FDIT:
PROJECT MANAGEMENT AND PERFORMANCE

Week 3

J Zander

Week 3 Overview

Learning Objectives:

- Interpret and analyze client briefs
- Draft a project scope with objectives, stakeholders, and deliverables
- Break down features into tasks and **estimate effort**
- Apply basic prioritization and risk assessment methods

What is Scope?

Definition of Project Scope

Project scope is the documented set of:

- **Objectives** → What problem the project is solving (the “why”).
- **Deliverables** → Tangible outputs (website, app feature, prototype, documentation).
- **Requirements** → Functional and non-functional details that the deliverables must meet.
- **Boundaries** → What’s *out of scope* (e.g., “this version won’t include Android support”).
- **Constraints** → Time, budget, technology, or resource limits.
- **Acceptance criteria** → How success will be measured.

Why Scope Matters

Why It Matters in Tech Projects

1. **Prevents scope creep** – uncontrolled growth of features beyond what was agreed.
2. **Sets expectations** – aligns client, developers, designers, and stakeholders.
3. **Defines success** – makes it possible to judge whether the project was completed as promised.
4. **Improves planning** – allows for accurate effort estimation and scheduling.

Example

Lululemon: Designing a wearable.

Briefs

Many projects (especially at agency) start with a BRIEF

- Design Brief, Client Brief, Dev Brief etc.)
- Key elements:
  - Audience (Users)
  - Objectives
  - Constraints

Take a Design Brief: get into a group of 2.
Identify the following:

- Audience
- Key Deliverables
- Goals
- Constraints
- Missing information!

WHAT QUESTIONS WOULD YOU ASK A CLIENT FOR CLARITY
ARE YOU READY TO START WORK BASED ON WHAT YOU SEE?

Writing a Project Brief

Should be concise, informative, and actionable.
The purpose of the brief:

- Get stakeholder alignment
- Define Scope
- Define the Problem to be Solved
- Define Audience (USERS)
- Constraints

Project Brief: Step 1 WHY

- What is the problem to be solved?
- What is the opportunity?

Example: 80% of users bounce during onboarding, we will redesign onboarding to optimize retention.

Project Brief: Step 2: Objectives

Objectives (goals) need to be SMART
S=Specific
M=Measurable
A=Achievable
R=Relevant
T=Time Bound

Don’t go crazy: Keep to a few (2-5) that are specific and realistic. Too many isn’t good- people lose focus.

Project Brief: Step 3: Deliverables

What are we developing? (Do we know yet?)
Be specific: website, App, prototype, dashboard etc.

- This means it is important to avoid being vague: i.e., improve UX is vague- go back to the goals and be specific- what are the problems and what do you need to improve?

Project Brief: Step 4: Scope

- What is out of Scope? This is important because Scope Creep is always a problem!
- For example: we are doing responsive design down to 688 pixels: smaller screens are not supported/
- Designing for iOS, Not Android etc.

Project Brief: Step 5: People!

- Who are the stakeholders?
- What are their roles?

1) Who is the client (is there a client?)
2) Who approves the work?
3) Who needs to be consulted?
4) Who are end users ?

Note- this is like the RACI- you can just use that here!

Project Brief: Step 6: Define the User!

User Centred Design! Critical to know your user!

1) User Profile
2) Personas

How much have we done here? (JZ to go into other presentation)

Project Brief: Step 7: Constraints/ Assumptions

Budget
Timeline
Tech stack (if you know - if not you need to build this in)
Integrations
Dependencies

I.e., Must launch before Chinese New Year.

Project Brief: Step 8: Define Success

What does success look like?

Metrics should tie back to goals/ objectives

Draft a brief:

What is our problem to be solved?

SCOPE DOCUMENT

A project brief gives a brief scope explanation- but you also need to define scope in a SCOPE DOCUMENT- that usually starts to be defined (in detail) once a project brief has been delivered (brief is really an estimate)
This is really an agreement of what work you and your team will do!

Scope Document

- Goals:
  - Objectives (measurable)
- Deliverables:
  - This is the reason you are doing the work- what specifically will you deliver?
  - These often include ACCEPTANCE CRITERIA
- Timeline
  - Roadmap
  - Gantt Chart or similar
- Milestones
  - Phases, sprints, what will be delivered initially vs later on?
- Reports
  - What will be reported? When and how?

Template

- Link to sample template
- HERE (it is in Moodle)
- Called Scope of Work Template for Project Management Course

- Let’s work on Scope of Work
- We’ll do it together

- Things to think about: how much detail do we need?
- Where do we find the detail? How would we know?

Estimating Effort and Prioritizing

- T shirt sizing (seems popular lately)
- Story Points (Agile standard, but often Agile works with T shirt sizing)
- Planning Poker
- Person Hours/ Person Days
- Functional Points Analysis
- There are more, but these are the commonly used ones and they all have similarities

T Shirt Sizing

What it is: Relative estimation using simple categories like XS, S, M, L, XL. (Need to define what these mean as a team first!)

- **When to use:** Early-stage planning, when details are fuzzy. Good for backlog grooming.
- **How to do it:**
    1. Take a feature/story.
    2. Compare it to others in the backlog.
    3. Place it into a size bucket. (e.g., *"This feature is about as big as that M-sized one, but smaller than that L."*)
- **Pros:** Quick, non-threatening, works when info is limited.
- **Cons:** Not precise, eventually needs translation into time or story points.

Points

- **What it is:** Relative measure of complexity, effort, and risk using numbers (see Planning Poker)
- **When to use:** Sprint planning in Scrum or Agile teams.
- **How to do it:**
    1. Team discusses the user story.
    2. Each member secretly picks a point value (using *planning poker or similar*).
    3. Reveal estimates → discuss discrepancies → agree on a number.
- **Pros:** Balances effort and complexity, builds team consensus.
- **Cons:** Can confuse outsiders (not “hours”), needs calibration over time.

Planning Poker

- **What it is:** A game-like version of story points where team members estimate simultaneously to avoid groupthink. I.e., you don't know anyone else's opinion before you 'show your cards'
- **When to use:** Sprint planning, backlog refinement.
- **How to do it:**

    1. Each team member gets a deck with numbers (Fibonacci or T-shirt sizes).
    2. PM reads the story.
    3. Everyone reveals a card at once.
    4. Discuss big gaps, then vote again until consensus.

- **Pros:** Fun, democratic, reduces anchoring bias.
- **Cons:** Takes time with large backlogs. Requires a lot of knowledge to estimate.

Points

- **What it is:** Relative measure of complexity, effort, and risk using numbers (see Planning Poker)
- **When to use:** Sprint planning in Scrum or Agile teams.
- **How to do it:**
    1. Team discusses the user story.
    2. Each member secretly picks a point value (using *planning poker or similar*).
    3. Reveal estimates → discuss discrepancies → agree on a number.
- **Pros:** Balances effort and complexity, builds team consensus.
- **Cons:** Can confuse outsiders (not “hours”), needs calibration over time.

Person Hours (very commonly used)

- **What it is:** Estimate in actual time units (hours, days).
- **When to use:** When deadlines or budgets require calendar-based commitments.
- **How to do it:**
    1. Break tasks into small enough pieces (ideally ≤ 1 day each).
    2. Estimate each piece in hours/days.
    3. Add up totals, include buffer (usually +20–30%).
- **Pros:** Direct and familiar for business stakeholders. This is often used in conjunction with other methods at the team level.
- **Cons:** Tends to be over-optimistic; can ignore risks and hidden complexity

Functional Points Analysis

- **What it is:** Breaking down software into measurable “functions” (inputs, outputs, data files, user interactions) and assigning weights.
- **When to use:** Large enterprise projects, regulated industries.
- **How to do it:**

    1. Identify all functions (e.g., login screen, report generator).
    2. Assign complexity weights.
    3. Calculate total function points.
    4. Convert into effort via historical productivity data.

- **Pros:** Rigorous, standardized, good for enterprise budgeting.
- **Cons:** Time-consuming, less common in modern Agile startups. IT IS SLOW

Which one do you think you’d like to try?

Prioritization

You will end up with a LONG list of features- and you need to PRIORITIZE

Popular:MoSCoW method (Must, Should, Could, Won't)

Others:

- RICE
- Value Vs Effort Matrix (grid)
- Opportunity Scoring
- WSJF (Weighted Shortest Job First)

MoSCoW

- **M**ust have → Essential to project success.
- **S**hould have → Important, but not critical for the first release.
- **C**ould have → Nice to include if time/resources permit.
- **W**on't have (this time) → Agreed exclusions for now, might be added later.

How it works

1. Gather a list of requirements, features, or tasks.
2. Assign each item to one of the four categories
3. Review and validate with stakeholders to confirm priorities.
4. Use these categories to define scope for releases or sprints.

Pros

- **Simple & fast** → Easy to understand and apply, even for non-technical stakeholders.
- **Manages expectations** → "Won't have" clearly communicates exclusions.
- **Great for MVP definition** → Perfect for deciding what goes into the first release.

Cons

- **Subjective** → No scoring system, decisions may be influenced by opinions.
- **Overloaded categories** → Risk that everything becomes a "Must have."
- **Lacks nuance** → Doesn't rank within categories (all "Musts" are treated equally).
- **Not data-driven** → Unlike RICE or WSJF, it doesn't use effort/impact calculations.

RICE

- **What it is:** Prioritization based on **Reach, Impact, Confidence, Effort**.
- **How it works:**
  - Score each feature on:
    - *Reach* (how many users will it affect?)
    - *Impact* (how much value does it add?)
    - *Confidence* (how sure are we of the estimate?)
    - *Effort* (how much work/time is required?).
  - Formula: **(Reach × Impact × Confidence) ÷ Effort**.
- **Use case:** Great for data-driven teams prioritizing product roadmaps.

Value vs Effort

- **What it is:** Plots tasks/features on a matrix of **Value (high/low)** vs. **Effort (high/low)**.
- **Quadrants:**
    - *Quick Wins* (High value, low effort) → do first.
    - *Major Projects* (High value, high effort) → plan carefully.
    - *Fill-Ins* (Low value, low effort) → do if time.
    - *Time Wasters* (Low value, high effort) → avoid.
- **Use case:** Simple visual tool for teams new to prioritization.
-

Opportunity Scoring

- **What it is:** Plots tasks/features on a matrix of **Value (high/low)** vs. **Effort (high/low)**.
- **Quadrants:**
  - *Quick Wins* (High value, low effort) → do first.
  - *Major Projects* (High value, high effort) → plan carefully.
  - *Fill-Ins* (Low value, low effort) → do if time.
  - *Time Wasters* (Low value, high effort) → avoid.
- **Use case:** Simple visual tool for teams new to prioritization.

Weighted Shortest Job First (WSJF)

- **What it is:** From SAFe (Scaled Agile Framework). Focuses on maximizing ROI by prioritizing jobs with highest value delivered per time unit.
- **Formula: WSJF = Cost of Delay ÷ Job Duration**.
- **Factors in "Cost of Delay":** User/business value, time criticality, risk reduction/opportunity enablement.
- **Use case:** Works well in Agile at scale (portfolios, multiple teams).

Risk Assessment

- What should we be looking for?
- Why do we care about risk assessment?
- Whose job is it to ID risks?

Create a Risk Register (or similar) Risk Tracking- so you can log likelihood, severity, mitigation plans.

Common Risk Factors

1. Technical Risks
2. Project Management Risks
3. Resource Risks
4. Operational Risks
5. Strategic Risk
6. UX Risk (*)

Technical Risks (common)

- **System failures / downtime** → Server crashes, outages, or unstable architecture
- **Integration risks** → Compatibility issues between APIs, third-party services, or legacy systems.
- **Scalability** → Can the system handle increased users or data? (how do you know? Are there limits?)
- **Security vulnerabilities** → Data breaches, weak authentication, lack of encryption.
- **Unproven technology** → Using a new framework, tool, or platform that may not be stable.

Project Management Risks

- **Scope creep** → Uncontrolled expansion of requirements without resources.
- **Unclear requirements** → Ambiguity leading to rework or wasted effort.
- **Unrealistic timelines** → Pressure to deliver faster than feasible.
- **Budget overruns** → Underestimating costs for dev, QA, or infrastructure.
- **Dependency delays** → Reliance on external vendors, other teams, or approvals.

Resource Risks

- **Skill gaps** → Team lacks expertise in specific tech or methods.
- **Key-person dependency** → Reliance on one developer, designer, or PM.
  - (what if they get sick? Quit? Go on Vacation?)
- **Turnover / burnout** → High workload or stress leading to attrition.
- **Poor collaboration** → Misalignment between dev, design, QA, or stakeholders

Operational Risks

- **Data management issues** → Data loss, poor backups, or compliance failures.
- **Infrastructure failures** → Cloud misconfigurations, network downtime.
- **Process bottlenecks** → Manual workflows slowing down delivery.
- **Third-party risks** → SaaS provider downtime, API changes, or licensing issues.

Strategic Risks

- **Market shifts** → Competitor releases, tech obsolescence, or user expectations changing. (Let's talk about who manages this risk)
- **Regulatory & compliance** → GDPR, HIPAA, SOC 2, or other industry standards.
- **Customer adoption** → Risk that product doesn't meet user needs or gain traction.
- **Reputation risk/ Brand Damage** → Poor release quality harming trust.

UX Risks (poor usability)

- **Accessibility gaps** → Product not accessible (litigation and usability problem).
- **Performance issues** → Slow load times, lag, poor mobile optimization.
  - Performance is a FEATURE!
- **Usability challenges** → Confusing workflows, leading to abandonment.

- **NOTE: all other risks can also apply to UX. And Poor usability can certainly damage the brand!**