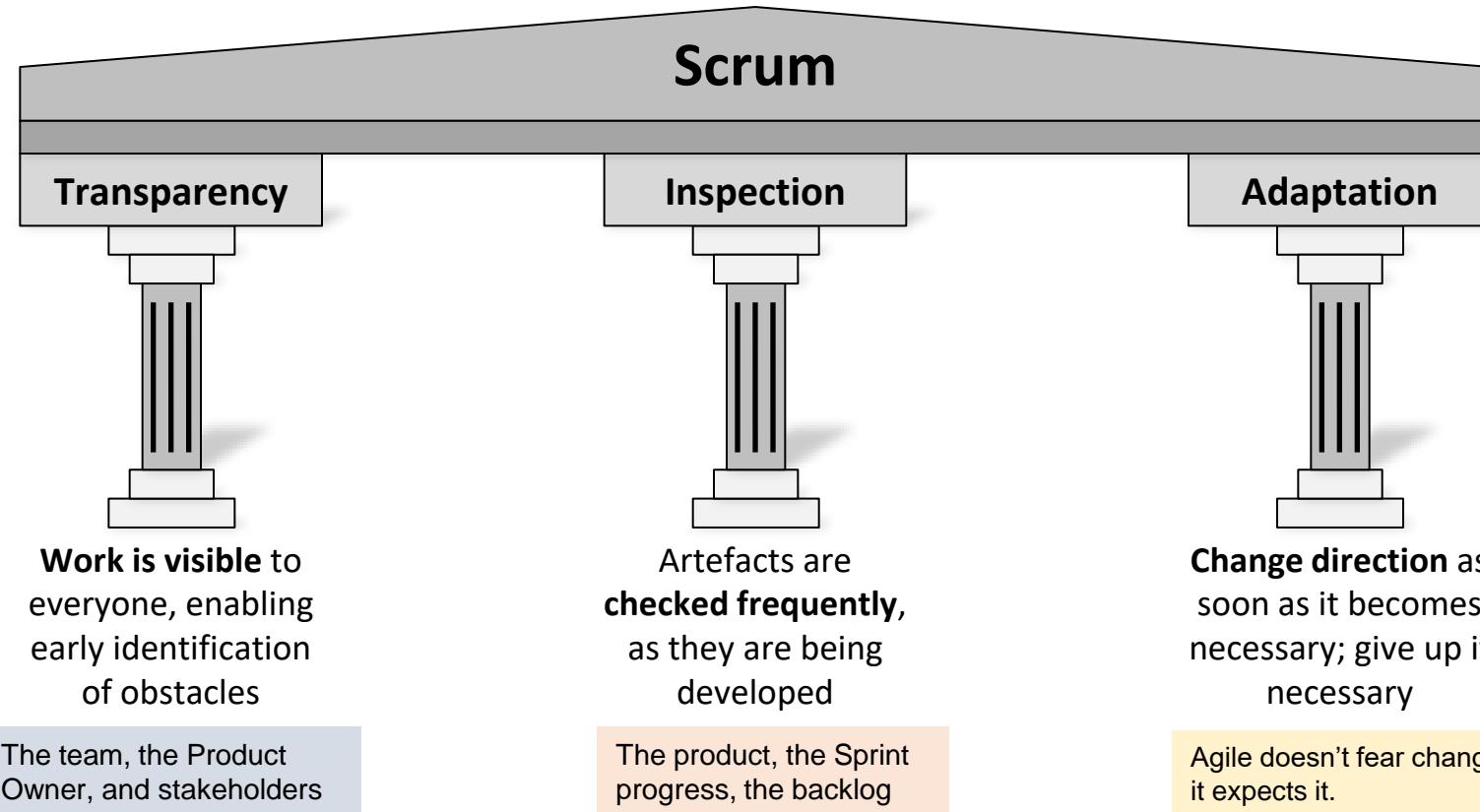
Values according to Scrum

1. **Commitment** to achieving the goals and supporting each other
2. **Focus** on the work of the Sprint to make the best possible progress toward the sprint goal
3. **Openness** about the work and the challenges you run into
4. **Respect** each other to be capable, independent people
5. **Courage** to do the right thing and work on tough problems

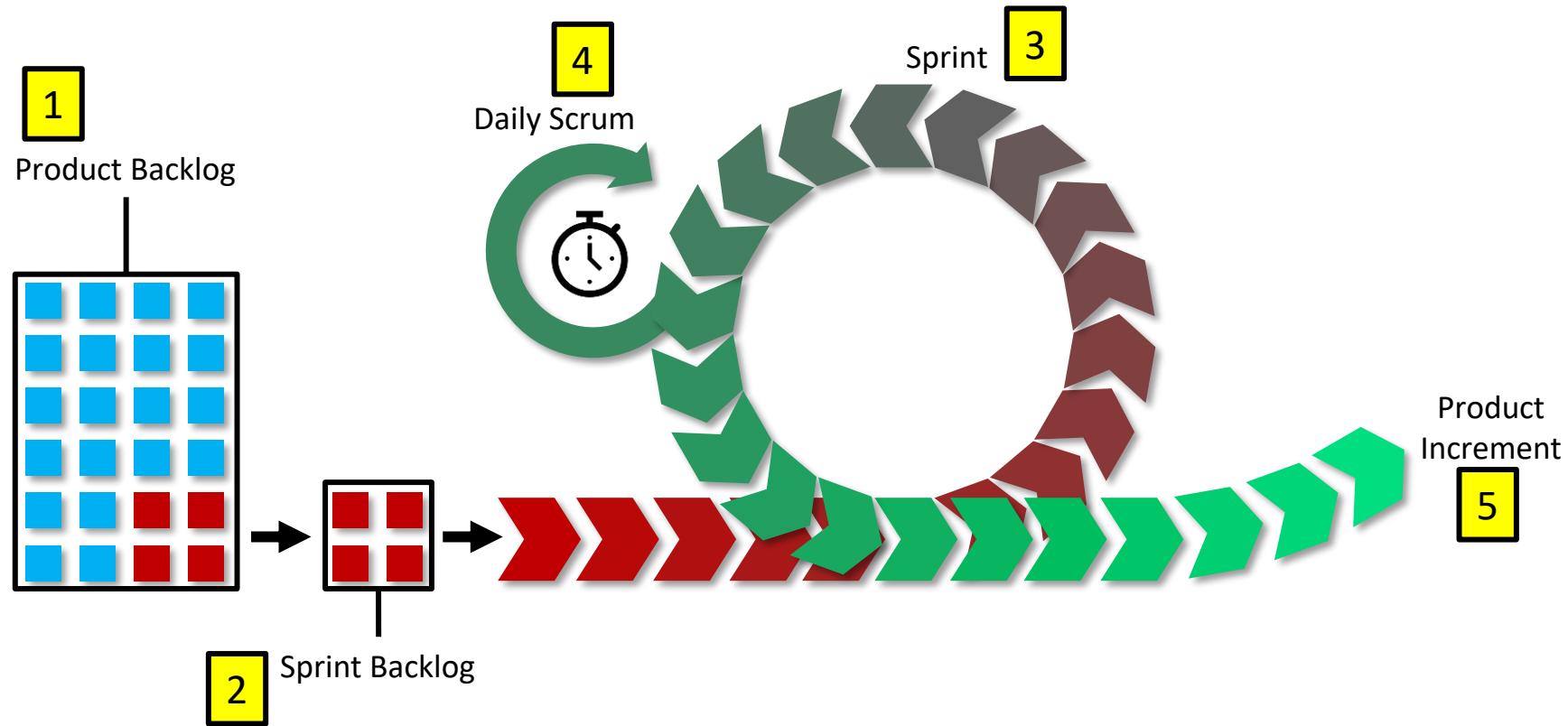
What is Scrum? Sequential vs. Overlap



The three pillars of Scrum

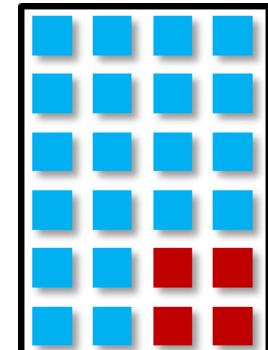


Product Backlog



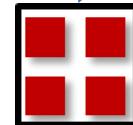
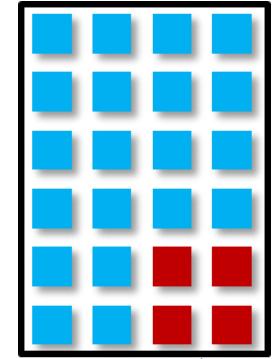
1 - Product Backlog

- Defines what will be built
- Contains a broad list of descriptions of all required features, wish-list items, etc. prioritised by business value.
- Product owner is in charge of defining priorities here
- Items here are gradually refined into items that can be completed in a sprint



2 - Sprint backlog

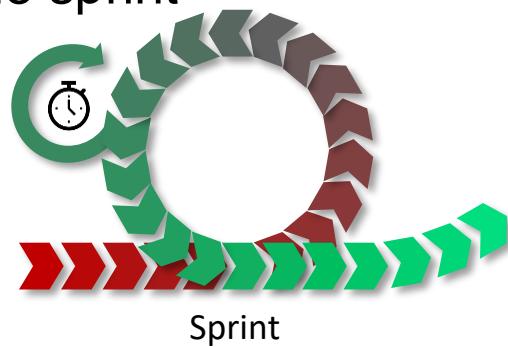
- Task items from the product backlog that will be completed in the next 'sprint'
- Collectively planned by the whole team
- Updated every day to reflect the team's latest decisions



Sprint backlog

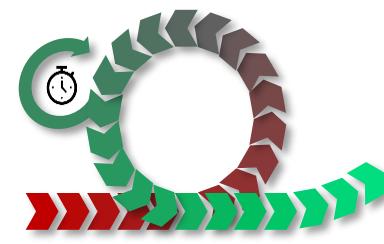
3 - Sprint

- Also known as an iteration, these are the units into which large, complex problems are broken down
- Occurs over a timeframe of typically 1-4 weeks, with a new sprint beginning as soon as one ends
- Team must respect the prioritisation agreed for the sprint backlog



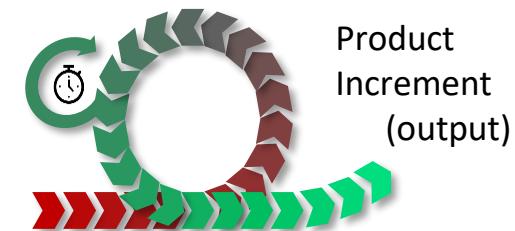
4 - Daily Scrum

- Also known as a 'stand-up meeting', this is a short but essential daily meeting involving everyone working on the sprint.
 - Updates on the previous day are shared
 - Plans for the day ahead are shared
 - Anything that has been learned since the last meeting, including problems, is discussed
- Run by an individual titled 'Scrum Master'
- An opportunity to reflect and plan, and ensure work is both synchronised and relevant to the sprint goal



5 - Product Increment

- The output of every sprint, which should be a deliverable, functional, usable product
- Each increment builds on all previous increments; the next increment should comprise everything that existed already, plus everything from the latest sprint
- Each increment should be a working version of the product that could, if necessary, be shipped.



fully tested, integrated, and meeting the definition of “done”

Roles

- **Product owner**

- Possibly a Product Manager / Project Sponsor / Key end-user
- Decides features, release date, prioritisation, budget

maximising value

- **Scrum Master**

- Typically a Project Manager or Team Leader
- Responsible for enacting Scrum values and practices
- Remove impediments / politics, keeps everyone productive

creating space
for productivity

- **Project Team**

- 5-10 members; Teams are self-organising
- Cross-functional: QA, Programmers, UI Designers, etc.
- Membership should change only between sprints

delivering value
through collaboration

- **Sprint planning**

- Determining what will take place during a sprint

- **Daily stand-up**

- What did I do yesterday, what will I do today, what obstacles am I facing?

- **Sprint review**

- Everyone who wants to attend can attend; accomplishments are shown

- **Sprint retrospective**

- Examining lessons learned for the next sprint

FYP and Agile practice – A few tips...

- Apply **Agile thinking** – plan in small, achievable steps
- Be ready to **adapt** when feedback or new ideas emerge
- Keep your **scope realistic** and deliver value early
- **Reflect regularly** – what's working, what needs improvement
- **Document your process** clearly — show evidence of iteration and learning
 - **From Agile:** Product Backlog, Sprint Log or Progress Journal, Reflections
 - **From Waterfall (still useful!):** Project Definition, Requirements document, meetings and decisions log, design diagrams (UML, mock-ups, flowcharts), Timeline (Gantt chart), Trello board – keep versioned documentation



Aston University

BIRMINGHAM UK

Software Project Management

Unit 4: Agile & Scrum (Intro)

Thais Webber
Richard Lee





Aston University

BIRMINGHAM UK

Software Project Management

Unit 4: Agile, Scrum & Kanban (2)

Thais Webber
Richard Lee

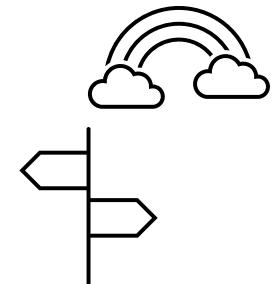
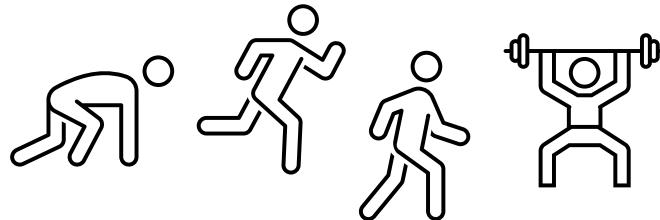


Goals for this week

- Explain the **structure, participants, events, and artefacts** of the **Scrum** methodology
- Understand **Kanban** principles and artefacts
- **Compare** the Scrum and Kanban methodologies, and understand their **advantages and disadvantages**

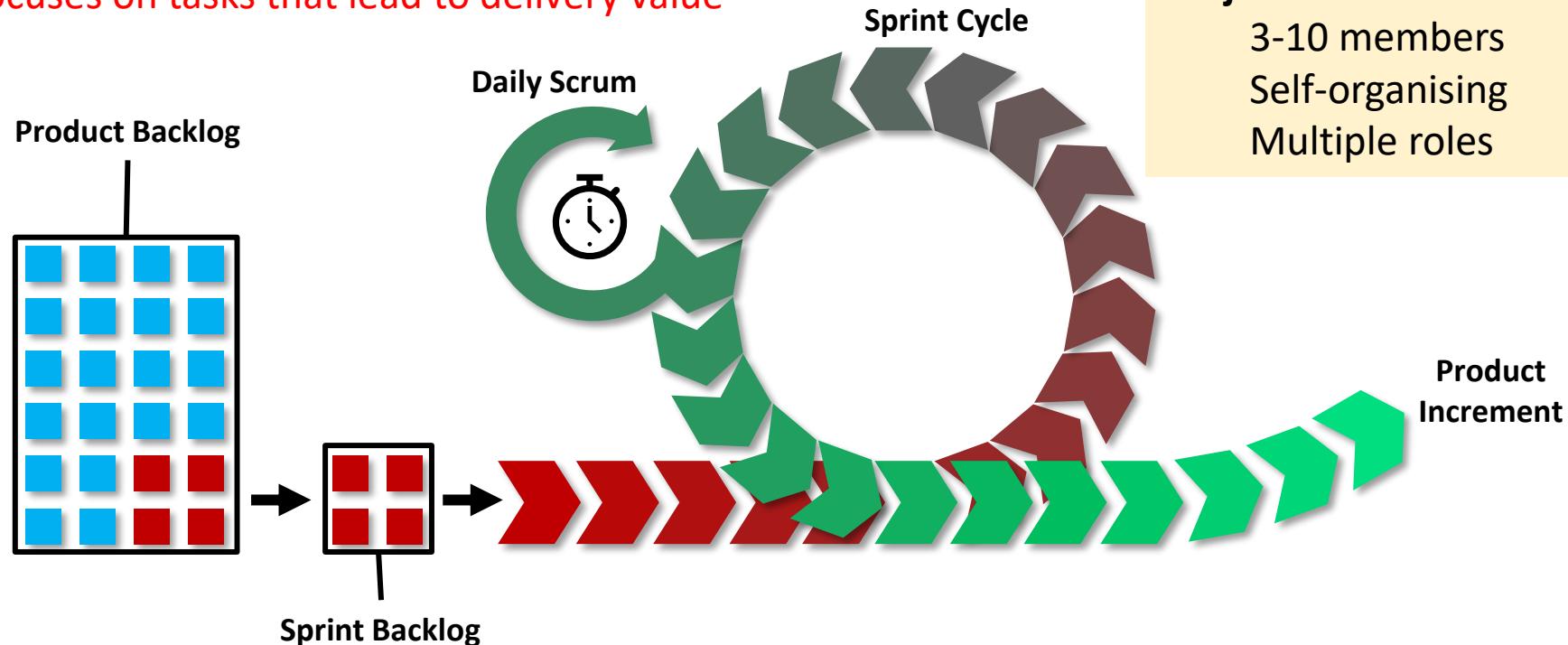
From past lecture... What is Scrum?

- An agile, lightweight framework
- Focuses on **teamwork, collaboration, and adaptability.**
- Uses **iterative and incremental** cycles to deliver value step by step.
- Helps teams **increase productivity** and **deliver benefits faster** through continuous feedback and improvement.

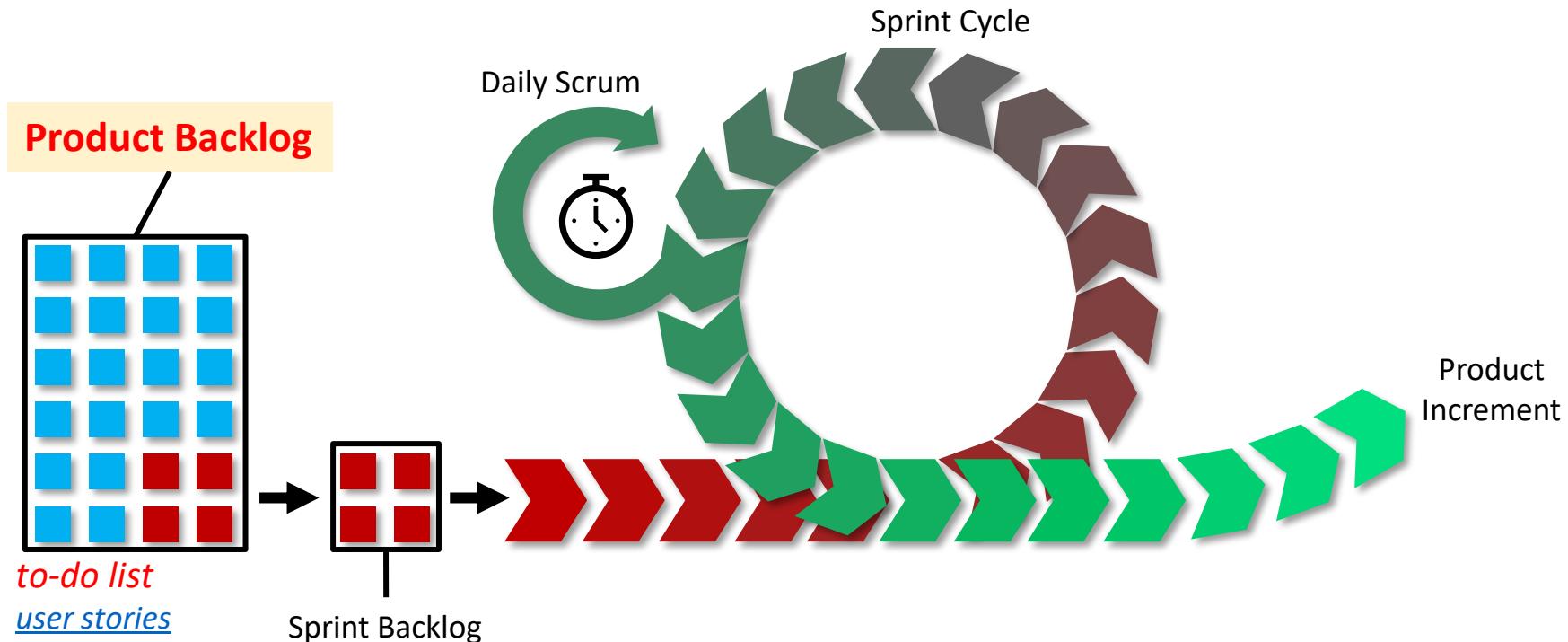


Structure of Scrum

Sprints are fixed time periods in which a team focuses on tasks that lead to delivery value



Structure of Scrum

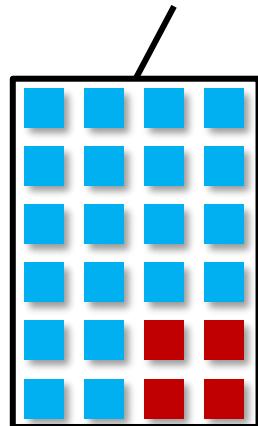


Structure of Scrum

Product Owner

responsible for maintaining the product backlog

Product Backlog

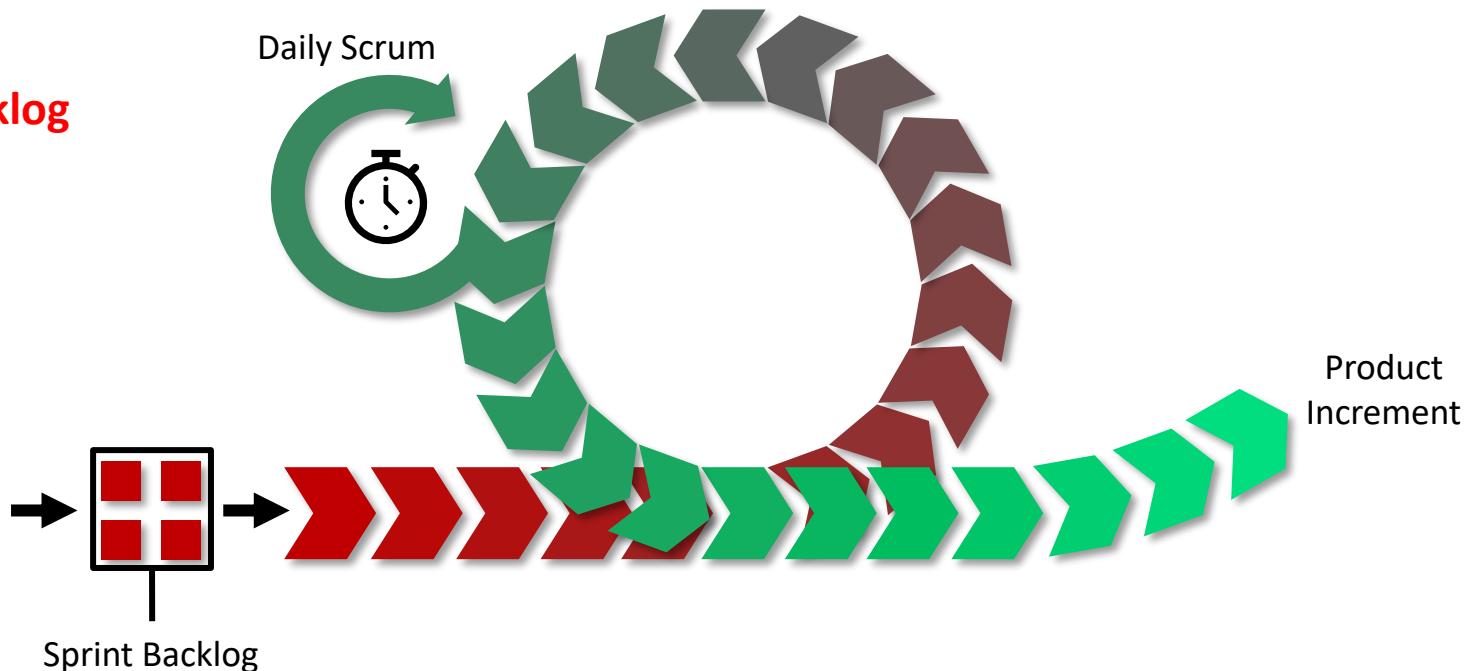


to-do list
user stories

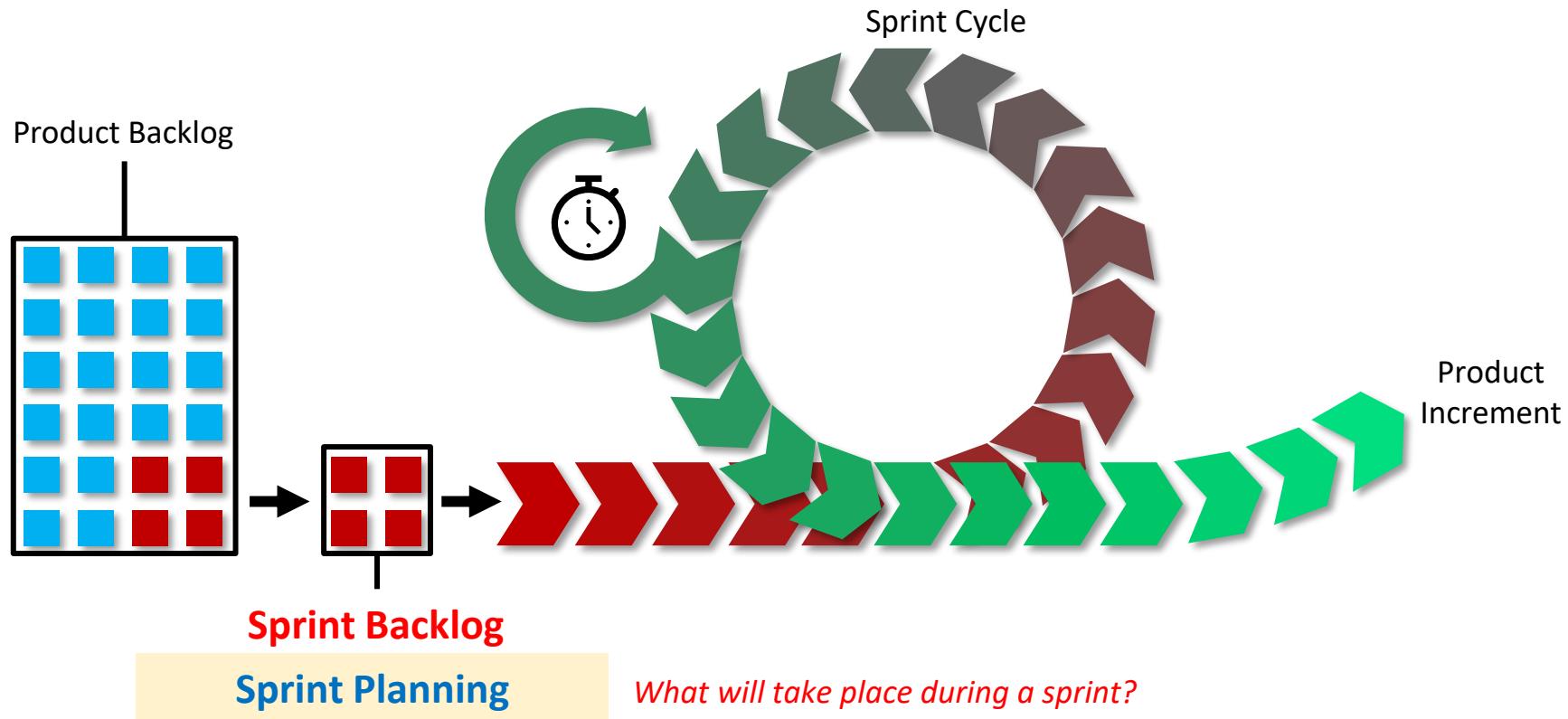
Daily Scrum



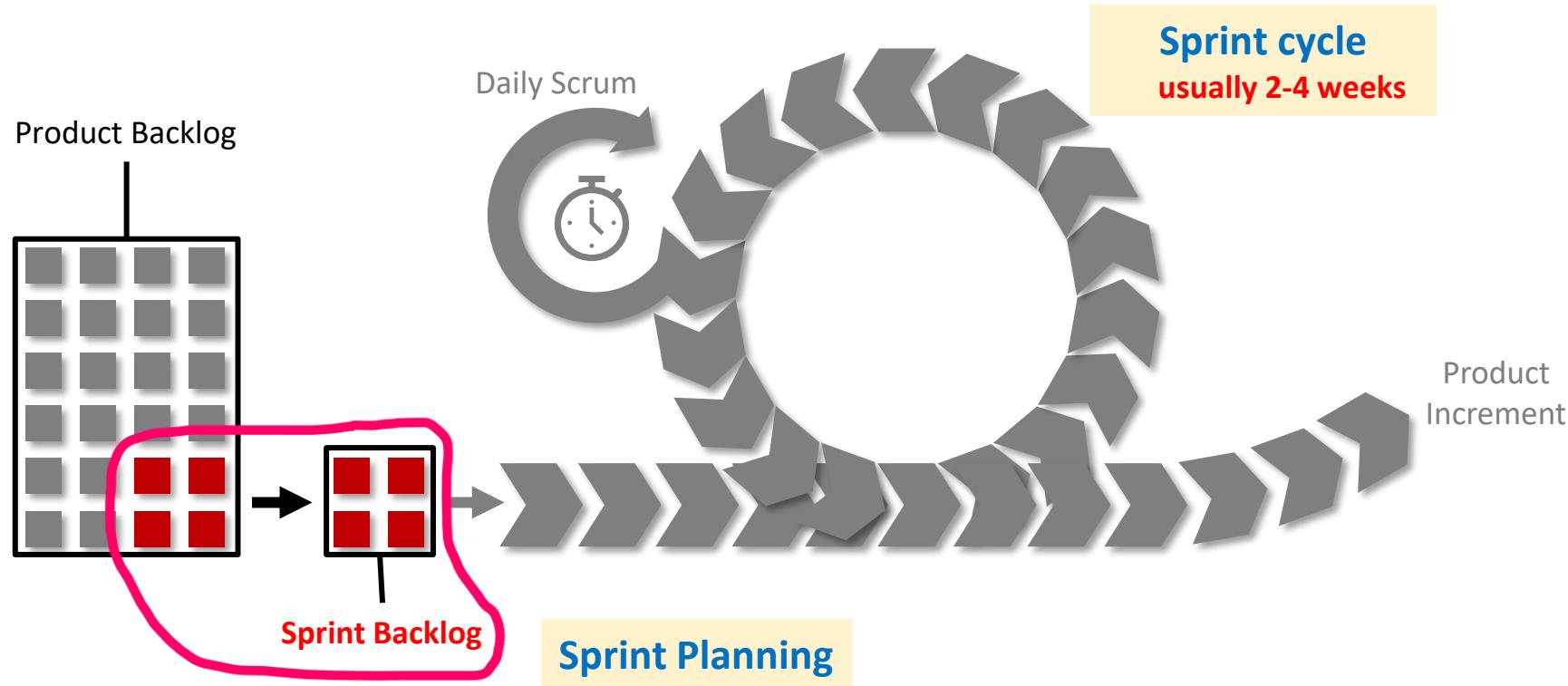
Sprint Cycle



Four key events in Scrum



Sprint Planning



Sprint Planning

- Provides a **short-term** vision: from 1 to 4 weeks
- Takes place **only at the beginning of an iteration**

Objective: set the **Sprint backlog** (i.e., user stories list) that the team are committed to develop during the sprint according to:

- Effort estimation defined for each user story
- Team capacity and constraints for that sprint

Input: Product backlog

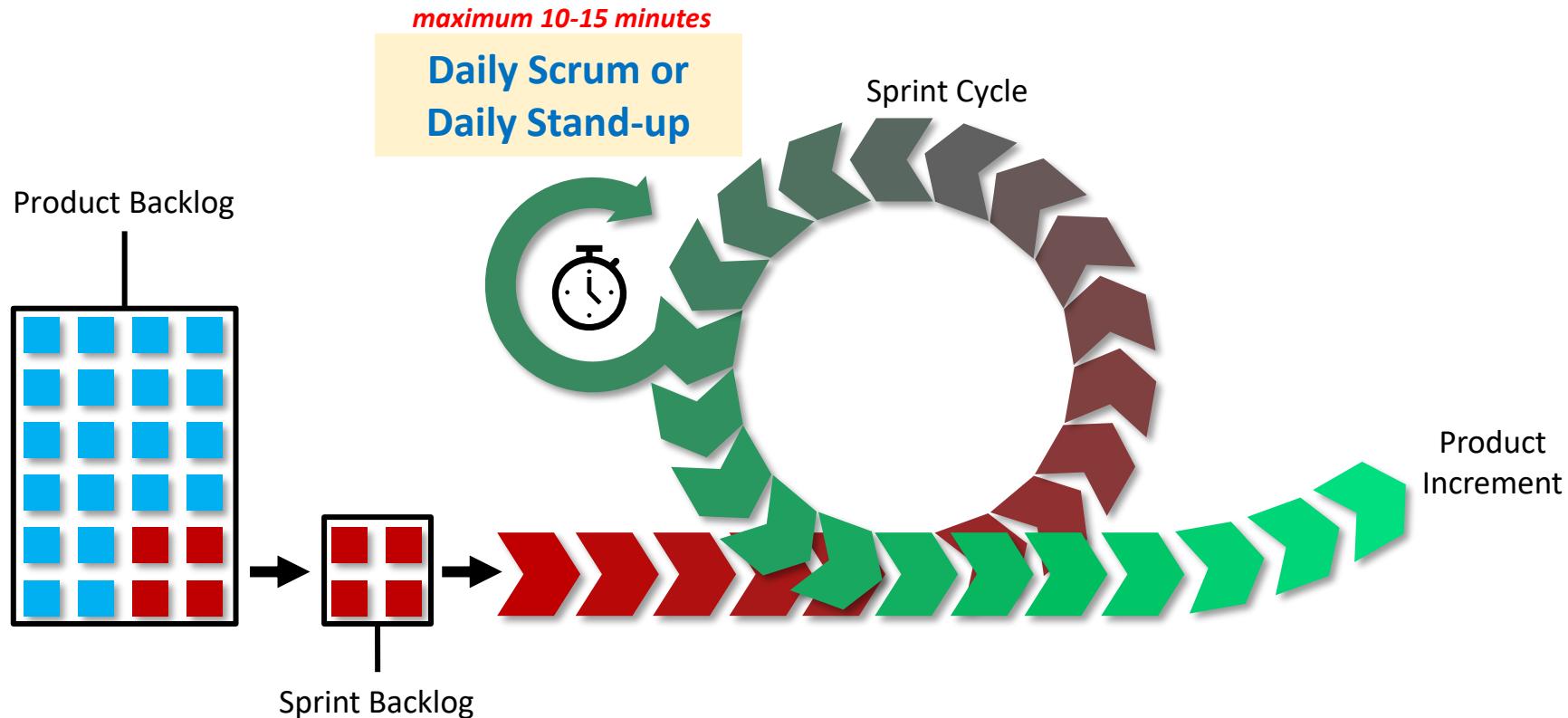
Output: Sprint backlog

Prerequisite: *Product backlog is up to date*

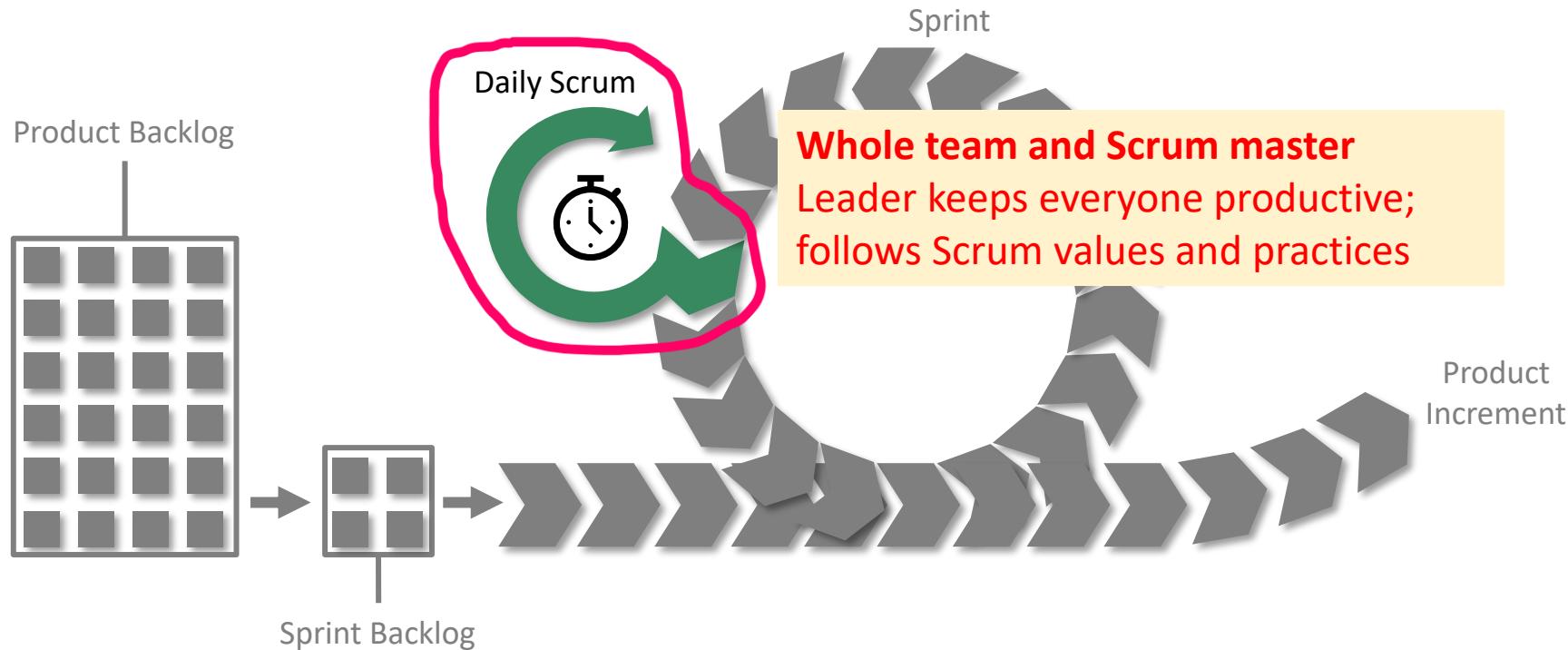
Sprint Planning

- Purpose:
 - To determine **how much work can be accomplished in the upcoming sprint** and answer the question 'how will we do it?'
- Outputs:
 - **Commitment** of several features, stories or bug fixes
 - **Sprint backlog** describing how to accomplish the commitment
- Rules
 - Product owner must come prepared with clearly defined, prioritised backlog items
 - Product owner has authority over the priorities for the next sprint
 - Team has authority over the amount of work that can be accomplished during the sprint
 - **Team must make a commitment before the end of the meeting**

Four key events in Scrum



Daily Stand-up



Daily Stand-Up

- Goal:
 - Enable the team to **share progress** with each other
 - Make visible the blocks (or impediments) - daily - for whole team to see
- Everyone stands in a circle and reports 3 things:
 - What did I do since the last Daily standup Meeting?
 - What will I try to do by the next Daily standup meeting?
 - Any issues that prevent us to progress?
- 10-15 minutes maximum

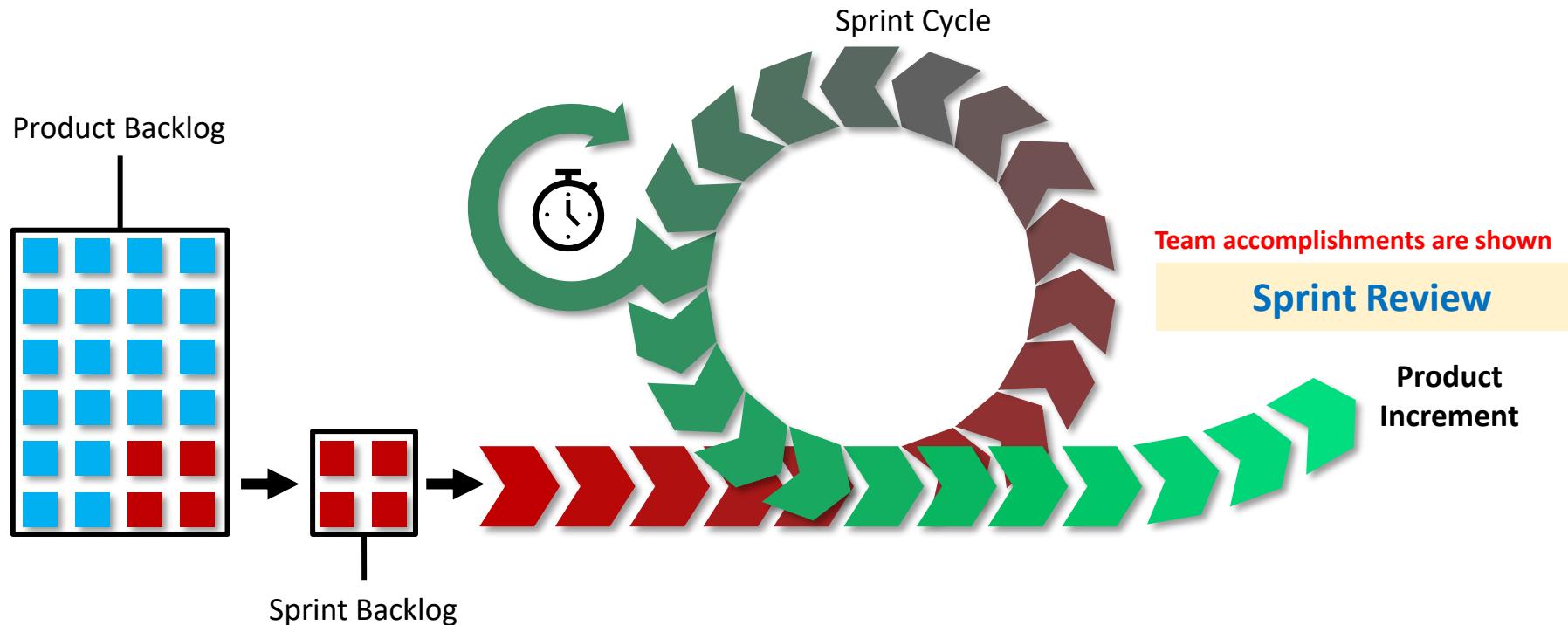
This is the only formal communication each day.

- **Informal communication may take place throughout the day** on issues or things that need to change (perhaps via Slack or Discord).

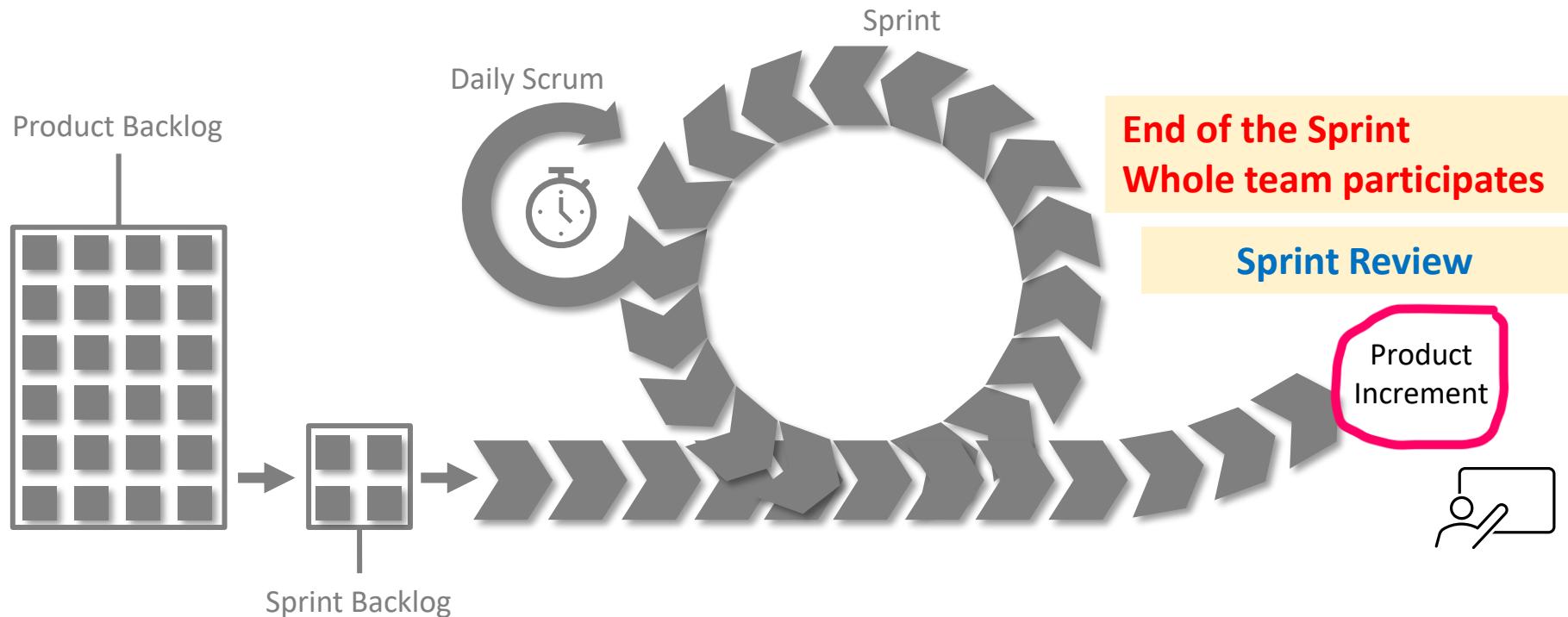
Daily Stand-Up

- No discussion or debate: **listening** only
 - After the meeting ends, discussion and problem-solving can begin
- Team and Scrum master only
 - Product owner can be invited, as can others, but that's up to the team
- After the meeting, the **Scrum Master leads the removal of issues/blocks**

Four key events in Scrum



Sprint Review

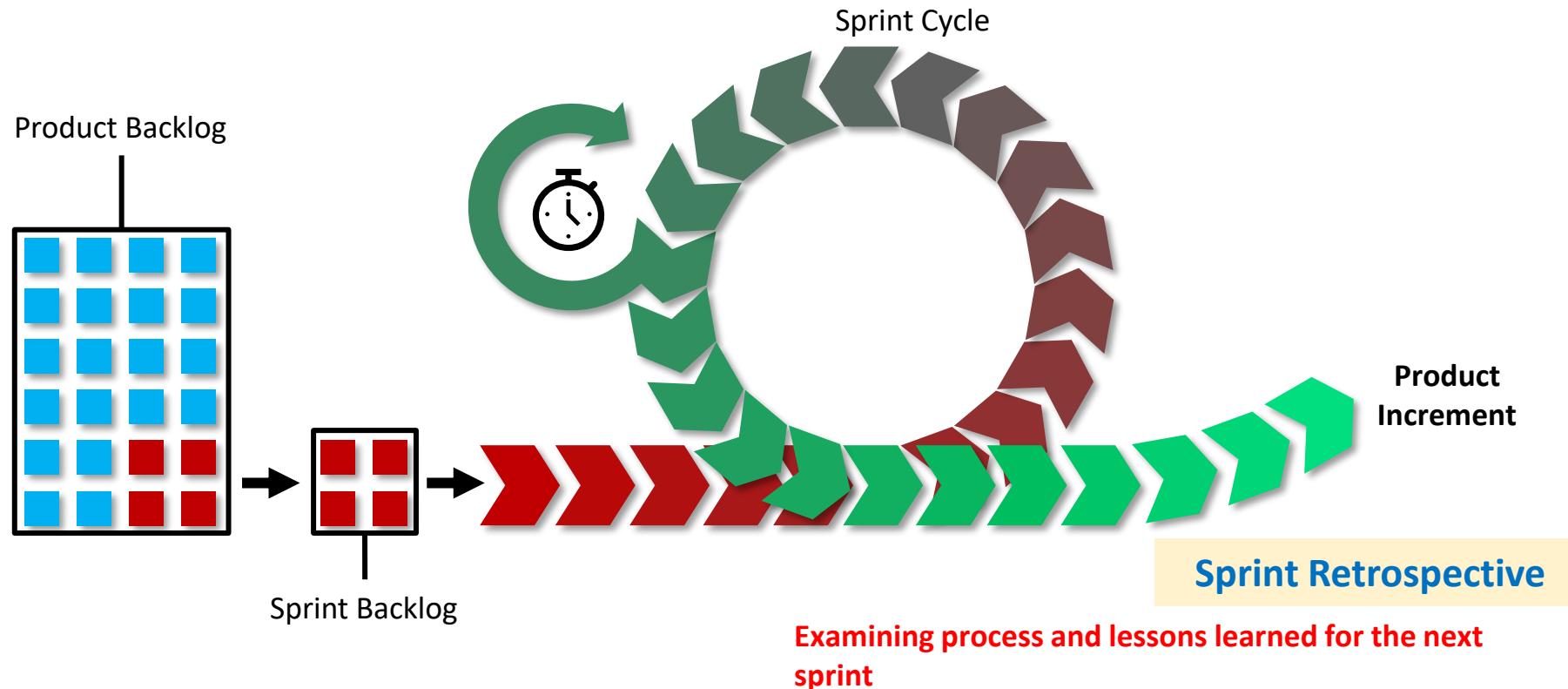


Sprint Review

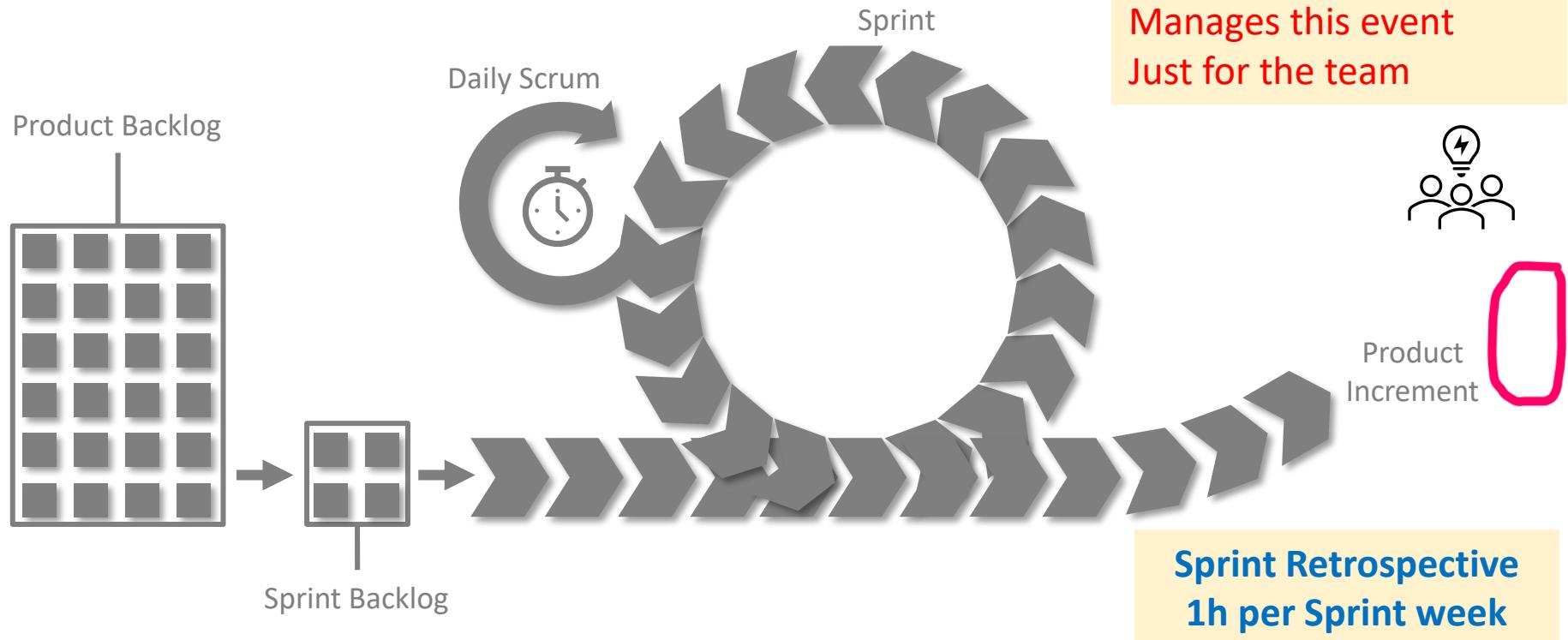
- Takes place at the end of the sprint
- Whole team participates
- Only completed user stories (product increment) are presented
- Team presents what it accomplished during the sprint to the business stakeholders and end-users for feedback and ideas
- Typically takes the form of a demo of new features
 - Informal atmosphere
 - Follows 2-hour prep time rule
 - No slides (usually)



Four key events in Scrum



Sprint Retrospective



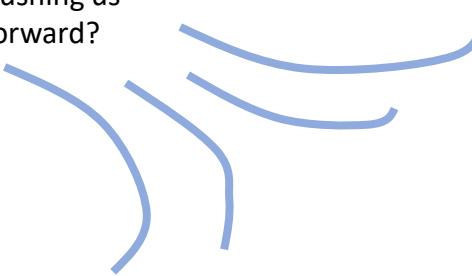
Sprint Retrospective

- Takes place **after the Sprint review**
- Managed by the Scrum master
- Duration: 1 hour per sprint week
- **Period of reflection** around Scrum project **progression**
 - At the organisational level and technical aspects
- Teams can address issues met during the sprint:
 - Issues are identified
 - Issues' importance evaluated
 - Solutions and decisions made
- Format: **brainstorming, sailboat, post-it...**



Sprint Retrospective (sailboat)

Wind: what is pushing us forward?



Anchor: what is holding us back?



Goal/vision: what are we working towards?



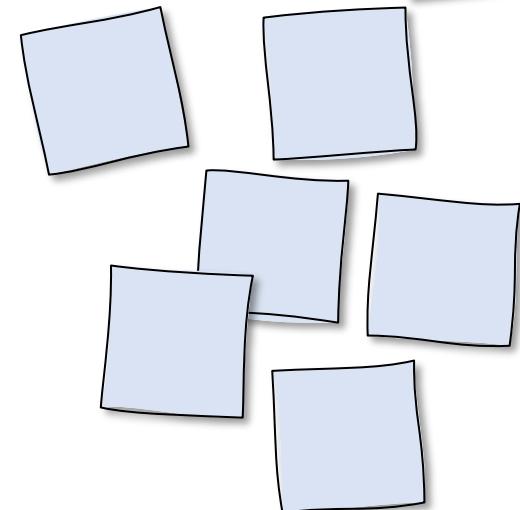
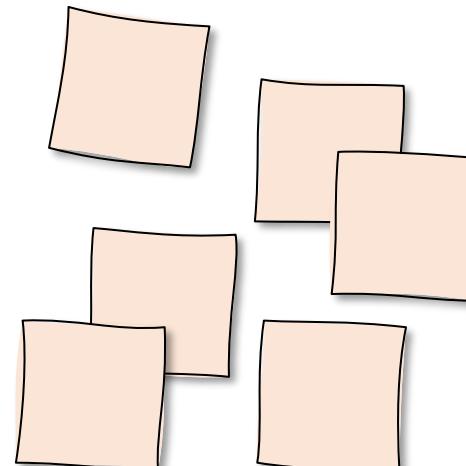
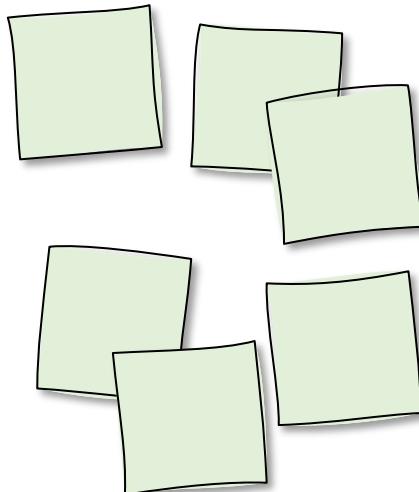
Rocks: what risks do we face?

Sprint Retrospective (post-its)

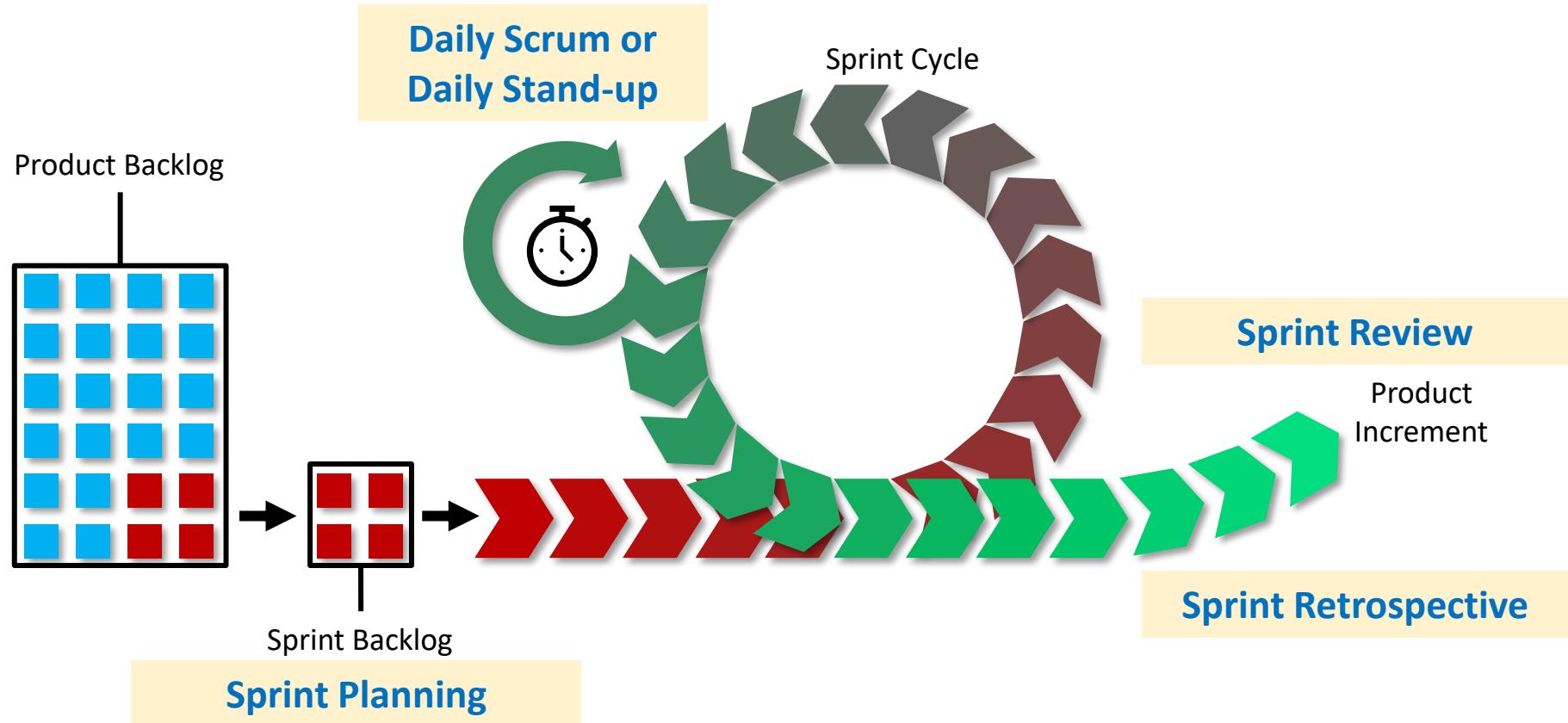
what
went
well

what
could
be
improved

Actions
for the
future



Four key events in Scrum



- **Sprint planning**

- Determining what will take place during a sprint

- **Daily stand-up**

- What did I do yesterday, what will I do today, what obstacles am I facing?

- **Sprint review**

- Everyone who wants to attend can attend; accomplishments are shown

- **Sprint retrospective**

- Examining lessons learned for the next sprint

Participants in Scrum

- **Product owner**

- Possibly a Product Manager / Project Sponsor / Key end-user
- Decides features, release date, prioritisation, budget

- **Scrum Master**

- Typically, a Project Manager or Team Leader
- Responsible for enacting Scrum values and practices
- Remove impediments / politics, keeps everyone productive

- **Project Team**

- 3-10 members; Teams are self-organising
- Cross-functional: QA, Programmers, UI Designers, etc.
- Membership should change **only** between sprints

- **Responsible for the ‘what’**
- One product owner per product (avoid mixed directions)
- Responsibilities and activities:
 - Accountable for ROI
 - Engaged with the final product and business users
 - Has a clear vision about the product and have a good understanding of the product
 - **Manages the Product backlog**
 - Prioritises the functionalities
 - Makes sure that the project team share the same vision and understanding related to the challenges
 - Formalises business needs with different granularity levels according to the importance of functionalities
 - Communicates about progress made

Scrum Master

- **Facilitator role**
- One scrum master per project
- Responsibilities and activities:
 - Involved in the product implementation
 - Engaged in team progress
 - Manages the project team
 - **Removes obstacles that can reduce team effectiveness**
 - Makes sure team members collaborate
 - Makes sure the product owner is always feeding the project team

Project Team

- **Cross-functional team** - includes design, coding, testing, and other resources required for potentially shippable software
- **Self-organising and self-managing**
- **Inspects and adapts** through Daily Scrum Meeting and Retrospective
- **Assists** product owner to prepare the backlog
- **Plans the sprint**
- Swarms over tasks – minimise Idle work
- Transparent, focused (no more than 2 tasks), works sustainably, **stays together**

Advantages of Scrum

- Enables projects where the business requirements are hard to quantify to be successfully developed
- Fast moving, cutting-edge developments can be quickly coded and tested
- Due to short sprints and constant feedback, it becomes easier to cope with changes
- Daily meetings make it possible to measure individual productivity. This leads to improvements in the productivity of each team member
- The overhead cost in terms of process and management is minimal, leading to a quicker, cheaper result.

Disadvantages of Scrum

- Scrum is one of the leading causes of scope creep because unless there is a definite end date, the project management stakeholders will be tempted to keep demanding new functionality
- If the team members are not committed, the project will either never complete or fail
- It is good for small, fast-moving projects as it works well with small teams
- This methodology requires experienced team members
- If any of the team members leave during a development, it can have a huge effect on the project development

Scrum – Advantages and Disadvantages

| Advantages | Disadvantages |
|--|---|
| • Works well when requirements are unclear or changing | • Risk of scope creep if there is no clear end date |
| • Allows fast coding and testing for new ideas | • Needs strong team commitment to succeed |
| • Short sprints and constant feedback make change easier to manage | • Better suited to small, experienced teams |
| • Daily meetings improve visibility and productivity | • Team member turnover can slow or harm progress |
| • Low overhead – lighter processes lead to faster delivery | • Requires discipline to avoid confusion or burnout |

Introduction to Kanban

- Kanban means ‘signboard’ or ‘visual card’ in Japanese
- Lean method for managing work — not only for software.
- Originally used for just-in-time car manufacturing at **Toyota** (late 1940s) where parts were manufactured only when demanded ('pulled')
 - This reduced work-in-progress stocks and quickly pointed to any bottlenecks hidden by traditional push-oriented schedules
- Adapted for software by David Anderson of Microsoft to handle software maintenance requests

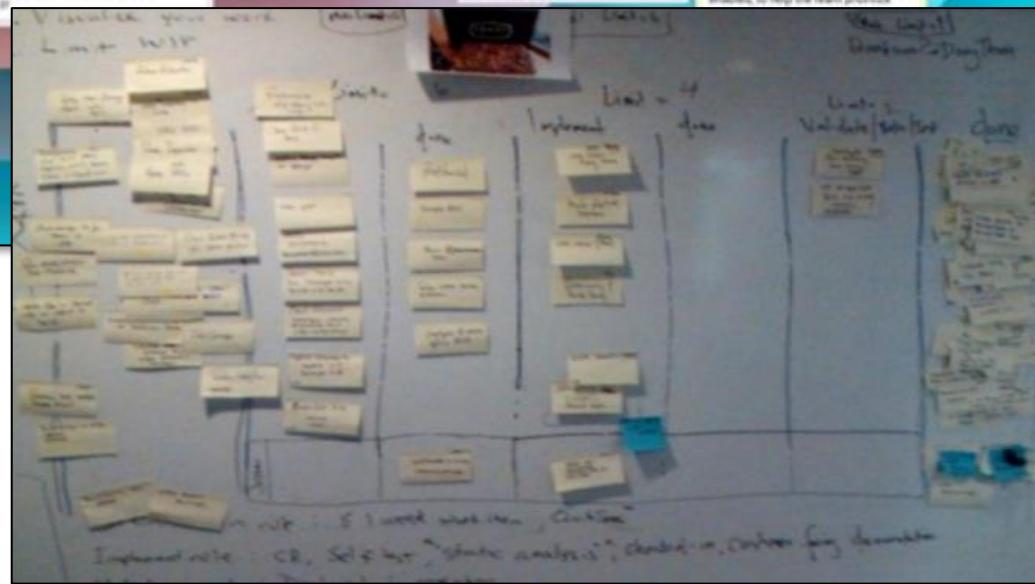
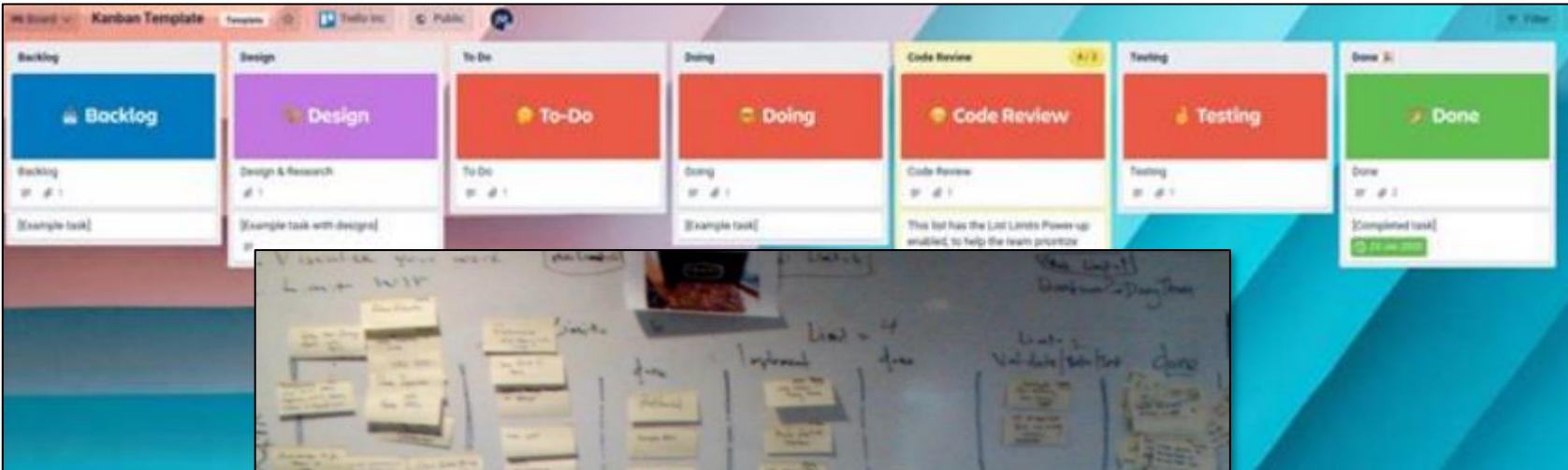
Kanban Steps

1. Capture high-level routines
2. Redecorate your wall (e.g., To Do, In Progress, Done)
3. Define **done**
4. Do your daily stand-ups

Kanban: capturing high-level routine

- We need to define the **pipeline** that works goes through
- For instance, when developing new features for your software:
 1. Take items from backlog
 2. Specify the work to implement it
 3. Implement it
 4. Validate that it works
 5. Deliver to customers or partners

Kanban: redecorate your wall



To-Do / Doing / Done
Pipeline visualisation
Limit to number of cards
- work-in-progress (WIP)

Kanban: define done

- Just like Scrum's **definition of done**
 - 'Done' means the specific requirements for moving a card from one stage to the next
- Examples of 'done' by stage:
 - **Specify (done)**: items broken into tasks doable within a day, and quick specs done for each item
 - **Implement (done)**: code reviewed and tested, static analysis conducted, code is checked in, customer-facing documentation is completed
 - **Validate (done)**: deployment to production, trial by real customers, issues identified are resolved

Kanban: stand-ups

- Project manager **asks for any blockers** and **helps solve** these
 - (That's it: there's no discussion of who's doing what, as that's on the board)
- Should take **5-15 minutes**
 - Longer discussions can take place separately, involving only those to whom the discussion is relevant
- If a particular step or person keeps getting **blocked** (or takes **longer than expected**), further analysis should take place:
 - » Was the work not broken up evenly?
 - » Are there inefficiencies in the process?
 - » Can someone else step in and help?

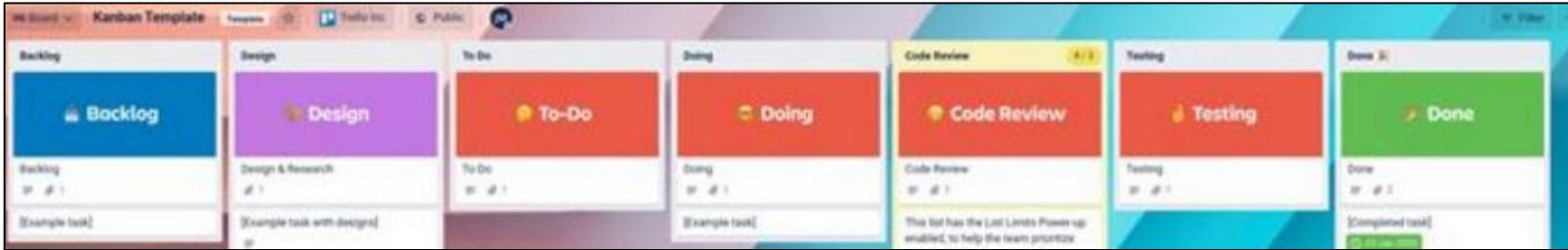
Advantages of Kanban

- **Ease of use:** Kanban is a very simple and easy to understand approach, which makes it practical to apply effectively. No need to be an expert to work with Kanban.
- **Promotes continuous and sustainable improvements:** Kanban approach not only consists of manual cards but also draws visualizations of the process outputs which makes the analysis of work easier.
- **Adaptability:** Kanban encourages maximum adaptability
- **Collaboration:** Kanban focuses on collaboration; this makes the team work together to produce the ideal outcomes/deliverables.
- **Low Overheads:** Supervision of the use of a Kanban board, cards, and analysis of output is easier as compared to most methods/approaches to project management.

Disadvantages of Kanban

- **Lack of iteration:** building software in iterations is a foundation for most development processes, which is not integral to Kanban at a ticket level. You can build iteration on top of Kanban, but it often ends up being its own separate process
- **Lack of timing:** there are no timeframes associated with each phase, which can be disadvantageous
- **Dependency on the board:** if the board is not updated, or is too simplistic to begin with, any advantages that come from Kanban are lost

Kanban – Advantages and Disadvantages



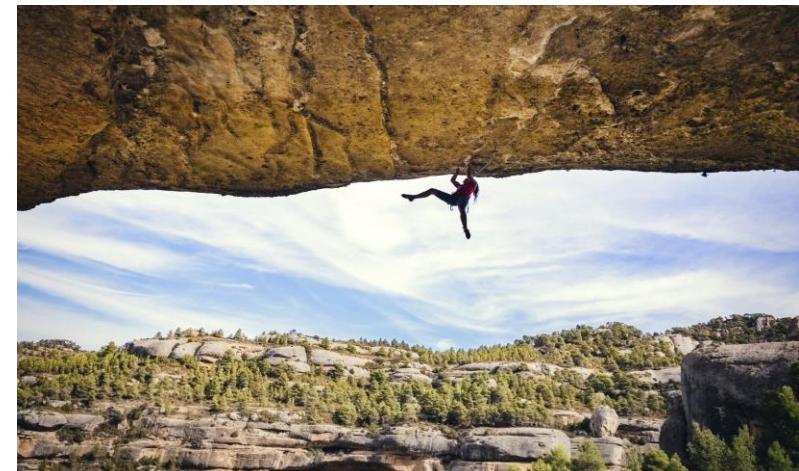
| Advantages | Disadvantages |
|------------------------|-------------------|
| Ease of use | Lack of iteration |
| Continuous improvement | Lack of timing |
| Adaptability | Board dependency |
| Collaboration | |
| Low overheads | |

Scrum vs Kanban

| | <u>Scrum</u> | <u>Kanban</u> |
|----------|---|--|
| Type | <ul style="list-style-type: none">Agile framework with defined structure and events | <ul style="list-style-type: none">Agile method focused on visualising and managing workflow |
| Key idea | <ul style="list-style-type: none">Work in short, planned sprints | <ul style="list-style-type: none">Maintain continuous flow of work |
| Roles | <ul style="list-style-type: none">Clear roles; team setup contains Product Owner, Scrum Master, Developers | <ul style="list-style-type: none">No fixed roles; flexible team setup |
| Planning | <ul style="list-style-type: none">Work is planned before each sprint | <ul style="list-style-type: none">Work is pulled as capacity allows |
| Focus | <ul style="list-style-type: none">Delivering increments of value at regular intervals | <ul style="list-style-type: none">Keeping steady progress and improving flow |
| Changes | <ul style="list-style-type: none">Not allowed during a sprint | <ul style="list-style-type: none">Can be made anytime |

Upcoming sessions overview

- **Next tutorial session scheduled:**
 - **Friday 9:00-10:00** – Reviewing Agile principles and approaches
 - Q&A forums – please your participation is highly encouraged!
- **Coming up next:**
 - **Monday 9:00-11:00**
 - **Risk Management in Software Projects**
 - Learn how to identify, assess, and respond to potential risks before they impact project success.





Aston University

BIRMINGHAM UK

Software Project Management

Unit 4: Agile, Scrum & Kanban (2)

Thais Webber
Richard Lee





Aston University

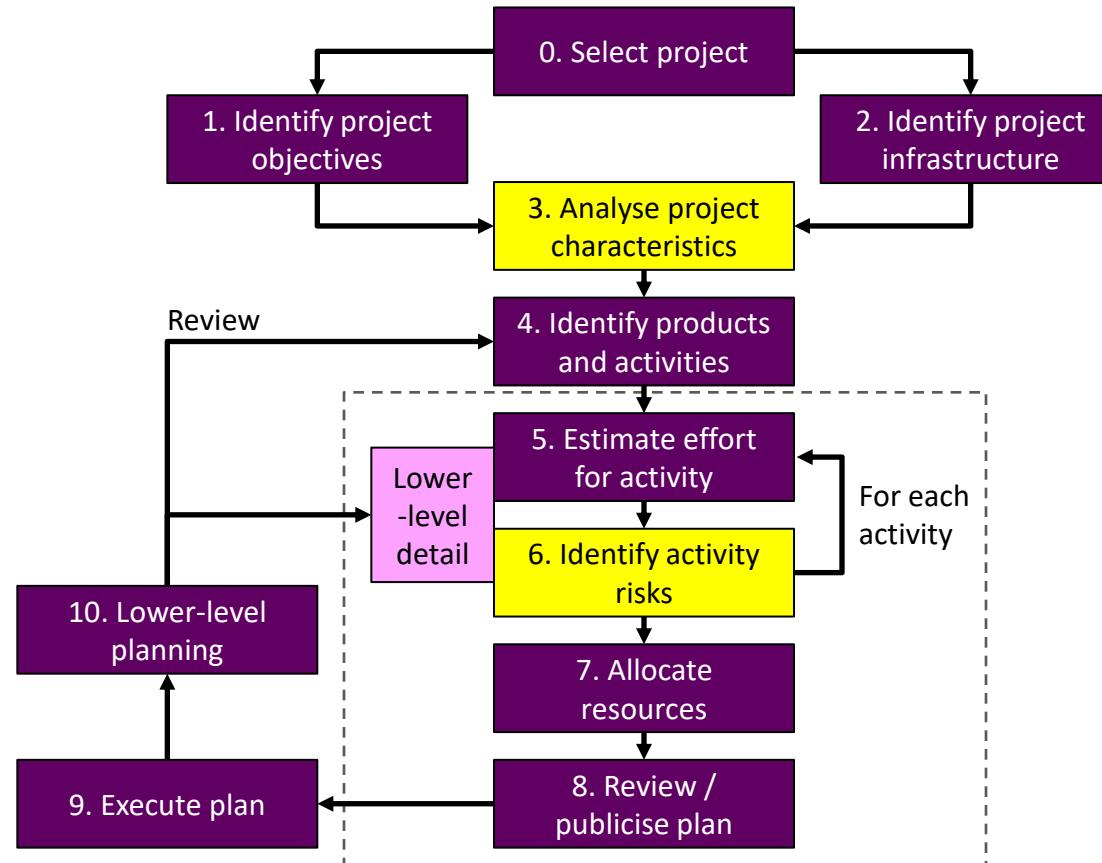
BIRMINGHAM UK

Software Project Management

Unit 5: Risk Management

Thais Webber
Richard Lee





Unit Objectives

- Define risk and risk management in the context of software projects
- Categorise software project risks based on their impact and probability
- Understand risk management steps including risk identification, risk analysis, risk planning and risk monitoring

Outline

- **Introduction**
- Categories of risk
- Dealing with risk in software projects
- Evaluating risks to the schedule using PERT

Definition of Risk

- The chance of **exposure** to the adverse consequences of future events (PRINCE2)
- An **uncertain event or condition** that, if it occurs, has a **positive or negative effect** on a project's objectives (US Project Management Body of Knowledge standard – PM-BOK)
- Risks can be negative or positive
- Risks relate to possible **future problems**, not current ones
- Risks involve a **possible cause and its effect(s)**

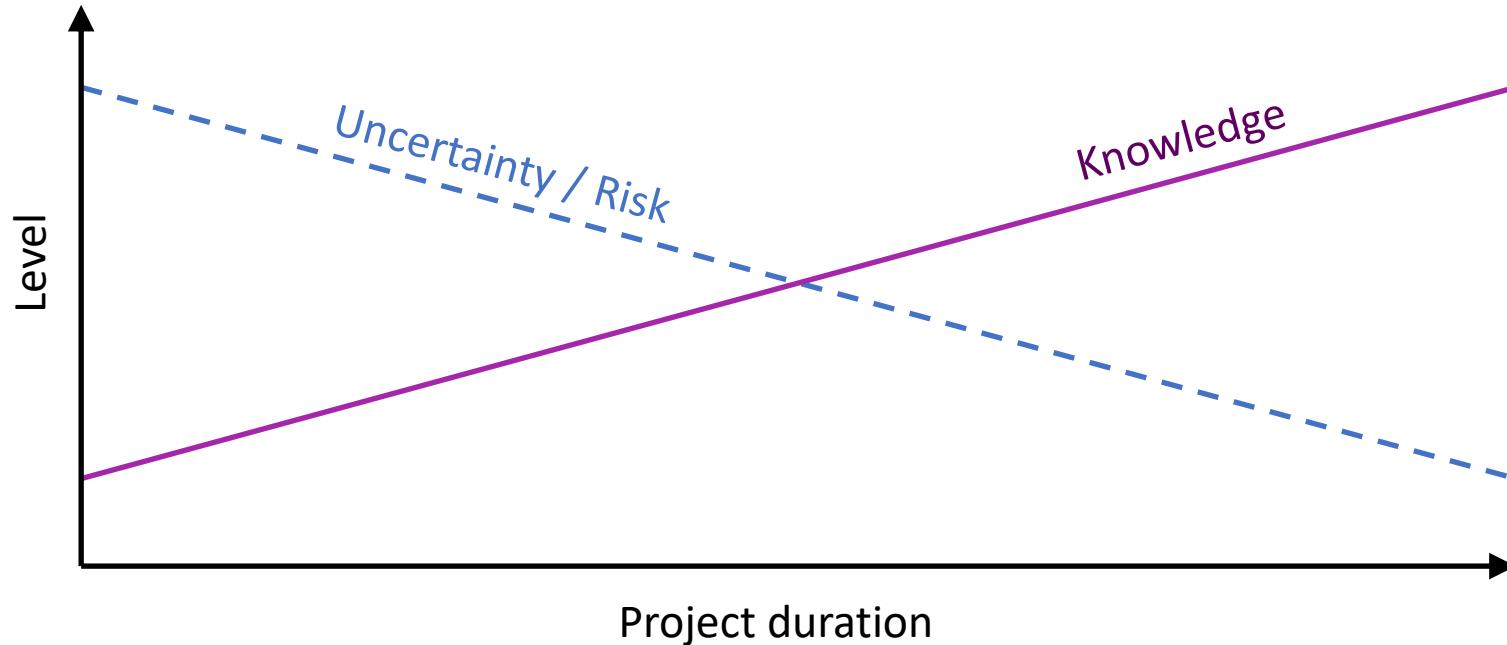
Negative Risk

- Negative risk involves understanding **potential problems** that might occur in the project and how they might impede project success
- Example : Key staff are ill at critical times in the project

Positive Risk

- Positive risks *can* result in good things happening; sometimes called opportunities
- Positive risk management is an investment
- Example : a new technology developed saving you time if released

Risk changes as the project progresses



Why is risk planning not widely used?

- Lack of **awareness** of the approach
- Unwillingness to spend **additional time and resources** on risk management
- Development managers may want projects to go ahead and do not want project sponsors to be deterred by consideration of possible failure
- If successful, you might not experience any tangible benefit, in spite of there being a cost associated with its use (which is actually a risk)

Project Risk Management

- Project Risk Management includes the **processes** concerned with identifying, analysing, and responding to project risk
- It aims at:
 - maximizing the results of positive events
 - minimizing the consequences of adverse events

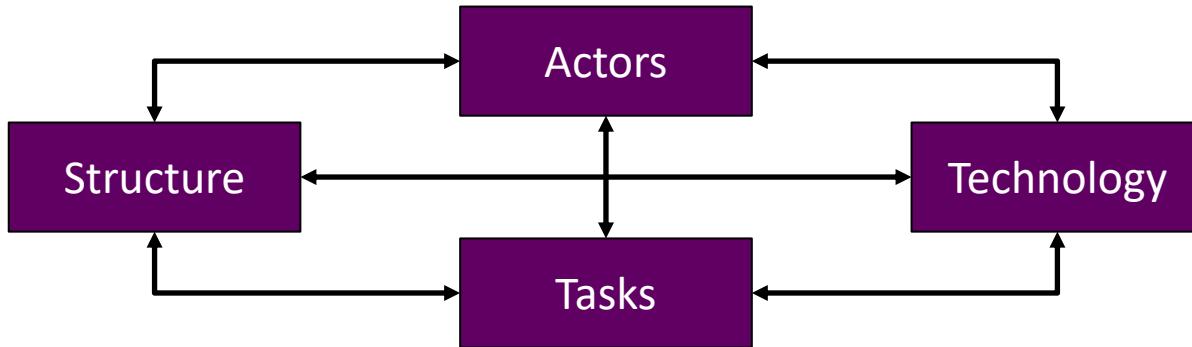
Project Risk Management

- Project Risk Management includes the **processes** concerned with identifying, analysing, and responding to project risk
- It aims at:
 - maximizing the results of positive events
 - minimizing the consequences of adverse events
- Risks factors:
 - Project size
 - Complexity
 - Speed of implementation

Outline

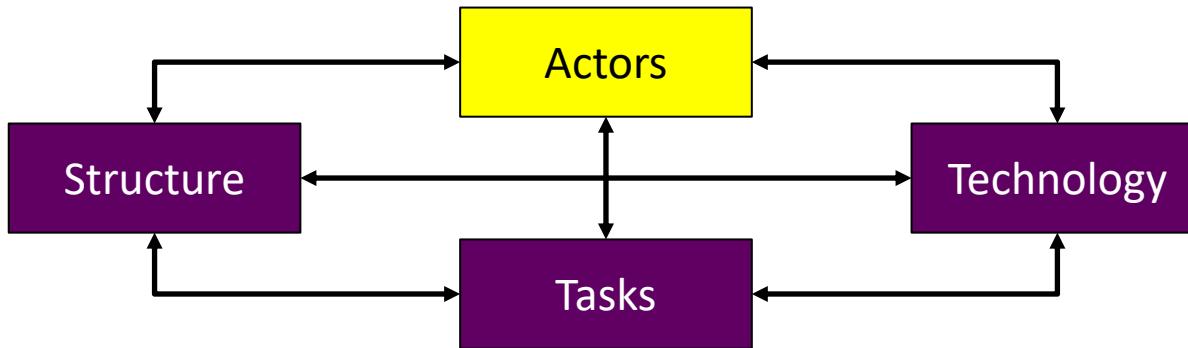
- Introduction
- **Categories of risk**
- Dealing with risk in software projects
- Evaluating risks to the schedule using PERT

Categories of Risk



Based on a socio-technical model of system development
(Lyytinen, Mathiassen & Ropponen, 1998)

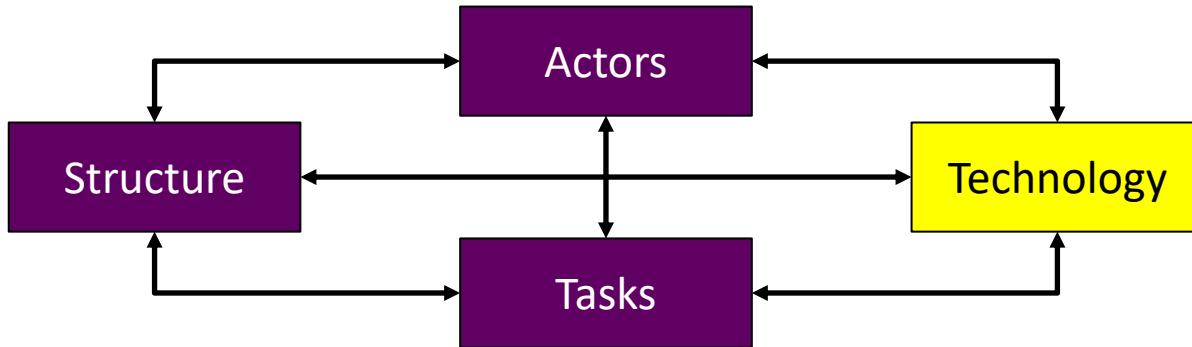
Categories of Risk



Based on a socio-technical model of system development
(Lyytinen, Mathiassen & Ropponen, 1998)

- **Actors** include **people**, all those involved in the project - developers, users, managers, etc.
- For example, high turnover/staff leaving leads to loss of information that is important to the project

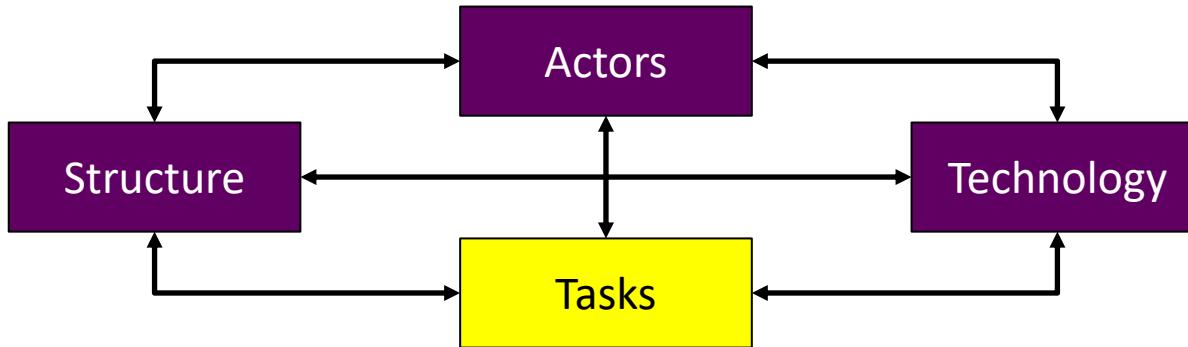
Categories of Risk



Based on a socio-technical model of system development
(Lyytinen, Mathiassen & Ropponen, 1998)

- **Technology** relates to development **tools and techniques** used to implement the project and to technology embedded in the project deliverables

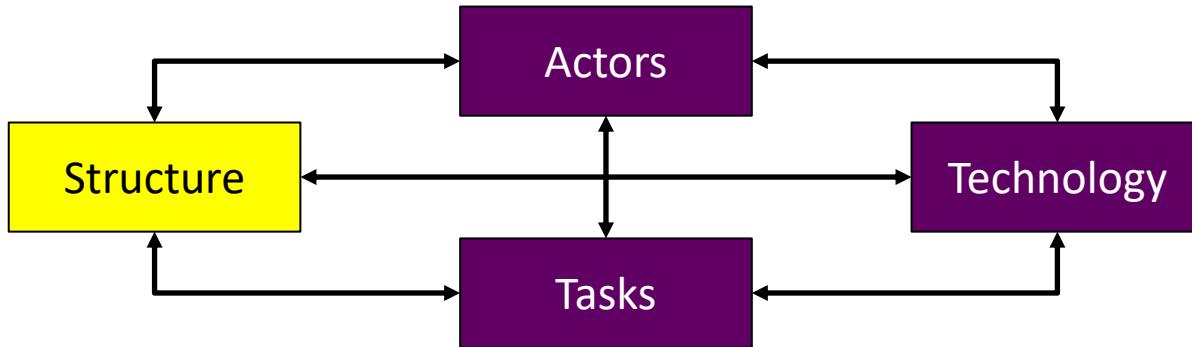
Categories of Risk



Based on a socio-technical model of system development
(Lyytinen, Mathiassen & Ropponen, 1998)

- **Tasks** represent **the work** to be carried out
- A typical risk is that the amount of effort needed to carry out the task is underestimated, so work is not completed by the deadline

Categories of Risk



Based on a socio-technical model of system development
(Lyytinen, Mathiassen & Ropponen, 1998)

- **Structure** covers **management** procedures
- For example, a group assigned a project task is not informed of the assignment because it is not part of the project communication network, so task completion is delayed

Outline

- Introduction
- Categories of risk
- **Dealing with risk in software projects**
- Evaluating risks to the schedule using PERT

The planning for risk includes four steps:

1. **Risk identification** – what risks might we find?
2. **Risk analysis and prioritisation** – which are the most serious risks?
 - Quantitative risk analysis
 - Qualitative risk analysis
3. **Risk planning** – what are we going to do about them?
4. **Risk monitoring** – what is the current state of the risk?

Risk Identification

The most serious effects of risk are:

- Failure to keep within the **cost** estimate
- Failure to achieve the required completion **date (schedule)**
- Failure to achieve the required **quality** or operational requirements

Boehm's Top Ten Risks (1/2)

| | Risk Item | Risk Management Technique |
|---|--|---|
| 1 | Personnel shortfalls | Staffing with top talent, job matching, team building, key personnel agreements, cross training. |
| 2 | Unrealistic schedules and budgets | Detailed multisource cost and schedule estimation, design to cost, incremental development, software reuse, requirements scrubbing. |
| 3 | Developing the wrong functions & properties | Organisation analysis, mission analysis, operations-concept formulation, user surveys and user participation, prototyping, early users' manuals, off-nominal performance analysis, quality-factor analysis. |
| 4 | Developing the wrong user interface | Prototyping, scenarios, task analysis, user participation. |
| 5 | Gold-plating (superfluous features) | Requirements scrubbing, prototyping, cost-benefit analysis, designing to cost. |

Boehm's Top Ten Risks (2/2)

| Risk Item | Risk Management Technique |
|--|--|
| 6 Continuing stream of requirements changes | High change threshold, incremental development (deferring changes to later increments). |
| 7 Shortfalls in externally-furnished components | Benchmarking, inspections, reference checking, compatibility analysis. |
| 8 Shortfalls in externally-performed tasks | Reference checking, pre-award audits, award-fee contracts, competitive design or prototyping , team-building. |
| 9 Real-time performance shortfalls | Simulation, benchmarking, modelling, prototyping , instrumentation, tuning. |
| 10 Straining computer science capabilities | Technical analysis, cost-benefit analysis, prototyping , reference checking. |

Quantitative Risk Analysis

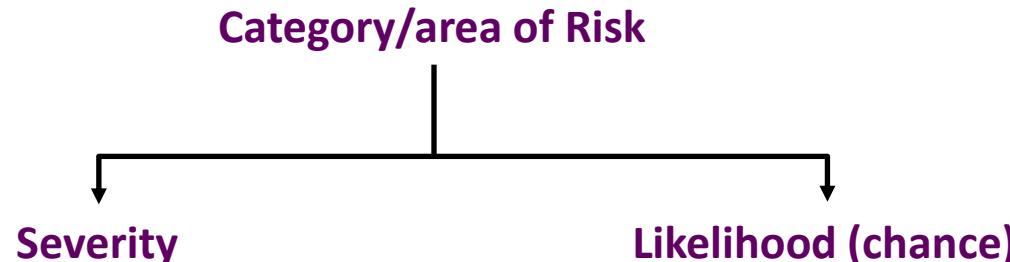
- Numeric estimate of the overall risk on project objectives
- Used for projects:
 - that require a contingency reserve for schedule and budget
 - that are large and complex, involving go/no-go decisions
 - for which senior management wants more detail
- Techniques
 - Three-point estimate
 - Expected Monetary Value (EMV) or Risk Exposure
 - Program Evaluation and Review Technique (PERT)

Quantitative Risk Analysis

- A key quantitative indicator in risk analysis and prioritisation is **probability**
 - Lowest probability is 0 (absolutely no chance - 0%)
 - Highest probability is 1 (absolutely certain - 100%)
 - In practice, risks tend to be one extreme or the other very rarely
- Another quantitative indicator is the **cost** of the risk occurring:
 - Money
 - Time
 - A quality metric, such as defects per 1,000 lines of code, increasing
- The **threat** can be readily quantified by multiplying these
 - Probability x impact

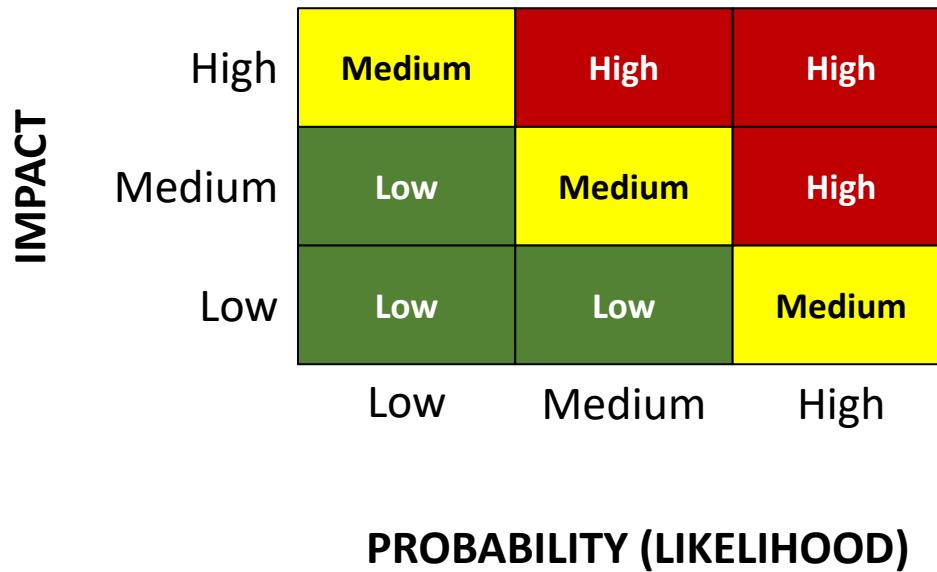
Qualitative Risk Analysis

- Prioritisation of risks uses a pre-defined rating scale, which considers **likelihood** and **impact**
 - **More subjective evaluation of both probability and impact**
 - Quick and easy to perform
 - No special software or tools are required

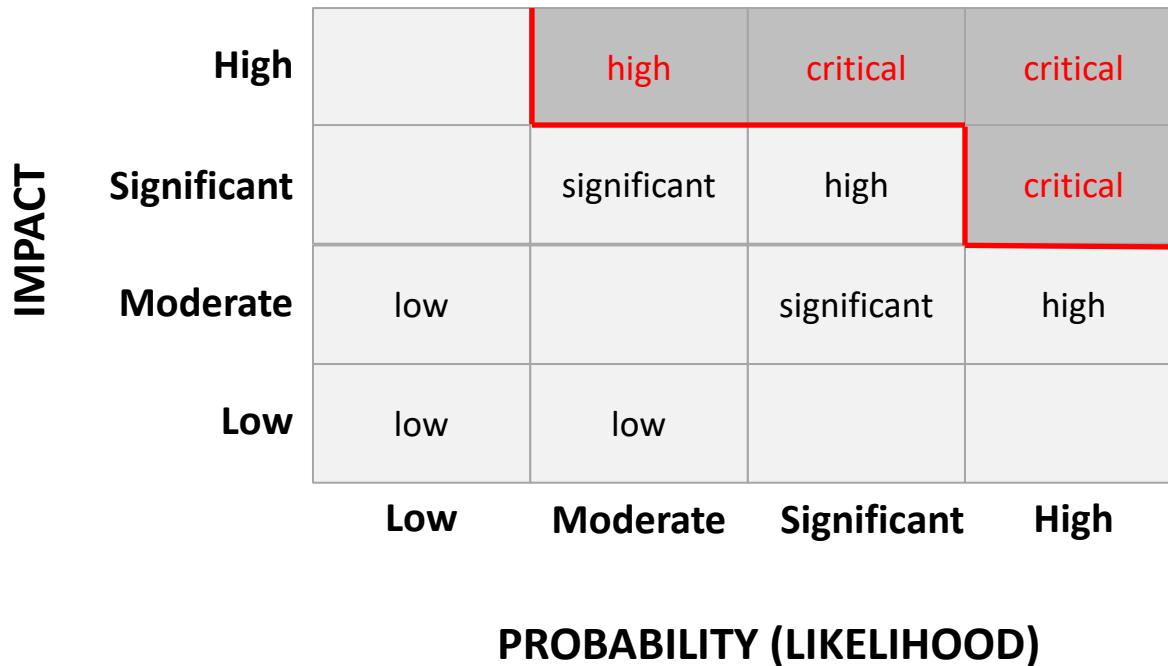


Probability-impact matrix

Risk matrix



Probability-impact matrix



- Risk exposure cannot be calculated
- An area (in the top-right) is selected as containing tasks that require risk planning

How to deal with risks (5 ways)

- **Risk acceptance** – when the cost of avoiding the risk is (estimated to be) greater than the actual cost of the damage that might be inflicted
- **Risk avoidance** – **avoid the cause** associated with the risk
 - e.g., buying an off-the-shelf application **avoids** the risk associated with developing the application (such as poor estimates of effort)
- **Risk reduction** – actions are taken to **reduce the likelihood** of the risk
 - e.g., **prototyping** ought to reduce the risk of incorrect requirements; or offer generous **bonuses on project completion** to reduce likelihood that staff may leave during project

How to deal with risks (5 ways)

- **Risk transfer** – the risk is transferred to another person or organisation
 - e.g., the risk of incorrect development estimates can be transferred by negotiating a fixed price contract with an external software supplier
- **Risk mitigation/contingency measures** – actions are taken to reduce the impact if the risk does occur
 - need to monitor progress of project activities to identify occurrence of causes of risk
 - e.g., backups can be taken to allow rapid recovery in the case of data corruption

Cost-effectiveness of risk actions

- Addressing a risk usually has a **cost** associated with it, and the cost-effectiveness can be measured:

$$\text{leverage} = (\text{RE}_{\text{before}} - \text{RE}_{\text{after}}) / \text{cost}$$

- leverage: how much have we reduced risk (should be > 1.00)**
 - $\text{RE}_{\text{before}}$: the risk exposure cost before the intervention
 - RE_{after} : the risk exposure cost after the intervention
 - cost: usually money, but not necessarily

Cost-effectiveness of risk actions

Example:

- 1% chance of a fire causing £200,000 damage
- Fire alarm costing £500 reduces probability of fire to 0.5%
- Cost-effectiveness of this risk reduction action?
 - leverage = $(RE_{\text{before}} - RE_{\text{after}}) / \text{cost}$

Cost-effectiveness of risk actions

Example (answer):

- 1% chance of a fire causing £200,000 damage
 - $RE_{\text{before}} = 0.01 * 200,000 = 2,000$
- Fire alarm costing £500 reduces probability of fire to 0.5%
 - $RE_{\text{after}} = 0.005 * 200,000 = 1,000$
 - cost = £500
- Cost-effectiveness:
 - leverage = $(RE_{\text{before}} - RE_{\text{after}}) / \text{cost}$
 - leverage = $(2,000 - 1,000) / 500 = \underline{\underline{2.00}}$ Buy the fire alarm!

Risk Management Plan

| Risk Description | Likelihood (low/med/high) | Impact (low/med/high) | Severity | Owner | Mitigation strategies |
|--------------------------------|------------------------------|--------------------------|----------|-----------------|-----------------------------|
| Poorly-defined project purpose | Medium | High | High | Project manager | Complete a business case... |
| | | | | | |
| | | | | | |
| | | | | | |

Outline

- Introduction
- Categories of risk
- Dealing with risk in software projects
- **Evaluating risks to the schedule using PERT**

- Program Evaluation and Review Technique
- Used to determine the **expected time for project activities** based on a combination of **optimistic**, **pessimistic** and **expected durations**
- Can be used to calculate the **probability of project overrun**
- Requires an understanding of critical path analysis

What do we need?

- For the whole project:
 - A list of activities
 - Predecessor/dependency relationships
 - Critical path
- For each activity (three-point estimate):
 - Likely duration
 - Optimistic duration
 - Pessimistic duration

What do we need?

| Task | Description | Predecessors | Duration Estimates | | |
|------|----------------|--------------|--------------------|-------------|--------|
| | | | Optimistic | Pessimistic | Likely |
| A | Requirements | N/A | 6 | 12 | 9 |
| B | Software | A | 5 | 12 | 8 |
| C | Hardware | A | 7 | 13 | 9 |
| D | Training | A | 12 | 18 | 14 |
| E | Implementation | B, C | 3 | 7 | 5 |
| F | Testing | E | 4 | 9 | 5 |

Optimistic time (the best-case scenario, where everything goes as planned)

Likely time (the most realistic time, based on experience or best estimate)

Pessimistic time (the worst-case scenario, where things go wrong and delays occur)

What do we calculate?

| Task | Description | Predecessors | Duration Estimates | | | Expected Duration | Variance |
|------|----------------|--------------|--------------------|-------------|--------|-------------------|----------|
| | | | Optimistic | Pessimistic | Likely | | |
| A | Requirements | N/A | 6 | 12 | 9 | 9.00 | 1.00 |
| B | Software | A | 5 | 12 | 8 | 8.17 | 1.36 |
| C | Hardware | A | 7 | 13 | 9 | 9.33 | 1.00 |
| D | Training | A | 12 | 18 | 14 | 14.33 | 1.00 |
| E | Implementation | B, C | 3 | 7 | 5 | 5.00 | 0.44 |
| F | Testing | E | 4 | 9 | 5 | 5.50 | 0.69 |

(For each task, new columns...)

- We calculate the mean of (optimistic), (pessimistic) and (4 * likely) to get the expected duration. e.g. for A, $(6 + 12 + 9 + 9 + 9 + 9) / 6 = 9.00$

$$\text{Expected duration} = \frac{\text{(optimistic)} + \text{(pessimistic)} + (4 * \text{likely})}{6}$$

Expected duration =

What do we calculate?

| Task | Description | Predecessors | Duration Estimates | | | Expected Duration | Variance |
|------|----------------|--------------|--------------------|-------------|--------|-------------------|----------|
| | | | Optimistic | Pessimistic | Likely | | |
| A | Requirements | N/A | 6 | 12 | 9 | 9.00 | 1.00 |
| B | Software | A | 5 | 12 | 8 | 8.17 | 1.36 |
| C | Hardware | A | 7 | 13 | 9 | 9.33 | 1.00 |
| D | Training | A | 12 | 18 | 14 | 14.33 | 1.00 |
| E | Implementation | B, C | 3 | 7 | 5 | 5.00 | 0.44 |
| F | Testing | E | 4 | 9 | 5 | 5.50 | 0.69 |

(For each task, new columns...)

1. We calculate the mean of (optimistic), (pessimistic) and (4 * likely) to get the **expected duration**. e.g. for A, $(6 + 12 + 9 + 9 + 9 + 9) / 6 = 9.00$
2. For the **variance** (a measure of how much these values vary), we calculate $((\text{pessimistic} - \text{optimistic}) / 6)^2$
e.g. for A, $((12-6) / 6)^2 = 1.00$

What do we calculate?

| Task | Description | Predecessors | Duration Estimates | | | Expected Duration | Variance |
|------|----------------|--------------|--------------------|-------------|--------|-------------------|----------|
| | | | Optimistic | Pessimistic | Likely | | |
| A | Requirements | N/A | 6 | 12 | 9 | 9.00 | 1.00 |
| B | Software | A | 5 | 12 | 8 | 8.17 | 1.36 |
| C | Hardware | A | 7 | 13 | 9 | 9.33 | 1.00 |
| D | Training | A | 12 | 18 | 14 | 14.33 | 1.00 |
| E | Implementation | B, C | 3 | 7 | 5 | 5.00 | 0.44 |
| F | Testing | E | 4 | 9 | 5 | 5.50 | 0.69 |

Sum CP: 28.33 3.14

- We need to know the critical path, which can be determined using an activity network
- For the sake of simplicity, the critical path is highlighted here in yellow
- Only these items feed into subsequent calculations
 - The sum of expected duration CP and the sum of variances CP)

What do we calculate?

| 1 | 2 | 3 | 4 | 5 |
|----------------|----------------|-------------|---------|-----------------------|
| Total Duration | Total Variance | Target Date | Z-Score | Probability of Target |
| 28.83 | 3.14 | 28 | -0.470 | 31.90% |
| 28.83 | 3.14 | 29 | 0.094 | 53.75% |
| 28.83 | 3.14 | 30 | 0.659 | 74.49% |
| 28.83 | 3.14 | 31 | 1.223 | 88.93% |
| 28.83 | 3.14 | 32 | 1.787 | 96.31% |
| 28.83 | 3.14 | 33 | 2.352 | 99.07% |

1. Sum of all expected durations on the critical path
2. Sum of all variance values on the critical path

All rows now will take these same values into account

What do we calculate?

| 1 | 2 | 3 | 4 | 5 |
|----------------|----------------|-------------|---------|-----------------------|
| Total Duration | Total Variance | Target Date | Z-Score | Probability of Target |
| 28.83 | 3.14 | 28 | -0.470 | 31.90% |
| 28.83 | 3.14 | 29 | 0.094 | 53.75% |
| 28.83 | 3.14 | 30 | 0.659 | 74.49% |
| 28.83 | 3.14 | 31 | 1.223 | 88.93% |
| 28.83 | 3.14 | 32 | 1.787 | 96.31% |
| 28.83 | 3.14 | 33 | 2.352 | 99.07% |

1. Sum of all expected durations on the critical path
2. Sum of all variance values on the critical path
3. The number of days we've told our client this will take (target date)

What do we calculate?

| 1 | 2 | 3 | 4 | 5 |
|----------------|----------------|-------------|---------|-----------------------|
| Total Duration | Total Variance | Target Date | Z-Score | Probability of Target |
| 28.83 | 3.14 | 28 | -0.470 | 31.90% |
| 28.83 | 3.14 | 29 | 0.094 | 53.75% |
| 28.83 | 3.14 | 30 | 0.659 | 74.49% |
| 28.83 | 3.14 | 31 | 1.223 | 88.93% |
| 28.83 | 3.14 | 32 | 1.787 | 96.31% |
| 28.83 | 3.14 | 33 | 2.352 | 99.07% |

1. Sum of all expected durations on the critical path
2. Sum of all variance values on the critical path
3. The number of days we've told our client this will take (target date)
4. Z-score: a measure of the difference between columns 1 and 3:
 - $(\text{Column 3} - \text{Column 1}) / \text{Square Root}(\text{Column 2})$
 - $(\text{Target Date} - \text{Total Duration}) / (\text{Standard Deviation})$

What do we calculate?

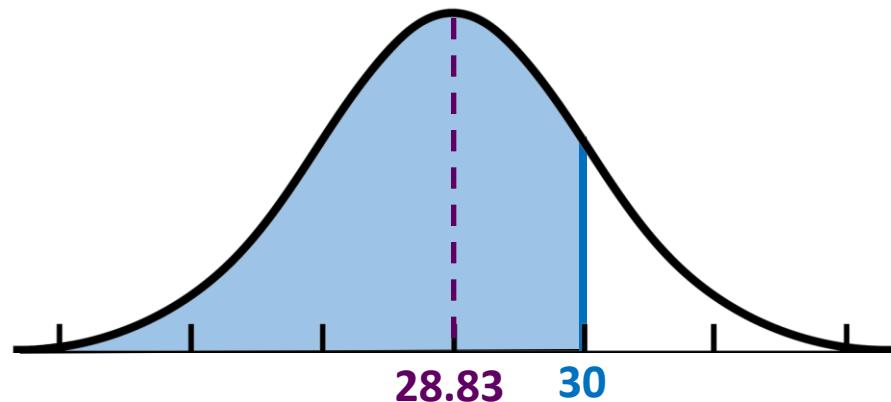
| 1 | 2 | 3 | 4 | 5 |
|----------------|----------------|-------------|---------|-----------------------|
| Total Duration | Total Variance | Target Date | Z-Score | Probability of Target |
| 28.83 | 3.14 | 28 | -0.470 | 31.90% |
| 28.83 | 3.14 | 29 | 0.094 | 53.75% |
| 28.83 | 3.14 | 30 | 0.659 | 74.49% |
| 28.83 | 3.14 | 31 | 1.223 | 88.93% |
| 28.83 | 3.14 | 32 | 1.787 | 96.31% |
| 28.83 | 3.14 | 33 | 2.352 | 99.07% |

1. Sum of all expected durations on the critical path
2. Sum of all variance values on the critical path
3. The number of days we've told our client this will take (target date)
4. Z-score: a measure of the difference between columns 1 and 3:
 - $(\text{Column 3} - \text{Column 1}) / \text{Square Root}(\text{Column 2})$
 - $(\text{Target Date} - \text{Total Duration}) / (\text{Standard Deviation})$
5. The probability that we will meet the target date (using Z-Score)

Normal Distribution

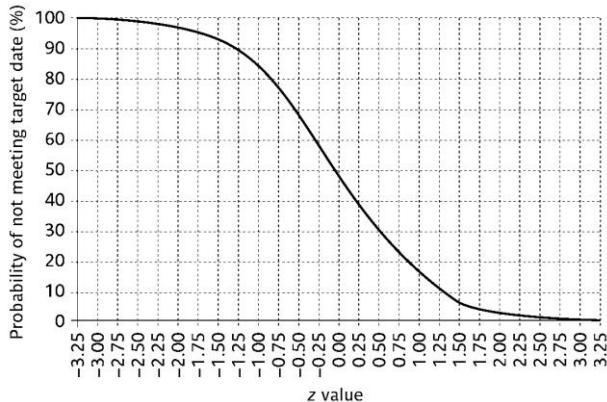
| Total Duration | Total Variance | Target Date | Z-Score | Probability of Target |
|----------------|----------------|-------------|---------|-----------------------|
| 28.83 | 3.14 | 28 | -0.470 | 31.90% |
| 28.83 | 3.14 | 29 | 0.094 | 53.75% |
| 28.83 | 3.14 | 30 | 0.659 | 74.49% |
| 28.83 | 3.14 | 31 | 1.223 | 88.93% |
| 28.83 | 3.14 | 32 | 1.787 | 96.31% |
| 28.83 | 3.14 | 33 | 2.352 | 99.07% |

- These calculations assume a normal distribution of potential duration; the likeliest duration is the interval in which sits 28.83 (e.g., weeks), and “longer is as likely as shorter”



What do we calculate?

| Total Duration | Total Variance | Target Date | Z-Score | Probability of Target |
|----------------|----------------|-------------|---------|-----------------------|
| 28.83 | 3.14 | 28 | -0.470 | 31.90% |
| 28.83 | 3.14 | 29 | 0.094 | 53.75% |
| 28.83 | 3.14 | 30 | 0.659 | 74.49% |
| 28.83 | 3.14 | 31 | 1.223 | 88.93% |
| 28.83 | 3.14 | 32 | 1.787 | 96.31% |
| 28.83 | 3.14 | 33 | 2.352 | 99.07% |



Z-table

0.659 →
Prob. (%)
meeting
target

| <i>z</i> | .00 | .01 | .02 | .03 | .04 | .05 | .06 | .07 | .08 | .09 |
|----------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|
| 0.0 | .5000 | .5040 | .5080 | .5120 | .5160 | .5199 | .5239 | .5279 | .5319 | .5359 |
| 0.1 | .5398 | .5438 | .5478 | .5517 | .5557 | .5596 | .5636 | .5675 | .5714 | .5753 |
| 0.2 | .5793 | .5832 | .5871 | .5910 | .5948 | .5987 | .6026 | .6064 | .6103 | .6141 |
| 0.3 | .6179 | .6217 | .6255 | .6293 | .6331 | .6368 | .6406 | .6443 | .6480 | .6517 |
| 0.4 | .6554 | .6591 | .6628 | .6664 | .6700 | .6736 | .6772 | .6808 | .6844 | .6879 |
| 0.5 | .6915 | .6950 | .6985 | .7019 | .7054 | .7088 | .7123 | .7157 | .7190 | .7224 |
| 0.6 | .7257 | .7291 | .7324 | .7357 | .7389 | .7422 | .7454 | .7486 | .7517 | .7549 |
| 0.7 | .7580 | .7611 | .7642 | .7673 | .7704 | .7734 | .7764 | .7794 | .7823 | .7852 |
| 0.8 | .7881 | .7910 | .7939 | .7967 | .7995 | .8023 | .8051 | .8078 | .8106 | .8133 |
| 0.9 | .8159 | .8186 | .8212 | .8238 | .8264 | .8289 | .8315 | .8340 | .8365 | .8389 |
| 1.0 | .8413 | .8438 | .8461 | .8485 | .8508 | .8531 | .8554 | .8577 | .8599 | .8621 |
| 1.1 | .8643 | .8665 | .8686 | .8708 | .8729 | .8749 | .8770 | .8790 | .8810 | .8830 |
| 1.2 | .8849 | .8869 | .8888 | .8907 | .8925 | .8944 | .8962 | .8980 | .8997 | .9015 |
| 1.3 | .9032 | .9049 | .9066 | .9082 | .9099 | .9115 | .9131 | .9147 | .9162 | .9177 |
| 1.4 | .9192 | .9207 | .9222 | .9236 | .9251 | .9265 | .9279 | .9292 | .9306 | .9319 |
| 1.5 | .9332 | .9345 | .9357 | .9370 | .9382 | .9394 | .9406 | .9418 | .9429 | .9441 |
| 1.6 | .9452 | .9463 | .9474 | .9484 | .9495 | .9505 | .9515 | .9525 | .9535 | .9545 |
| 1.7 | .9554 | .9564 | .9573 | .9582 | .9591 | .9599 | .9608 | .9616 | .9625 | .9633 |
| 1.8 | .9641 | .9649 | .9656 | .9664 | .9671 | .9678 | .9686 | .9693 | .9699 | .9706 |
| 1.9 | .9713 | .9719 | .9726 | .9732 | .9738 | .9744 | .9750 | .9756 | .9761 | .9767 |
| 2.0 | .9772 | .9778 | .9783 | .9788 | .9793 | .9798 | .9803 | .9808 | .9812 | |
| 2.1 | .9821 | .9826 | .9830 | .9834 | .9838 | .9842 | .9846 | .9850 | .9854 | .9857 |
| 2.2 | .9861 | .9864 | .9868 | .9871 | .9875 | .9878 | .9881 | .9884 | .9887 | .9890 |
| 2.3 | .9893 | .9896 | .9898 | .9901 | .9904 | .9906 | .9909 | .9911 | .9913 | .9916 |

Summary of Key Points

- Risks represent potential future **problems** or **opportunities**, and link causes to their effects
- Risks in software projects are related to **actors**, **technology**, **structure**, **tasks** (or a combination thereof)
- Identified risks needs to be analysed in terms of **impact** and **likelihood**
- High-**impact** and/or high-**likelihood** risks may need to be addressed by eliminating or reducing their likelihood or impact
- Risks to the schedule can be analysed using PERT

References (further reading)

- Boehm, B. (1989) 'Software risk management', *European Software Engineering Conference, Berlin*, vol. 387. https://doi.org/10.1007/3-540-51635-2_29
- Hughes, B. & Cotterell, M., (2009) *Software Project Management*, 5th ed., London: McGraw-Hill. (Chapter on Risks - pp.162-191)
- Lyytinen, K., Mathiassen, L. & Ropponen, J. (1998) 'Attention Shaping and Software Risk – A Categorical Analysis of Four Classic Risk Management Approaches', *Information Systems Research*, 9(3), pp.233-255.
<https://pubsonline.informs.org/doi/pdf/10.1287/isre.9.3.233>



Aston University

BIRMINGHAM UK

Software Project Management

Unit 5: Risk Management

Thais Webber
Richard Lee





Aston University

BIRMINGHAM UK

Software Project Management

Unit 6: Project Monitoring & Control
(plus) Quality Management

Thais Webber
Richard Lee



Unit Objectives

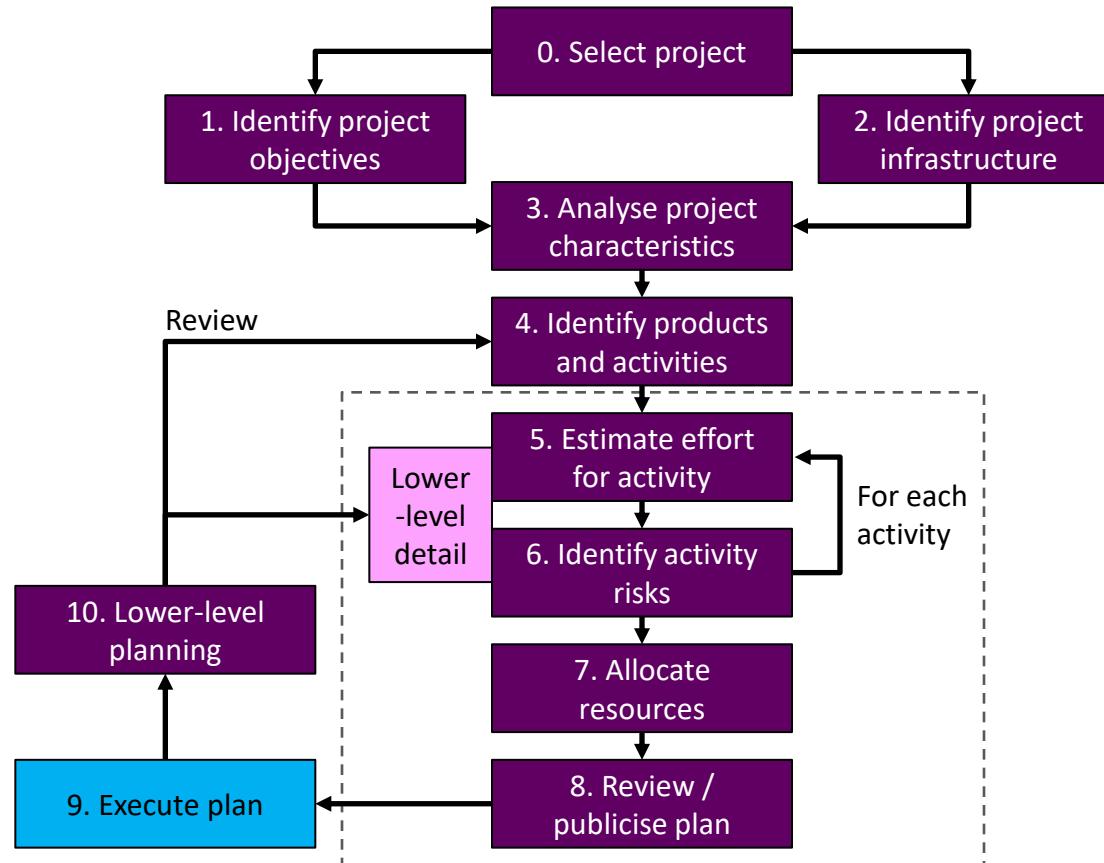
When you have completed this unit, you will be able to:

- Monitor project progress and assess the risk of slippage
- Visualise and assess the state of the project, revising targets as necessary to address drift
- Control changes to project requirements
- Quality management importance and standards in software projects

Outline

- Introduction
- Monitoring project progress
- Monitoring cost
- Getting the project back on target
- Quality management importance and standards

Step Wise



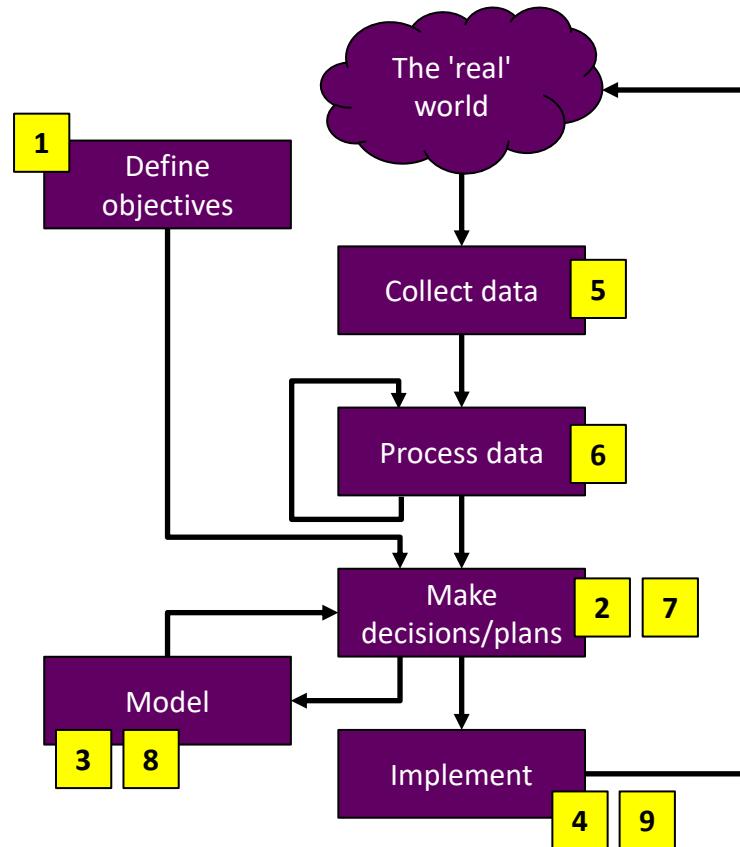
Motivations

- To ensure that the project progresses according to the plan
 - monitor what is happening and compare actual achievement against the schedule
 - may need to revise schedule to bring the project back on target as soon as possible

Have you done this during the progress of your Final Year Project?

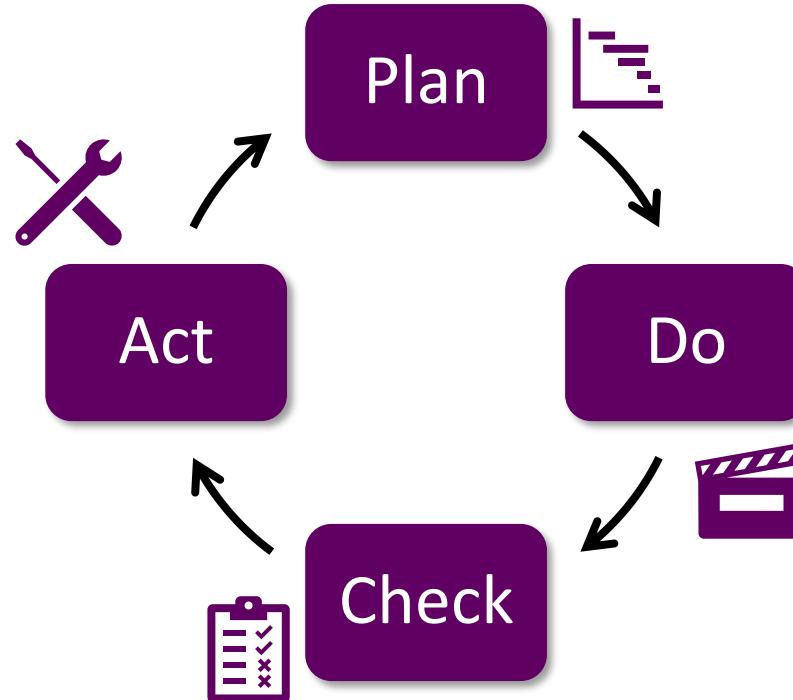
- To keep track of changes imposed from outside (e.g., changes in requirements)

A project control cycle



An alternative view

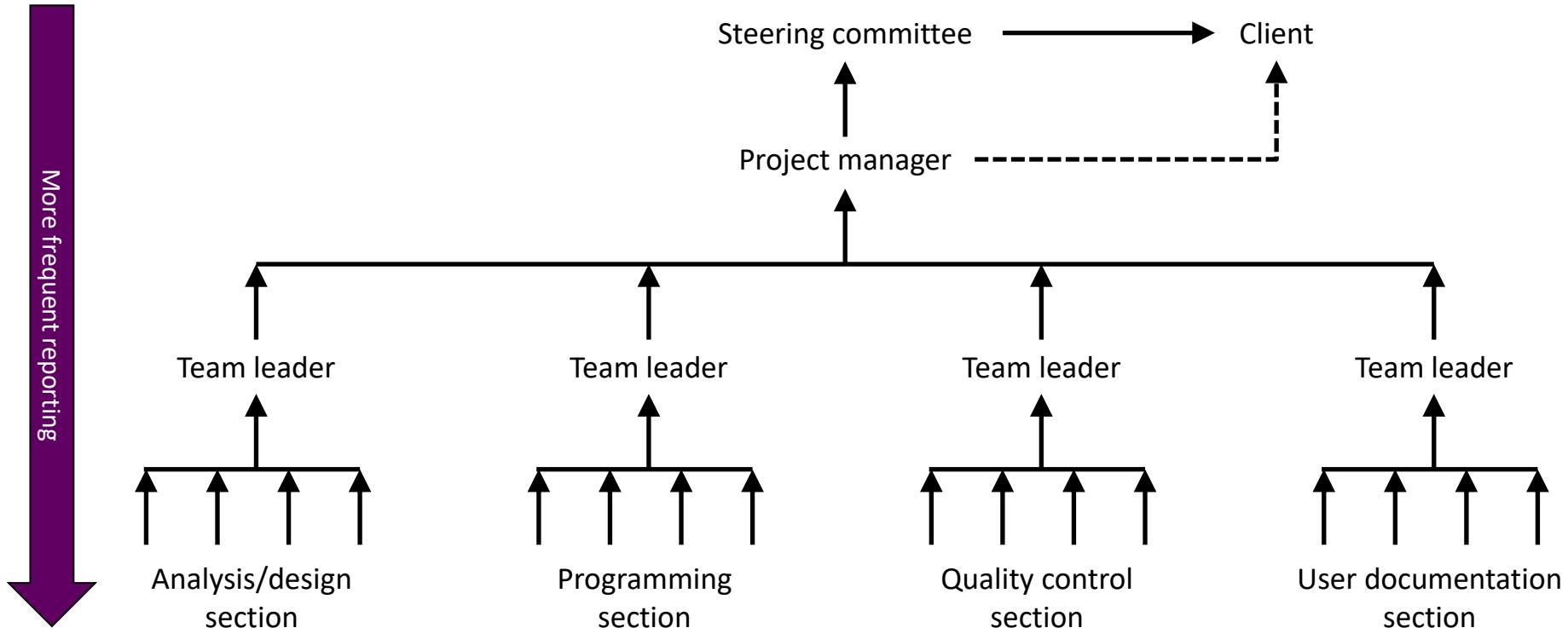
PDCA cycle



Responsibilities

- The overall responsibility for ensuring satisfactory progress on a project belongs to the Project Board
 - alternative names include project steering committee and project management board
- Day-to-day responsibility is the role of the Project Manager
 - aspects of this can be delegated to team leaders
- A reporting hierarchy that help teams stay aligned

Reporting frequency across the hierarchy



An alternative view (Scrum)

- Beyond hierarchies
- Focus on collaboration

An alternative view (Scrum)



Product Owner

value-based decisions



Scrum Master

supports the team
embracing principles



Development Team

works collaboratively
as one unit

Outline

- Introduction
- Monitoring project progress
- Monitoring cost
- Getting the project back on target
- Quality management importance and standards

Assessing progress

- A **checkpoint** is a point in the project where progress is checked
- **Two types of checkpoint:**
 - **Event-driven:** check takes place when a particular event has been achieved (e.g., at the end of project stages in PRINCE2)
 - **Time-driven:** date of the check is pre-determined (e.g., between sprints in Scrum)
- Checkpoints ensure that intermediate products are compatible. They support project monitoring.
- FYP checkpoints: e.g., the proposal form and the first report submission

Collecting progress details: challenges

- Dealing with partial completions is a problem
- Estimates are affected by the **99% completion syndrome**
 - developer reports that the task is 25%, 50% and 75% complete at the end of weeks 1, 2 and 3, respectively
 - however, at the end of week 4 it is reported that the task is 99% complete
- Possible solutions are based on **objective verification of sub-task completion:**
 - control of products, not activities
 - subdivide into small sub-activities that each generates a product

Red / Amber / Green (RAG) reporting

- **Traffic-light method steps:**

- identify key tasks
- break down into sub-tasks; assess subtasks as:
 - **Red:** not on target and recoverable only with difficulty
 - **Amber:** not on target but recoverable
 - **Green:** on target
- **status of critical tasks is particularly important**

- However...

- Involves estimates based on imprecise data
- Imposes discrete classifications on a potentially more nuanced situation
- Takes time

RAG example

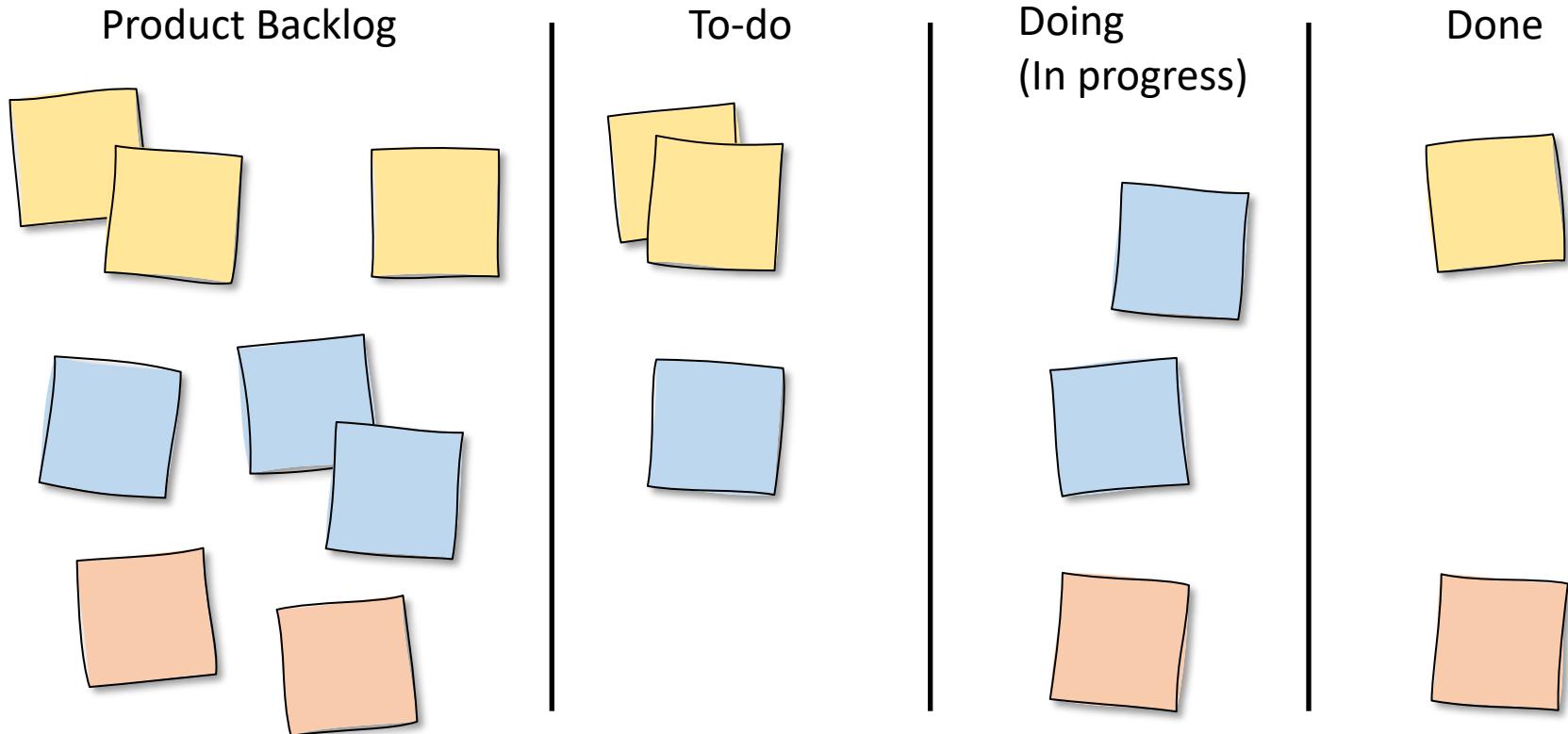
Staff: Digital Rick

Activity: Code & test module C



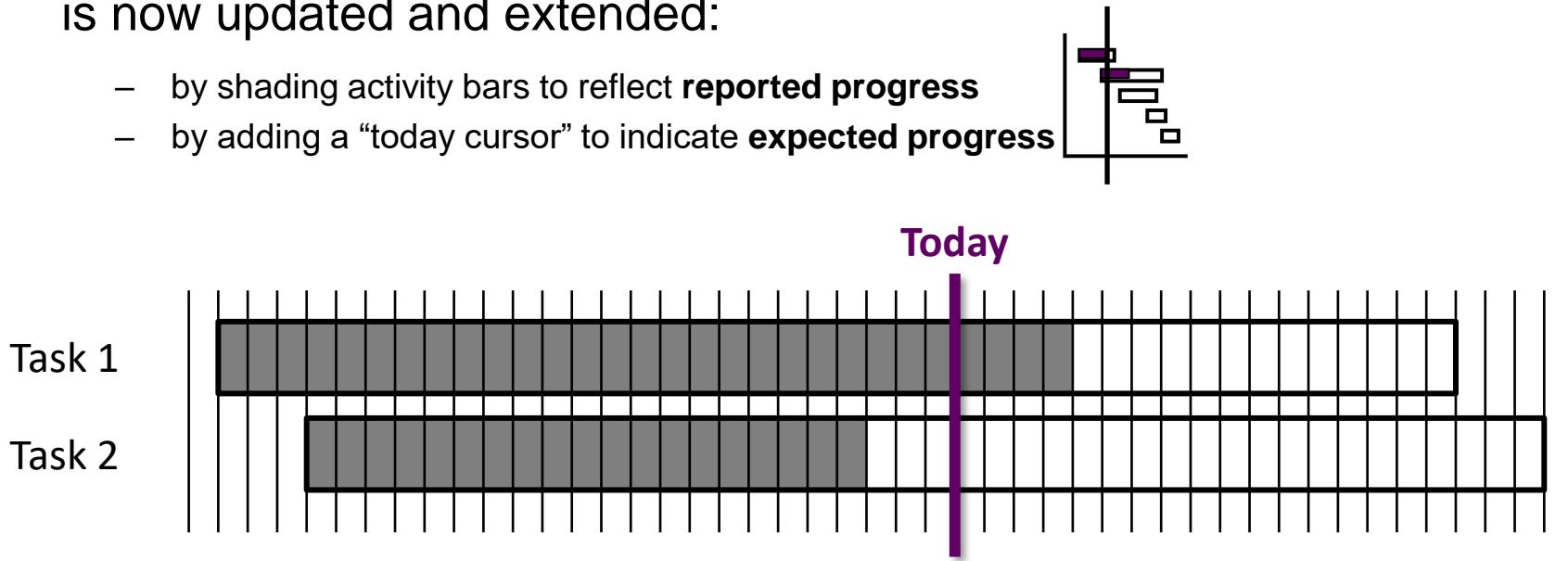
| Week number: | 13 | 14 | 15 | 16 | 17 | 18 | |
|------------------------|----|----|----|----|----|----|--|
| Activity summary | G | A | A | R | | | Major issue in update script |
| Component | | | | | | | Comments |
| Unit tests | G | A | A | G | | | No issues — tests stable. |
| File update procedures | G | G | A | R | | | The file update script failing on large files — needs fix! |
| Refactoring | G | G | G | A | | | Refactoring uncovered hidden bug — fixing now. |
| Compilation | G | G | G | A | | | Build warnings increasing — must review. |
| Documentation | G | G | G | A | | | Draft written, but examples still missing. |

An alternative view (Scrum/Kanban)



Using Gantt charts to track project progress

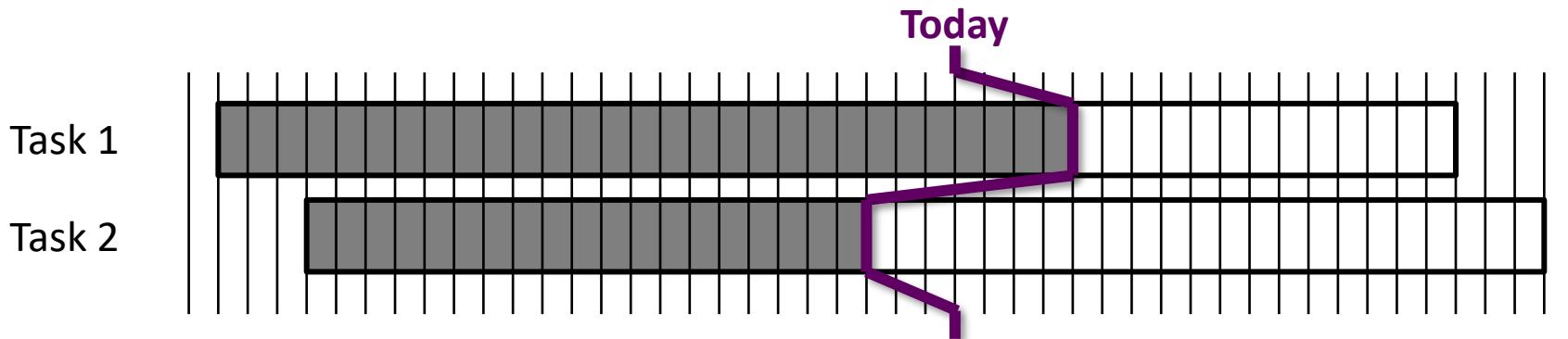
- Gantt chart containing the project schedule from the initial project planning is now updated and extended:
 - by shading activity bars to reflect **reported progress**
 - by adding a “today cursor” to indicate **expected progress**



Task 1 is ahead of schedule; task 2 is behind schedule

Using slip charts to track project progress

- A Gantt chart variant with a **bent 'today' cursor** to indicate activity positions
- The more jagged the line, the more inconsistent the progress
- In this event, resources might be reallocated from task 1 to task 2



Limitations of Gantt / slip charts

- Neither shows clearly the **slippage of the project completion date** through the life of the project
- They do not allow the analysis of the **project trends**:
 - We cannot see how delays are growing or improving
 - We cannot see if things are getting better or worse after we make changes
- This information is essential in deciding what action to take:
 - In case a delay comes from **productivity consistently low** (e.g., each week of work from the original plan takes a week and a half), the final completion date will keep slipping unless appropriate action is taken in the project.
 - In case a delay is a **one-off event**, a small fix — like adding a team member briefly — may be enough.

Outline

- Introduction
- Monitoring project progress
- Monitoring cost
- Getting the project back on target
- Quality importance and standards

Cost monitoring

- Cost monitoring is an important component of project control, but also as an indicator of the effort that has gone into the project:
 - A project could be late because the staff originally committed have not been deployed – in this case the project will be behind time but under budget
 - A project could be on time but only because additional resources have been added and so be over budget
- There is a need for the monitoring of both achievements and costs

Earned Value Approach

- A popular approach to **cost monitoring**
 - Proposed by the US Department of Defence for their contractors
- Each task is assigned a '**value**' based on the original cost estimates
 - Often as financial cost, but could be effort - e.g., (person days)
 - This attribute of a task is termed the task **planned value**
- Tasks are also assigned an **earned value** that is calculated based on the state of the task:
 - A task on which work has not started has an earned value of zero
 - A completed task has an earned value equal to the planned value
 - **The earned value of a project is the sum of the earned values for all its tasks** – a quantity that increases over the lifetime of the project

Partly-completed tasks

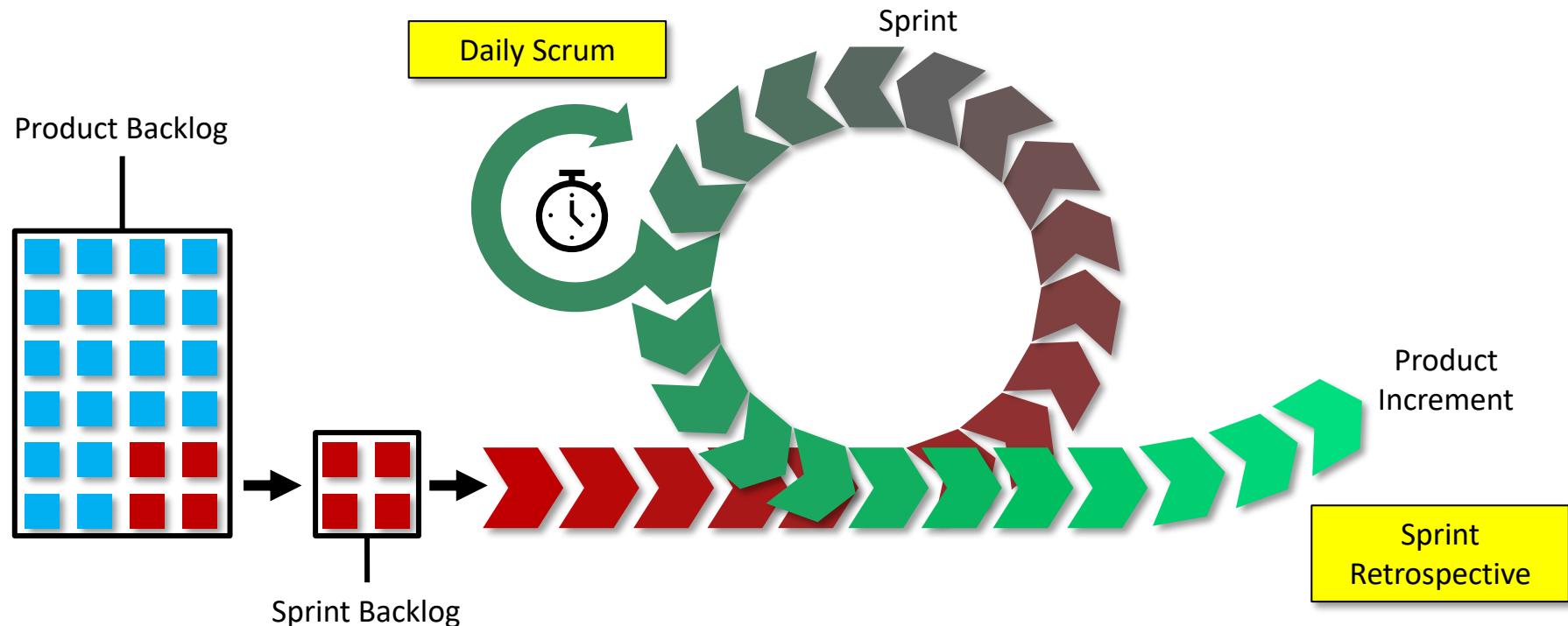
- The **0/100** technique
 - Earned value of task is zero until completion, and 100% of the planned value on completion
- The **50/50** technique
 - Half of the planned value allocated at start, the other half on completion
- The **milestone** technique
 - Earned value of task is the planned value of last milestone achieved
- The **percentage** technique
 - Works only if there is a way of assessing the percentage of the task that was completed (e.g., what percentage of 500 data records have been manually typed into a database)

Exception planning

- Project manager typically allowed to change project plan as long as the agreed project outcomes are produced on time and within budget
- But changes to delivery date, scope and/or cost of project can affect:
 - Users (e.g., reductions in the scope of the project)
 - The business case (e.g. costs increase reducing the potential profits of delivered software product)
- In these cases, an **exception report** is needed (unless using Scrum)
 - Written by the project manager to explain the reasons that justify such a deviation from the existing plan

An alternative view (Scrum)

* Scrum does not use formal exception reports



* issues are handled through sprint reviews, replanning, and backlog adjustments

Prioritising monitoring

- Monitoring takes time and uses resources
 - Should be applied with different levels of detail/effort
- We want to focus more on monitoring certain types of activity:
 - Critical path activities
 - Activities with no free float – if delayed later dependent activities (although not the whole project) are delayed
 - Activities with less than a specified amount of float
 - High-risk activities
 - Activities using critical resources

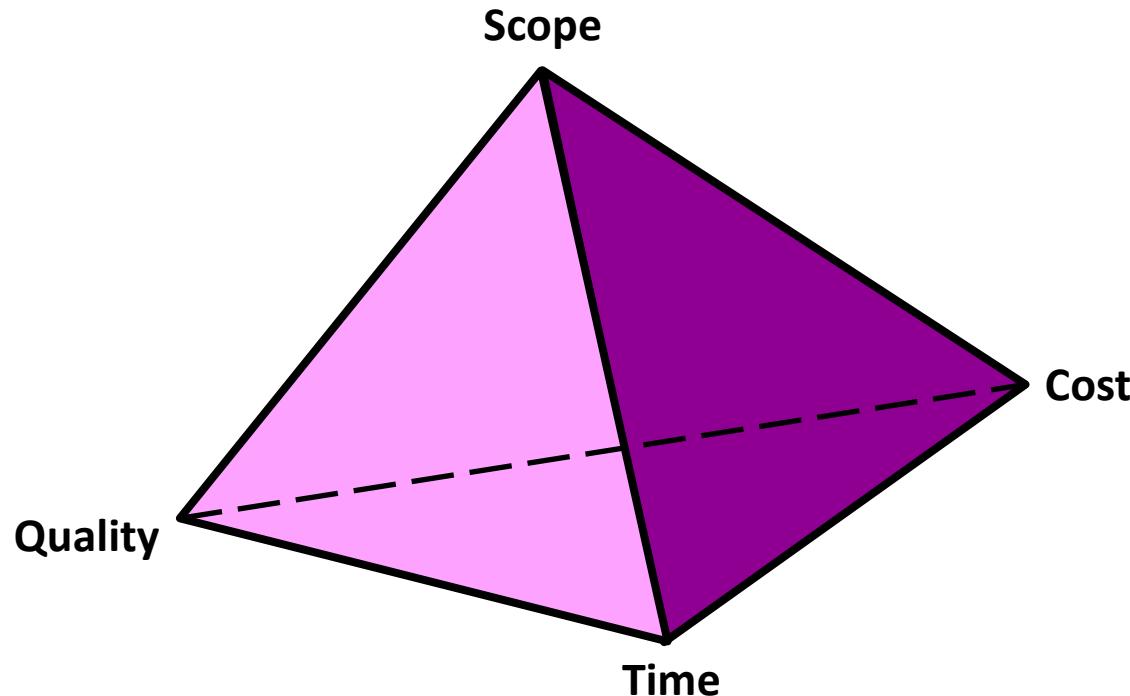
Outline

- Introduction
- Monitoring project progress
- Monitoring cost
- Getting the project back on target
- Quality management importance and standards

Getting back on track: options

- Most projects experience – at one time or another – **delays and unexpected events**
 - Monitoring project progress identifies these events
- **Several options for mitigation are available**
 - Renegotiate the deadline
 - Shorten critical path
 - Reconsider activity dependencies:
 - overlap the activities so that the start of one activity does not have to wait for completion of another – this may have a negative impact on quality
 - split activities

Getting back on track: balancing factors



Summary key points in Monitoring & Control

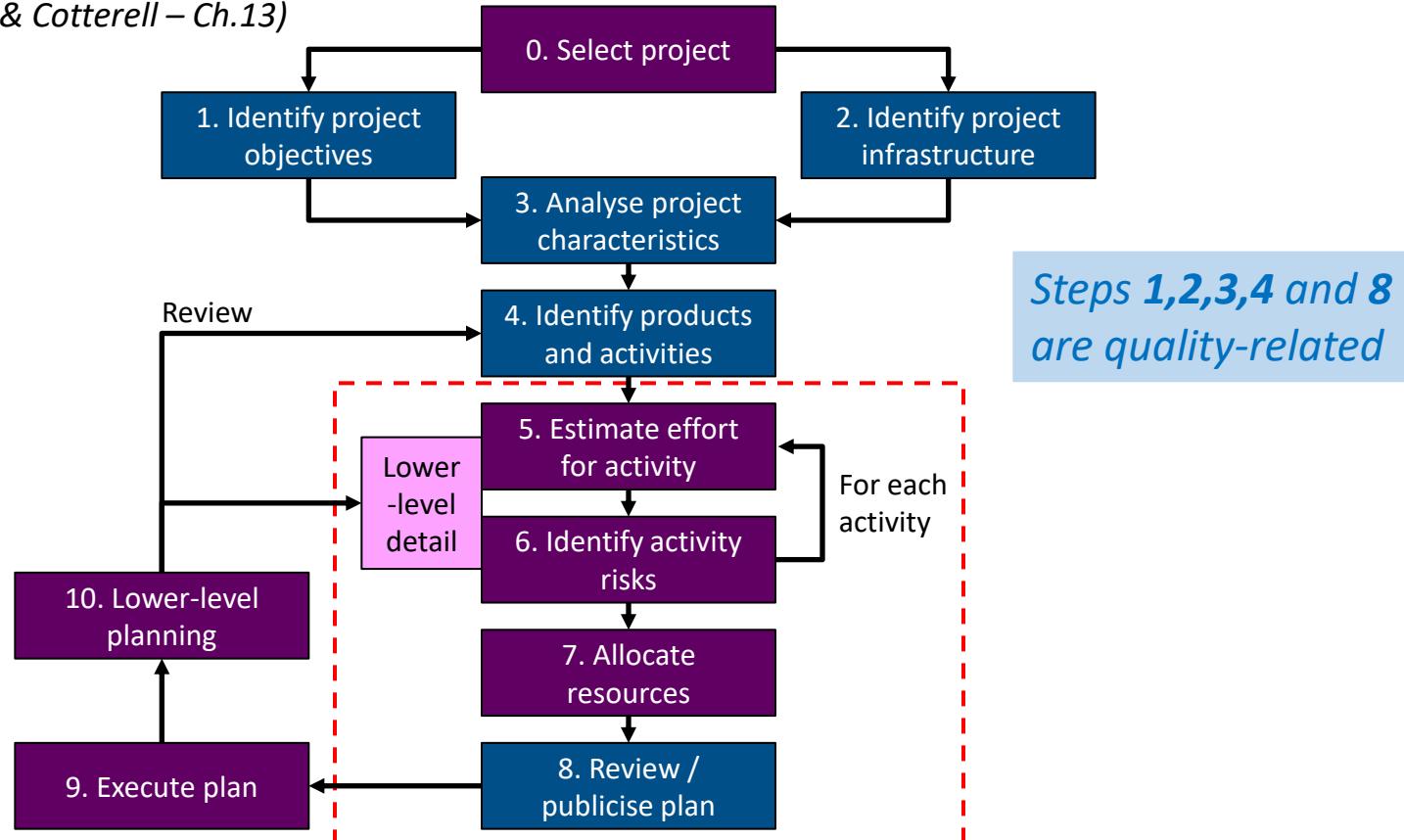
- **Monitoring compares actual progress with the plan** to detect delays early
- **Reporting hierarchy** ensures clear communication from team members
- **Checkpoints** (event-driven and time-driven) help track progress at key moments
- **Percentage completion** can be misleading, especially near the end of tasks
- **RAG reports** give simple status but lack trend detail
- **Agile boards** show task-level flow but not overall project health
- **Gantt and slip charts** show schedule status but not trends over time
- **Cost monitoring** shows the relationship between effort, schedule, and spending
- **Earned value techniques** provide objective measures of progress (PV, EV)
- **Exception planning** escalates changes to time, cost, or scope
- **Monitoring effort should vary** depending on float, risk, and criticality

Outline

- Introduction
- Monitoring project progress
- Monitoring cost
- Getting the project back on target
- Quality management importance and standards

Quality in the ‘Step Wise’ framework

(Book SPM Hughes & Cotterell – Ch.13)



Quality definition and importance

Quality often vaguely defined

Stakeholders have **different views on quality**

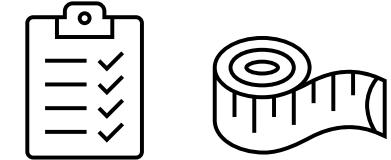
Common fallacy: quality can be defined and improved later after all functionality is built

Software Quality refers to the degree to which a software product:

- meets specified requirements (functional/non-functional)
- satisfies user needs and expectations
- performs reliably under specified conditions

How to define quality in a project

- **Start with requirements (F/NF)** and specifications
- **Use measurable, objective metrics** to track quality
- **Define clear criteria** so you can check whether quality goals are met



Examples of metrics:

- Performance: system response time under 2 seconds, etc.
- Defect limits: fewer than 5 critical defects per release; less than 10 defects per 1,000 lines of code, etc.
- User satisfaction: average score of 4.5/5 in post-task surveys, etc.

PRODUCT vs. PROCESS Quality Management

PRODUCT or
deliverable

Is the product good?

Termed **quality control (QC)** –
evaluation of the final product
against requirements

Ideal but difficult – complex
and costly

PROCESSES
used to build the
product

Did we build it in a good way?

Termed **quality assurance (QA)** –
evaluation of the processes
behind product development

Easier to manage

Can also be applied to third-party
software

Process Quality Management

The purpose is to ensure that the project conforms to the mutually agreed requirements, specifications, and expectations.

- **CMMI** (Capability Maturity Model Integration)
- **Six Sigma**
- **Agile and DevOps Perspectives:**
 - **Agile Quality Management Practices**
 - Agile Testing Quadrants
 - **DevOps Quality Management Practices**
 - Continuous Integration (CI)/ Continuous Delivery (CD)

Maintain consistency and reliability throughout the project lifecycle

Product Quality Management

ISO 9126: Evolution of Software Quality Standards

- **Development of checklists:** Increasingly comprehensive software quality checklists led to the ISO 9126 standard (1991)
- **ISO 9126 updates:** Expanded versions released periodically
- **Separate documents for stakeholders:**
 - **Acquirers:** obtaining software from suppliers
 - **Developers:** building the software
 - **Independent evaluators:** assessing software quality for users

Software Quality and the ISO 9126 Standard

How easy is to transfer the software to another environment?

Portability

Maintainability

How easy is to modify the software?

Are the required functions available in the software?

Functionality

ISO/IEC 9126

Efficiency

How efficient is the software?
(performance)

Six primary quality characteristics

How reliable is the software?

Reliability

Usability

Is the software easy to use?

Evolution into newer standards like ISO/IEC 25010

- It is part of the **ISO/IEC 25000 series**
- Also known as **SQuaRE** - Software Quality Requirements and Evaluation
 - This standard replaces and expands upon the earlier **ISO 9126** model.
- Expanded Quality Model is divided in two key models:
 - **Product Quality Model (8 key characteristics of software product itself)**
 - Added security and compatibility, refined functionality and efficiency
 - **Quality in Use Model (5 characteristics, focuses on user experience)**
 - Focuses on the impact of the software in real-world use

Summary of key points in Quality Management

- **Quality is essential in software**
- **Quality must be built-in from the start**, not added as an afterthought
- **Quality-related requirements** need clear, measurable definitions so we can verify them objectively
- **Quality Control (QC)** checks the **product**, and **Quality Assurance (QA)** improves the **process** — both are needed for success
- **Modern quality management combines frameworks** (CMMI, Six Sigma, Agile, and DevOps) focusing on processes: continuous improvement, testing, and automation
- **Best practices and standards** improve capability but do not guarantee quality



Aston University

BIRMINGHAM UK

Software Project Management

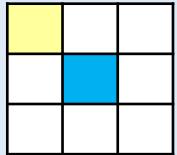
Unit 6: Project Monitoring & Control
(plus) Quality Management

Thais Webber
Richard Lee

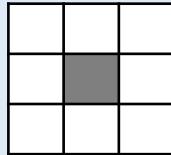


Conway's Game of Life

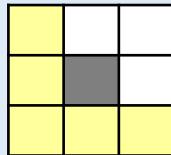
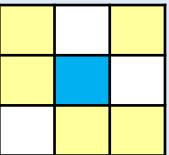
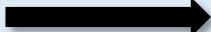
Before



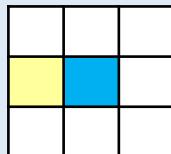
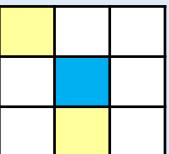
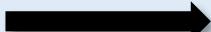
After



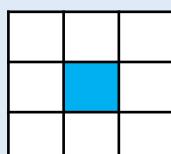
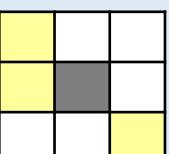
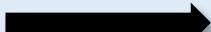
Each live cell with either one neighbour or no neighbours dies of solitude.



Each live cell with four or more neighbours dies of overcrowding.



Each live cell with either two neighbours or three neighbours survives.



Each dead cell with exactly three neighbours comes to life.

Other than a grid that begins empty, what configuration of cells would remain constant, with no cells dying or coming to life? The grids can be of any size.

How would you go about testing that an application that implements this works as required?



Aston University

BIRMINGHAM UK

Software Project Management

Exam Preparation

Thais Webber
Richard Lee



CS3SPM Exam Preparation

- Overview of exam structure and content
- Key advice and best practices
- Revision self-study guide materials
- SPM Q&A forum for exam preparation



- Questions on theoretical and practical aspects of Software Project Management covered in lectures and tutorials
 - Concepts, problems and case studies like those addressed in tutorials

- Questions on theoretical and practical aspects of Software Project Management covered in lectures and tutorials
 - Concepts, problems and case studies like those addressed in tutorials
- Main topics to focus
 - Introduction to Software Project Management
 - Planning software projects
 - Measurement and estimation in projects
 - Risk management in projects
 - Monitoring and control during projects' execution
 - Agile practice and methods

CS3SPM Assessment 100% Exam

- Preparation resources are available on Blackboard:
 - Lectures' Pdfs + Lectures' video recordings
 - Forums/Q&As per unit for sharing questions and discuss answers
 - Tutorials' pdfs: questions and answers provided

- A summary on the exam paper structure and type questions and tips
- A guide for self-study and preparation for the exam to help you organise the content and yourself for the exam



- **Mandatory part (Section A) – (30 marks)**
 - Multiple choice questions only
 - General understanding of module topics / reviews all units
 - Students are required to answer all questions in this part

CS3SPM 100% Closed-book Exam

5) Which of the following options best describes a key difference between PRINCE2 and Step Wise planning?

This one

→

| | |
|----|---|
| a. | PRINCE2 allows overlapping phases, while Step Wise follows a strict, sequential order. |
| b. | PRINCE2 has defined stages and roles; Step Wise is more informal and ad-hoc. |
| c. | PRINCE2 includes stages like planning and control, while Step Wise moves linearly from feasibility to execution. |
| d. | PRINCE2 focuses on risk and quality control, while Step Wise covers the full lifecycle, including budgeting and stakeholders. |

5th Jan — 16th Jan

(1 mark)

CS3SPM 100% Closed-book Exam

4) Mark one 'X' in each row to indicate the most appropriate type of scale for the data examples provided. Note that some scale types may be used multiple times, while others may not be used at all.

(5 mark)

| | Data examples | Ordinal | Ratio | Nominal | Interval |
|----|---|--|-------|---------|----------|
| a. | Language in which an application was developed (e.g. Java, C++) | X X X | | X | |
| b. | Development methodology adopted for a project (e.g. waterfall, incremental prototype) | | | | |
| c. | Measure of difficulty when assessing a user interface feature (e.g. very easy, quite easy, quite difficult) | | | | |
| d. | Probability of being awarded a contact (e.g. 0.25, 0.75) | | | | |
| e. | Rating of programmer expertise (e.g. novice, experienced, expert) | | | | |

CS3SPM 100% Closed-book Exam

1) Which of the following are characteristics of a project but **not** characteristics of exploration? Mark all answers that apply.

(3 mark)

| | | | | |
|----|-------------------------------------|--|-------------------------------------|-----|
| a. | <input checked="" type="checkbox"/> | Clearly-defined deliverable | <input checked="" type="checkbox"/> | + 1 |
| b. | | Open-ended deadline | | |
| c. | | Ambiguous success criteria | | |
| d. | <input checked="" type="checkbox"/> | Individual with overall responsibility | <input checked="" type="checkbox"/> | - 1 |
| e. | <input checked="" type="checkbox"/> | Fixed budget | <input checked="" type="checkbox"/> | + 1 |
| f. | | Iterative testing without a final goal | | |
| g. | <input checked="" type="checkbox"/> | Specific time frame | <input checked="" type="checkbox"/> | + 1 |
| h. | | Flexible objectives | | |

- **Choice-based part (Section B) – (35 marks on each set of questions)**
 - Comprising 3 sets of theoretical and practical questions, including some multiple choice (B.1, B.2, B.3)
 - Students are given the option to choose only two (2) out of these three (3) sets to answer
 - Each set of questions will focus on one main unit of the module, with additional questions from another unit included in the same section

Do the thing → evaluate it

CS3SPM Exam Rubric and Instructions

Date: TBC
Time: 9:30 or 14:00
Duration: 2 hours

5th - 16th Jan

Instructions to Candidates

1. Section A contains ONE question worth 30 marks. Answer the question in Section A.
2. Section B contains THREE questions worth 35 marks EACH. Answer TWO questions ONLY in Section B.
3. Use of calculators is allowed.

Calculators

Students should bring their own calculator, which must be from the Casio FX-83, ← **FX-83** — ✓
Casio FX-85,
Casio 991,
HP10s or HP10s+ series

No other type of calculator is allowed

If you have any questions relating to this,
please contact exams@aston.ac.uk



CS3SPM Exam Rubric and Instructions

Materials provided

- Formulas are provided at the end of the paper
- Answer booklet



Formulas

PERT technique

Lower Upper

$$\text{Activity expected time } t_e = (L + 4M + U) / 6$$

$$\text{Activity variance } v = ((U - L) / 6)^2$$

Critical Path

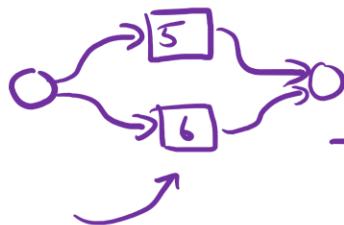
For a sequence/chain of activities:

$$t_e = t_e(A_1) + t_e(A_2) + \dots + t_e(A_n) \text{ and } v = v(A_1) + v(A_2) + \dots + v(A_n)$$

For activities running concurrently:

$$t_e = \max(t_e(A_1), t_e(A_2), \dots, t_e(A_n)) \text{ and } v = \max(v(A_1), v(A_2), \dots, v(A_n))$$

Given a target deadline T , the z value for an activity, a set of activities or an entire project is given by $z = (T - t_e) / \sqrt{v}$, where t_e and v are the PERT expected time and variance respectively.



10 : 00

Measurement, estimation and data analysis

$$\text{Effort} = S \times E_{PS}$$

$$\text{Effort} = S / SpE$$

$$\text{Cost} = S \times C_{PS}$$

$$\text{Cost} = \text{Effort} \times C_{PE}$$

per = divide = /

PM =

Person x Months

SLOC / employee

Ratio factor Additive factor

$$\text{New Project Effort} = \text{Old Project Effort} * (Rf_1 * Rf_2 * \dots) + (Af_1 + Af_2 + \dots)$$

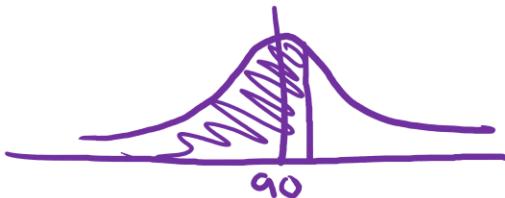
Z-score table

3.142

-1.21

$3.1 + .04$

$= 3.14$



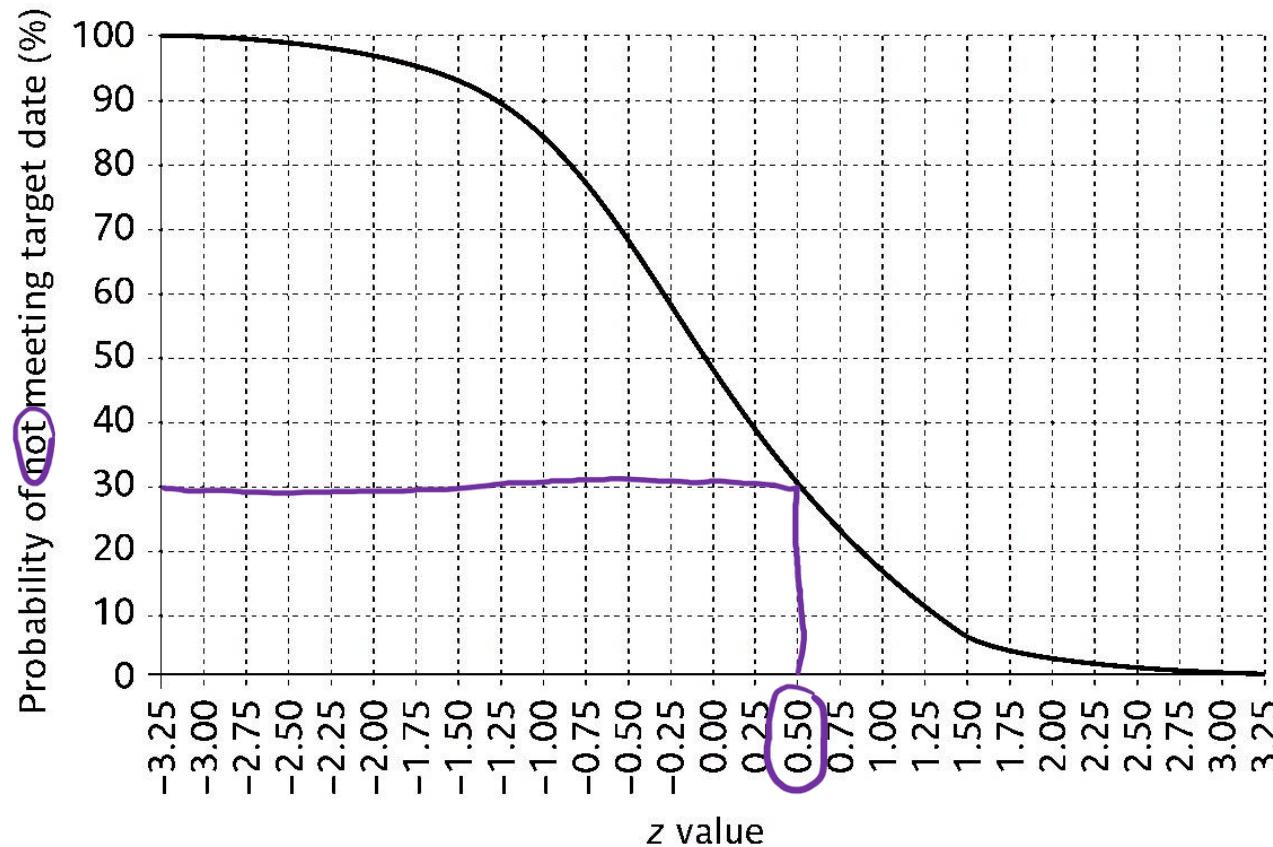
| <i>z</i> | .00 | .01 | .02 | .03 | .04 | .05 | .06 | .07 | .08 | .09 |
|----------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|
| 0.0 | .5000 | .5040 | .5080 | .5120 | .5160 | .5199 | .5239 | .5279 | .5319 | .5359 |
| 0.1 | .5398 | .5438 | .5478 | .5517 | .5557 | .5596 | .5636 | .5675 | .5714 | .5753 |
| 0.2 | .5793 | .5832 | .5871 | .5910 | .5948 | .5987 | .6026 | .6064 | .6103 | .6141 |
| 0.3 | .6179 | .6217 | .6255 | .6293 | .6331 | .6368 | .6406 | .6443 | .6480 | .6517 |
| 0.4 | .6554 | .6591 | .6628 | .6664 | .6700 | .6736 | .6772 | .6808 | .6844 | .6879 |
| 0.5 | .6915 | .6950 | .6985 | .7019 | .7054 | .7088 | .7123 | .7157 | .7190 | .7224 |
| 0.6 | .7257 | .7291 | .7324 | .7357 | .7389 | .7422 | .7454 | .7486 | .7517 | .7549 |
| 0.7 | .7580 | .7611 | .7642 | .7673 | .7704 | .7734 | .7764 | .7794 | .7823 | .7852 |
| 0.8 | .7881 | .7910 | .7939 | .7967 | .7995 | .8023 | .8051 | .8078 | .8106 | .8133 |
| 0.9 | .8159 | .8186 | .8212 | .8238 | .8264 | .8289 | .8315 | .8340 | .8365 | .8389 |
| 1.0 | .8413 | .8438 | .8461 | .8485 | .8508 | .8531 | .8554 | .8577 | .8599 | .8621 |
| 1.1 | .8643 | .8665 | .8686 | .8708 | .8729 | .8749 | .8770 | .8790 | .8810 | .8830 |
| 1.2 | .8849 | .8869 | .8888 | .8907 | .8925 | .8944 | .8962 | .8980 | .8997 | .9015 |
| 1.3 | .9032 | .9049 | .9066 | .9082 | .9099 | .9115 | .9131 | .9147 | .9162 | .9177 |
| 1.4 | .9192 | .9207 | .9222 | .9236 | .9251 | .9265 | .9279 | .9292 | .9306 | .9319 |
| 1.5 | .9332 | .9345 | .9357 | .9370 | .9382 | .9394 | .9406 | .9418 | .9429 | .9441 |
| 1.6 | .9452 | .9463 | .9474 | .9484 | .9495 | .9505 | .9515 | .9525 | .9535 | .9545 |
| 1.7 | .9554 | .9564 | .9573 | .9582 | .9591 | .9599 | .9608 | .9616 | .9625 | .9633 |
| 1.8 | .9641 | .9649 | .9656 | .9664 | .9671 | .9678 | .9686 | .9693 | .9699 | .9706 |
| 1.9 | .9713 | .9719 | .9726 | .9732 | .9738 | .9744 | .9750 | .9756 | .9761 | .9767 |
| 2.0 | .9772 | .9778 | .9783 | .9788 | .9793 | .9798 | .9803 | .9808 | .9812 | .9817 |
| 2.1 | .9821 | .9826 | .9830 | .9834 | .9838 | .9842 | .9846 | .9850 | .9854 | .9857 |
| 2.2 | .9861 | .9864 | .9868 | .9871 | .9875 | .9878 | .9881 | .9884 | .9887 | .9890 |
| 2.3 | .9893 | .9896 | .9898 | .9901 | .9904 | .9906 | .9909 | .9911 | .9913 | .9916 |
| 2.4 | .9918 | .9920 | .9922 | .9925 | .9927 | .9929 | .9931 | .9932 | .9934 | .9936 |
| 2.5 | .9938 | .9940 | .9941 | .9943 | .9945 | .9946 | .9948 | .9949 | .9951 | .9952 |
| 2.6 | .9953 | .9955 | .9956 | .9957 | .9959 | .9960 | .9961 | .9962 | .9963 | .9964 |
| 2.7 | .9965 | .9966 | .9967 | .9968 | .9969 | .9970 | .9971 | .9972 | .9973 | .9974 |
| 2.8 | .9974 | .9975 | .9976 | .9977 | .9977 | .9978 | .9979 | .9979 | .9980 | .9981 |
| 2.9 | .9981 | .9982 | .9982 | .9983 | .9984 | .9984 | .9985 | .9985 | .9986 | .9986 |
| 3.0 | .9987 | .9987 | .9987 | .9988 | .9988 | .9989 | .9989 | .9989 | .9990 | .9990 |
| 3.1 | .9990 | .9991 | .9991 | .9991 | .9992 | .9992 | .9992 | .9992 | .9993 | .9993 |
| 3.2 | .9993 | .9993 | .9994 | .9994 | .9994 | .9994 | .9994 | .9995 | .9995 | .9995 |
| 3.3 | .9995 | .9995 | .9995 | .9996 | .9996 | .9996 | .9996 | .9996 | .9996 | .9997 |
| 3.4 | .9997 | .9997 | .9997 | .9997 | .9997 | .9997 | .9997 | .9997 | .9997 | .9998 |

↓

88.69%

99.92%

Z-score chart



Not

- READ the instructions in each question carefully
- Ensure that ALL questions (mandatory and choice-based) are answered without leaving any blanks
- Each question generally covers one unit as the main topic, potentially incorporating another unit as a secondary topic

Multiple choice questions, some strategic tips that can be helpful:

- Preparation:
 - review key concepts and details from each unit, focusing on understanding rather than just memorisation
 - practice with sampled multiple-choice questions of your own or created by peers brainstorming *or LLMs*



Multiple choice questions, some strategic tips that can be helpful:

During the exam:

- Read each question carefully: understand what is being asked, especially if the question involves "except," "not," "check all that apply," "select one option," etc.
- Narrow down your choices by eliminating options that are clearly incorrect
- Some questions might relate to practical aspects or techniques, use the exam stationery provided to draft and analyse possible correct answers

Advice to answer questions

Theoretical questions, some tips that can be helpful:

- Use bullet points (or a table if a comparison is required)
- DO NOT write long paragraphs with all the differences.
- Be concise!

Advice to answer questions

Theoretical questions, some tips that can be helpful:

- Use bullet points (or a table if a comparison is required)
 - DO NOT write long paragraphs with all the differences.
 - Be concise!
-
- Examples:
 - DISCUSS** the advantages and disadvantages of **TWO** agile methods.
 - EXPLAIN WHY** software project execution requires monitoring?
 - DESCRIBE** the limitations of measuring and/or estimating software size using SLOC.

Advice to answer questions

MARK the correct answer for the scenario and JUSTIFY your choice.

You are the project manager for a software development project that is currently running behind schedule. The initial project plan was to release a new software update to customers in three months, but due to unforeseen technical issues and resource constraints, the project is now expected to be delayed by at least one month. Your team has proposed several options to get back on schedule.

- A. Add more staff 2
- B. Reduce features 14 *
- C. Extend working hours 2
- D. Delay the release 8

Question: Based on the scenario above, which option would you choose to bring the project back on schedule while minimising negative impacts? JUSTIFY your choice. (5 marks)

Advice to answer questions

Practical questions, some tips that can be helpful:

- Usually require to apply a technique or a reasoning approach covered in lectures, discussed during class, and further reinforced through examples and tutorial sessions
- Revisit examples, applications, in lectures and tutorial questions, as part of your exam preparation, and feel free to post any questions or comments in the Q&A forums!

See question → answer it → check mark scheme

Advice to answer questions

Practical questions, some tips that can be helpful:

- Estimation by analogy
- Some complex calculation to establish cost, SLOC, time
- Critical path analysis
- PERT analysis
- (etc.)

Look through
lecture slides

Questions for your FYP (management)

Applying the concepts to your Final Year Project or to any project example

- DESCRIBE (SMART) objectives of your project. HOW they are SMART.
- HOW will you measure the success of the objectives?
- WHAT risks have you identified for your project, and HOW do you plan to mitigate them or minimise their impact?
- WHICH software development approach did you choose? WHAT are the key characteristics of the management approach you are implementing?
- WHAT artefacts are you using to manage your project, and HOW do they specifically support your FYP management and objectives?

SMART objectives - examples

Objective: Develop a fully functional prototype of a mobile health app that accurately tracks and analyses user dietary habits, achieves at least 90% accuracy in identifying and logging food items from user inputs in controlled test scenarios, and incorporates feedback from at least 10 trial users to refine usability and functionality.

Objective: Develop and deploy a web-based interactive learning platform for high school students that supports at least three different science subjects, includes real-time feedback and automated grading systems for quizzes and assignments, and achieves a user engagement rate of over 70% as measured by average session durations and quiz completion rates.

Question on project planning

Suppose you have a project with a defined critical path. If the duration of one activity on this critical path is extended, what impact could this have on the project's overall timeline?

Why?
So what?
Example?
Definition?

Generic

Question on risk management

In project risk management, why is it important to assess both the likelihood and the impact of potential risks?

Provide an example of how you might position an identified risk within a matrix based on these two factors and justify your analysis.

Note: revise concepts and risk mitigation strategies or planning you would apply for each identified risk; also check examples, tutorial exercises on this unit.

Questions on overall module content

Note: revise concepts and techniques of ALL core topics in the CS3SPM module;

This session provided a few discussions as starting points, do not forget to check other units content and tutorials.

This lecture session was intended to set your mindset to prepare for the exam, and to organise your studies presenting the exam preparation materials. It was not intended to provide a full revision of all topics of the module.

For that, any questions, please use the Q&A, e-mails, or even you can book instructors WASS meetings.

We are here to support your learning journey.



Aston University

BIRMINGHAM UK

Software Project Management

Exam Preparation

Thais Webber
Richard Lee





Aston University

BIRMINGHAM UK

Here's to 2026 being
less of an unqualified
disaster than 2025

Thais Webber
Richard Lee

