Specification

Terminology

- **Output = what the project literally delivers** e.g. new software
- **Outcome = the change resulting from the output** e.g. reduce staff workload
- **Benefit = the amount of advantage gained by the outcome** e.g. increase in employee retention by X%
- Objective/goal = an intended benefit or outcome — **if asked to give objectives they should be SMART even thought this probably won't be specified:**
  - Specific
  - Measurable
  - Achievable
  - Relevant
  - Time-bound

Iron triangle

- **Iron triangle: Quality is rigidly constrained by: cost, time, and scope**
- Because business students can only think in 2D, we fix a quality level. Then, there are lower bounds on time and cost and an upper bound on scope — **these 3 constraints form a triangular feasible region that you can move around in as you make trade offs**
  - **The nearer to a side a point is, the better it is for that factor**
- Top vertex = fully-featured and on time but expensive
- Bottom left vertex = low-cost and on-time but few features
- Bottom right vertex = fully-featured and low cost but late

Work Breakdown Structure

- **Work Breakdown Structure (WBS): A deliverable-oriented hierarchical decomposition** of the work to be executed by the project team
  - **Deliverables = the outputs of the project. Should be aligned with the objectives**
- **Work packages = the children of the root of the WBS**
- A milestone = a control point that separate work packages
- Smaller work packages are easier to estimate and allow tasks to be parallelised
- Larger work packages reduce micromanagement

Numerical Tools

Time management: Project network diagram

- Arrows between items on a Gantt chart show dependencies
- Project network diagram = a graph of the tasks and dependencies, in which there is **a single (possibly dummy, with duration 0) source and sink**, used to identify a critical path
- **Activity on node form = node are activities, edges represent dependencies**
- **Event on node form = nodes are abstract, edges represent dependencies (possibly some only transitively) but are labelled with activities — some edges may need to be dummy activities with duration 0 to create combinations of activities**

Time management: Critical Path Method

- A critical path = a sequence of activities starting from the first activity and ending with the last such that no activity can be delayed without increasing the project duration — there always exists a critical path
- Each activity is assigned a box:
  - Top left = earliest start (ES) = maximum EF of parents or 0 if is start
  - Top middle = duration (D)
  - Top right = earliest finish (EF) = ES + D
  - Middle = activity name
  - Bottom right = Latest finish (LF) = minimum LS of children or own EF if is end
  - Bottom middle = Total float (TF) = LS - ES = LF - EF
  - Bottom left = Latest start (LS) = LF − D
- Forward pass: Using a breadth first search from source to sink: Populate ES then EF for each activity
- Backward Pass: Populate LF then LS and TF for each activity

Time management: Crashing

- **Total float (TF) = slack time = the amount of time an activity can be delayed from its ES without delaying the project end** = LS − ES = LF − EF
  - **A task is critical iff it has TF = 0**
  - **Critical path = a path along which all tasks are critical**
- **Free float (FF) = the amount of time an activity can be delayed without delaying the ES of any subsequent activities = min(ES of children) − own EF**
  - Deduce that FF ≤ TF — often FF = TF
  - Note that FF only uses information from the forward pass
- **Drag time = the amount the duration of a given <u>critical task</u> would need to be reduced by in order to be no longer critical = min(own duration, min(TF over all tasks that overlap <u>at all</u> with it (not including itself))** — if there are no tasks occurring at the same time, then it is just duration
- **Crashing = increasing costs to speed up project completion**
  - **To speed up the project we need to speed up tasks on the critical path**
  - It's a waste of money to speed up a task beyond the point where it becomes no longer critical
    - **Crash duration = amount it makes sense to remove from the duration of a critical task = duration − drag**

Time management: PERT

- **Program Evaluation and Review Technique (PERT): Estimate the shortest possible duration (a), the most likely (i.e. mode not mean) duration (m), and the longest possible duration (b)**
- Fit either a beta or triangular distribution: Formulae for mean and variance from these parameters will be given in exam but the meanings of the parameters won't be
- **The task-wise means and variances allow us to calculate (by summing up along the critical path) the expected duration and variance of critical path** *(by linearity of expectation, and preservation of summation of variance for independent random variables)*
    - Be careful: If working with standard deviations need to square into variances before summing (and square rooting the result to get the total standard deviation)
    - **We can then plug these into the normal distribution to calculate the probability of the project completing within a given time interval** — draw a diagram if given a table to ensure 1- vs 2-tailedness is handled appropriately

EVA: Primitives

- **Budget at completion (BAC) = planned total cost of all work = project budget**
- **Planned value (PV) = budgeted cost of work scheduled up to now = BAC * proportion of project scheduled for up to a given point in time**
- **Actual cost (AC) = actual cost of work performed up to now = how much of the planned budget has been spent at a given point in time = directly measured**
- **Earned value (EV) = budgeted cost of work performed up to now = how much of the planned value has been achieved by doing the work so far at a given point in time = BAC * proportion of project completed**
    - Allows us to distinguish between underspending due to being under budget and underspending due to running late (and symmetrically for overspending and running early)
- In these formulas we are assuming that cost and time are evenly distributed across the work!

EVA: Analysis

- **Schedule variance (SV) =** on schedule iff non-negative = **EV – PV**
- **Schedule performance index (SPI) =** time-efficiency = **EV/PV**
- **Cost variance (CV) =** within budget iff non-negative = **EV – AC**
- **Cost performance index (CPI) =** cost-efficiency = **EV/AC**
- **Cost schedule Index (CSI) =** overall efficiency = **SPI*CPI**
- Note that they all start with earned value
- Note that the variances are all subtractions, and the indexes are all divisions

EVA: Forecasting

- **Estimate at completion (EAC) = expected expenditure on all work, given certain assumptions**
  - **Based on progress to date: BAC * 1/CPI — rescales the entire budget by cost efficiency so far**
  - **Based on original estimate: AC + (BAC − EV) — assumes the project will continue exactly as planned from now on**
  - **Based on overall efficiency: AC + (BAC − EV) * 1/CSI — rescales the remaining original expected spend by both cost- and time-efficiencies so far**
- **Estimate to complete (ETC) = how much more will need to be spent to complete the project = EAC − AC**
- **Variance at completion (VAC) = expected overspend or underspend = BAC − EAC**
- **To-complete performance index = the future cost efficiency required to stay within a certain budget**
  - **Estimated budget: $TCPI_{EAC}$ = (BAC − EV)/(EAC − AC)**
  - **Original budget: $TCPI_{BAC}$ = (BAC − EV)/(BAC − AC)**

Methodologies

PMBOK: Knowledge areas

1. *Scope*
2. *Time*
3. *Cost*
4. *Quality*
5. *Risk*
6. *Resources*
7. *Procurement*
8. *Stakeholders*
9. *Communication*
10. *Integration (everything that doesn't fit into one of the other categories)*

PMBOK: Process areas

1. **Initiation**
2. **Planning**
3. **Executing**
4. **Monitoring and controlling**
5. **Closing**

PMBOK: Project initiation

- **Prior to initiation, a business case and project mandate are developed**
- **The project mandate** (outline specification) **is a statement of work: explains what the project would produce:**
  - Scope
  - Constraints
  - Assumptions
  - Known risks
- **In the initiation phase, the project charter (full specification) is produced** — requires sign-off from stakeholders before the project can advance to the planning phase

PRINCE2: Principles

1. **Continuous business justification** — don't fall victim to sunk cost fallacy! **(avoids waste like lean)**
2. **Learn from experience** — don't make the same mistake twice! **(like kaizen in lean)**
3. **Defined roles and responsibilities** — every task has someone accountable for it otherwise it might never get done
4. **Manage by stages** — don't plan everything right at the beginning a la waterfall, take a more iterative type approach **(like agile)**
5. **Manage by exception** — set limits but give workers agency within these limits — control without micromanaging! **(like agile and lean)**
6. **Focus on products** — focus on outputs/deliverables — focus on what not how (see give workers agency) or why (well maybe a little bit to determine which outputs are most important, but ultimately its only the outputs you can control not their outcomes)
    - **There are "management products" to create a paper trail just-in-case the project fails — not lean**
7. **Tailor to suit the project environment** — only follow PRINCE2 as closely as is not detrimental in <u>your</u> project

PRINCE2: Themes

1. *Business case — corresponds to business justification*
2. *Organisation — corresponds to roles and responsibilities*
3. *Quality — corresponds to product focus*
4. *Risks*
5. *Plan*
6. *Change — corresponds to manage by exception*
7. *Progress — corresponds to manage by stages*

PRINCE2: Processes

1. **Starting Up ((pre-)initiation in PMBOK)**
2. **Initiating a project (Initiation in PMBOK) — produce high-level plan for entire project**
3. **Managing a stage boundary (Planning in PMBOK) — produce a detailed plan for the current stage**
4. **Controlling a stage (Monitoring and controlling in PMBOK)**
5. **Managing product delivery (Execution in PMBOK)**
6. **Closing project (Closing in PMBOK)**
7. **Directing a project (no equivalent in PMBOK!) — runs the entire time, project board is responsible for approving the outputs of each process/stage before the next commences**

PRINCE2: Roles

- **Project board: Held accountable for the project's success (not the project manager). Members:**
  - **Executive: Represent the interests of stakeholders (the business)** — chairs the board
  - **Senior user: Represents interests of end user**
  - **Senior supplier** (poorly named): **Represents the interests of project workers**
- **Project manager: Runs the project day-to-day, but their power derives from the board and is limited** (must go to board to ask for more as necessary)

Agile

- Unlike traditional plan-driven approaches, Agile is requirements-driven
- Plan-driven approaches require the cost and time to be estimated for a fixed scope whereas agile approaches require the scope to be estimated for fixed cost and time
- **Agile manifesto:** (when we say value X over Y we don't mean that Y isn't valued, only that X is valued more)
  - **Individuals over processes**
  - **Software features over documentation**
  - **Customer collaboration over negotiation**
  - **Responding to change over following a plan**

Should we use Agile for a given project?

- Agile requires the customer to be available to provide regular feedback
- Agile allows the customer to change their mind
- Agile is well-suited to projects with unknown scope (such that those that involve complex problems for which the appropriate approach is not obvious at the outset) — for these plans wouldn't be worth the paper they are written on
    - Agile decreases known and unknown risks over time due to delivery of usable minimal viable products whereas PRINCE2 maintains risk until the project delivers at closure (albeit PRINCE2 goes to great effort to manage known risks, whereas Agile may take a we'll worry about that if it materialises attitude)
- Agile is especially useful when the customer can get genuine value out of incremental deliverables

The 12 agile principles

1. *Early and continuous delivery of valuable software*
2. *Welcome changing requirements, even late in development*
3. *Deliver working software frequently (e.g. every 2 weeks)*
4. *Business people and developers must work together daily*
5. *Build around motivated individuals with support and trust*
6. *Face-to-face conversations*
7. *Working software is the primary measure of progress*
8. *Sustainable: able to maintain a constant pace indefinitely*
9. *Excellence: good design enhances agility*
10. *Simplicity: the art of maximizing the amount of work not done*
11. *Self-organizing teams*
12. *Reflect on how to become more effective*

Scrum

- **Scrum is an agile methodology focused around user stories grouped into epics**
- **User story: As a … I want … so that …** — is output-focused not process-focused
  - Assigned a size estimate (story points) and a priority
- **The product owner converts stakeholder inputs into user stories in the product backlog and prioritises them — the product owner acts as the day-to-day representative of the customer**
  - **Engages in regular backlog refinement to keep these up to date**
- **The scrum master runs the logistics (e.g. hosting the meetings) without being controlling (leader not a manager)**
- **The development team does the actual work** — if there is more than 9 people you need to split the team or you will spend longer in standup than actually doing work

Scrum: Ceremonies

- **Daily stand up to synchronize on blockers etc**
- In each 1–4 week sprint the most important user stories are completed — **the sprint planning meeting determines how much can be done in the sprint and so which user stories will be completed**
- **Sprint review demonstrates project progress to stakeholders (e.g. customer)**
- **Sprint retrospective determines how better estimates can be made next time and what further user stories are needed**

Scaling agile

- Agile can be hard to scale — in reality you simply become less agile
- **Disciplined Agile Delivery adds PRINCE2 style governance to an iterative agile approach with sprints (e.g. Scrum)** — sprints are short as in proper agile
- **Scaled Agile Framework (SAFe): Agile (e.g. ScrumBan) teams each with their own backlog and (8-12 week** to the horror of true agile practitioners**) sprints. Teams are managed by the Agile Release Train (execs) who use Kanban to manage their workload**
    - **Has continuous delivery** but everything else is feeling quite cargo-cult

Lean: Principles

- For manufacturing (original):

1. **Eliminate waste**
2. **Amplify learning** — get feedback
3. **Defer commitment** — JIT
4. **Deliver fast** — get feedback early
5. **Empower the team**
6. **Build integrity in** — defects are wasteful
7. **See the whole**

- For software engineering:

1. Eliminate waste
2. Create knowledge
3. Defer commitment
4. Deliver fast
5. Respect people
6. Build quality in
7. Optimise the whole

Lean: The 3 wastes

- Sometimes called the 3 mudas which is just confusing
- **Muri — overburden**
  - **Working at an unsustainable pace**
  - **Muri → carry — do not confuse with mura**
- **Muda — no added value**
  - **Consuming resources without creating value**
  - **Muda → duff**
- **Mura — unevenness**
  - **Variability in flow** e.g. not aligning supply and demand
  - **Mura → ragged**

The Lean House

- **Jidoka = intelligent automation = automation with self-monitoring to stop if a fault occurs** — provided to software engineering by DevOps e.g. CI/CD pipelines
- **Heijunka = production levelling = steady consistent work that flows nicely without stop-start**
- **Kaizen = continuous improvement = always aiming to further reduce waste**

The 5 Pillars of Lean Thinking

1. **Identify Value — have customer define what value is to them, so you can plan how you will create it**
2. **Map The Value Stream — identify which steps in the product lifecycle create value, which don't create value but are unavoidable (type 1 muda), and which don't create value and aren't necessary so should be removed (type 2 muda)**
3. **Create Flow — execute the value stream, following heijunka**
4. **Establish Pull — JIT not just-in-case** — don't make things ahead of time
5. **Seek Perfection — kaizen**

Kanban: Principles

- A push system = make everything you could need just in case — more wasteful than a pull system in which you make only what is needed
- **A just-in-time (JIT) system = a pull system with small buffers (regulators** e.g. mcdonalds burger cache**) between steps, outputs flow out of the regulator and requests for new outputs flow into the regulator** — able to meet demand, but not stockpiling
- **A minimal viable product (MVP) = a deliverable <u>that is useful to the customer</u> and contains no more than is necessary**
    - **Working by iterating on MVPs maximises the flow of value to the customer and so is a lean approach**
- **Kanban** (japanese for look board) **is a work-in-progress limited pull system** best known for the kanban board (yes this means board board)
    - **Work-in-progress limits: Reduces muri by definition. Also, reduces muda as reduces number of task-switches** (and task-switches have overhead)
    - **Pull system: Kanban board is not a fixed to-do list — you do one thing, then decide what would give the most value to do next**

The Kanban board

- **Work is not pushed onto you (work is not assigned to you), instead you look at the board and pull the highest priority task aligned to your skill set when <u>you</u> think you are ready to take on another**
- **Visualises not only work but how it is flowing (what stage each task is in in the flow** (e.g. waiting for or actively: developing, testing, deploying) **and what dependencies each task has)**
    - The waiting for pile (i.e. done in the previous stage) is a pile to be pulled from when available, not work being pushed onto someone with a particular deadline
- Identifying and addressing blockers is a key part of making Kanban effective — as everyone can see the board, everyone can contribute towards kaizen
    - **Being blocked is a red-flag that there is a bottleneck in the flow — for kaizen, processes will be improved to even out the flow (heijunka)**
- **Hitting a WIP limit is a signal that flows backwards through the flow**
    - The smaller the WIP limit is, the sooner issues will be spotted

Scrum vs Kanban

- In Scrum work is in large chunks (sprints) that are then broken down (sprint goals) and work is towards a known long-term goal (horizon)
  - Fixed timboxes mean rate of progress can be measured to allow the time to reach the long-term goal to be estimated (e.g. burndown chart)
- In Kanban the work arrives to the team already in small pieces, the team are not planning towards a long-term objective. As far as the team are concerned work just appears
  - Very responsive to the arrival of new work to the project

Risk Management

Risk

- **Risk is the (not necessarily negative) effect of uncertainty on objectives** *[ISO 31000]*
- **An issue = a risk that has materialised**
- PMBOK has a risk knowledge area to avoid risk
- PRINCE2 has the board to avoid risk
- Agile's iterations avoid the risk of not delivering what the customer expects
  - Embracing change allows us to exploit positive risks
- Lean's mudas avoid the risk of waste

Analysing risk: Qualitatively

- **RACI Matrix: Records who is Responsible, Accountable, Consulted, and Informed for each project activity**
  - At least one person responsible (completing the task) for each activity
  - **<u>Exactly</u> one person accountable for each activity**
- **SWOT analysis:**
  - **Strengths**
  - **Weaknesses**
  - **Opportunities**
  - **Threats**
- **Delphi technique: Experts answer questionnaires anonymously, then see each others responses to update their own, iterating until convergence**

Ishikawa diagram

- **Ishikawa diagram** (aka fishbone diagram) *invented by Ishikawa Kaoru*: **Work backwards from a risk (an effect) to think about how (i.e. the causes of the risk) its probability can be increased or decreased (depending on whether it is a positive or negative risk)**

Sensitivity analysis

- Sensitivity analysis measures how changing an independent variable affects a dependent variable
- **Spider graph: Line graph of dependent variable against deltas to one or more independent variables** (each with their own base value and scale defined somewhere)
- **Tornado diagram: Bar graph of height between lowest and highest point on spider graph for each variable**

Analysing risk: (E)FMEA

- FMEA provides a structured method for determining the numbers for a risk matrix
- **FMEA (Failure Mode Effects Analysis): For each task, identity potential failure modes. For each failure mode, identify the: failure effect, severity of the effect (S), cause, probability of occurrence of the cause (O), controls (detection measures), difficulty of detection (D)** (this is difficulty so that for all the numbers big is bad) **— this allows for each failure mode the criticality (C) = S x O and risk priority number (RPN) = S x O x D to be computed**
  - **D only really takes into account whether it will ever be detected not how soon it will be detected** the consequences of late detection and with the probability of late detection should both be factored into S instead
- **Extended Failure Mode Effects Analysis (E-FMEA): Identify action that can be taken to improve the RPN for each failure effect. Compute new RPN and divide by the infeasibility of action F. Take the actions in order of descending $(RPN_{OLD} - RPN_{NEW})/F$** — we won't have the resources to take every action, so we want to get the most value possible from what we do do

Leadership

Decision making

- PrOACT decision making
  - Problem: Define the problem: what are you doing and why are you doing it
  - Objectives: Specify the objectives: what are the aims and how can they be measured
  - Alternatives: Imagine the alternatives to set a benchmark
  - Consequences: Compare the consequences of the alternatives
  - Trade-offs: Weigh up which objective is most important
- We need managers to be leaders: to be decisive, and to make good decisions quickly

Team-building

- Lencioni's 5 dysfunctions of a team: Absence of trust leads to fear of conflict leads to lack of commitment leads to avoidance of accountability leads to inattention to results
- Tuckman's stages of group development:
  1. Forming: reliant upon formalities; treating other team members as strangers — slight productivity as treating each other cordially but not working together
  2. Storming: starting to communicate feelings but still see it as me vs the rest — conflict reduces productivity
  3. Norming: feel a part of the team and start to welcome other viewpoints — productivity surges back up as the conflicts are resolved to build consensus
  4. Performing: open and trusting atmosphere; hierarchy is of little performance
  5. Adjourning: project ends; reflect on the teams achievements
- Edison's expansion of group development: When performing are initially informing in which lessons are being learnt and shared but eventually become conforming in which the team has converged too much and so is vulnerable to groupthink (no longer really more than the sum of the parts as the parts are too similar)
  - PM should engage in transforming the team when conforming occurs — reorganise the team and bring in new blood to restart the process from forming