Week 2

Agile Requirements & Planning – Detailed Note

1 Learning Outcomes

After this lesson, learners will be able to **explain**, **demonstrate**, and **apply** good practices for:

1. Requirements **review & confirmation**.
2. Establishing a **requirements baseline** and running **change-control**.
3. Translating baselined requirements into an **agile backlog**.
4. Building and iterating a **Gantt chart (v1)** that aligns milestones and sprints.
5. Creating a prioritized **product backlog** with **story points**.
6. Operating a lightweight tooling stack of **Notion + GitHub Issues** for transparent execution.

---

2 Requirements Review & Confirmation

2.1 Purpose

Why it Matters	Impact if Skipped
Validate understanding between business and delivery teams	Mis-aligned scope → rework + delays
Surface ambiguities early while change cost is low	Defects discovered during QA or UAT
Define acceptance criteria that drive testing	Vague requirements → subjective acceptance

2.2 Timing & Inputs

- Occurs **immediately after requirement elicitation** but **before estimation or commitment**.
- Inputs: draft Product Requirement Document (PRD); personas; prototypes; legal / regulatory controls.

2.3 Activities & Tools

1. **Structured Walk-through** – facilitator steps through each requirement, reviewers mark *Clear* / *Ambiguous* / *Missing*.
2. **Checklist Inspection –** use IEEE 830 or your team's Definition of Ready (DoR).
3. **Rapid Prototyping** – low-fi wireframes let stakeholders "see" the requirement.
4. **Notion Comment Threads –** tag @owners on unclear phrasing.

2.4 Outputs

- **Approved Requirements List** with unique IDs in Notion.
- Issue log capturing open questions → tracked as GitHub Issues labeled `requirement-clarification`.

2.5 Key Knowledge

- Requirement **types**: functional, non-functional, constraint, assumption, dependency.
- Modeling notations: **UML Use-Case**, **BPMN** flow, **User Story** format.
- Quality attributes: completeness, consistency, atomicity, testability, feasibility.
- Acceptance-criteria formats: **Gherkin** (Given-When-Then), SMART checklists.

2.6 Essential Skills

- **Facilitation & active listening** – guide workshops, capture nuances accurately.
- **Critical questioning** – "why?", "what-if?" to uncover hidden needs & risks.
- **Conflict resolution –** mediate divergent stakeholder viewpoints.
- **Traceable documentation –** unique IDs, hyperlinking, change logs.

- **Domain analysis** – quickly learn and speak the client's business vocabulary.

3 Requirements Baseline & Change Control

3.1 Baseline Definition

A **baseline** is a **version-controlled snapshot** of requirements that serves as the contractual reference for scope, cost, and schedule.

3.2 Establishing the Baseline

1. **Version & Freeze** – export PRD to PDF or lock Notion page *PRD v1.0*; configure read-only permissions.
2. **Formal Sign-off** – digital signatures / approval property in Notion (e.g., `Status = Approved`).
3. **Traceability Matrix** – link each requirement ID → user story ID → test case.

3.3 Change-Control Workflow

Step	Owner	Tool	Notes
Submit Change Request	Stakeholder	GitHub Issue (template `change-request`)	Include business value & impact
Triage & Impact Analysis	Product Owner + Tech Lead	GitHub labels `needs-analysis`	Estimate cost, schedule delta
CCB Decision	Change Control Board	GitHub PR review or Notion vote	Approve / Reject / Defer
Baseline Update	Product Owner	Increment PRD to **v1.X**	Update Gantt & backlog accordingly

> Tip: Keep an audit trail—every approved change produces a new tagged release (e.g., PRD_v1.1).

3.4 Key Knowledge

- **Configuration management** principles (IEEE 828) & artifact versioning.
- SemVer vs. document revision numbering conventions.

- CCB charter: membership, quorum, decision authority & SLAs.
- Metrics: change-request cycle time, scope-creep percentage, baseline volatility index.

3.5 Essential Skills

- **Impact analysis** – quantify cost, schedule & risk per change.
- **Negotiation & trade-off** – balance stakeholder value vs. capacity.
- **Risk assessment** – classify changes (low / medium / high), apply schedule or budget buffers.
- **Tool proficiency** – Git branching for docs, Notion page history, e-signature workflows.

---

4 Agile Planning – Translating Baseline → Backlog

4.1 Process Overview

1. **Epic Breakdown** – group related requirements into *epics* that map to product capabilities.
2. **User Story Mapping** – arrange stories along a user journey; slice vertically by MVP → MMO (Minimal Marketable Offering).
3. **INVEST Filters** – each story must be *Independent, Negotiable, Valuable, Estimable, Small, Testable*.
4. **Relative Estimation** – use Planning Poker to assign **story points** (see §6.2).
5. **Sprint Planning** – pull stories whose total points ≤ (team velocity × sprint length).

4.2 Key Knowledge

- Story hierarchy: **Theme → Epic → Capability → Feature → Story → Task**.
- **Vertical slicing** methods: workflow step, data layer, persona path, ops slice.
- Capacity planning: focus factor (availability), holidays, training, buffer.

- Release forecasting: velocity trend × number of sprints → feature set delivered.
- Risk-adjusted backlog: spikes, architectural runway, enablers tagged & sized.

4.3 Essential Skills

- **Story writing** – user voice, value statement, SMART acceptance criteria.
- **Estimation facilitation** – run Planning Poker, manage anchoring & halo bias.
- **Backlog refinement** – split / merge stories, prune obsolete items.
- **Data analytics** – generate burn-up/down & cumulative-flow charts to spot bottlenecks.

---

5 Gantt Chart (v1) – Milestones + Sprints

5.1 When & Why to Use

- Communicate *external* deadlines (e.g., regulatory date) and *internal* sprint cadence in a single picture.
- Align cross-functional teams not immersed in Scrum terminology.

5.2 Building v1 in Notion

1. Duplicate **Notion Gantt Template**.
2. Create milestones: *Baseline Approved*, *MVP Launch*, *Beta Feedback Closed*, *GA*.
3. Add tasks for **Sprint 1–4** (two-week bars) underneath.
4. Draw dependency connectors: e.g., *Security Pen-Test* must finish before *GA*.
5. Insert a *buffer* (e.g., 10 % of timeline) after risk-heavy tasks.

5.3 Iterating the Chart

- After each **Sprint Review**, shift dates based on actual velocity.
- Use color coding (red/yellow/green) or emojis to flag slippages and critical tasks.

5.4 Key Knowledge

- Dependency types: **FS, SS, FF, SF**; how they drive critical path.
- Critical path method (CPM) & float calculation basics.
- Schedule baselines vs. forecast versions; variance reporting (SPI, SV).
- Buffer strategies: **critical-chain**, Monte Carlo P-value analysis.

5.5 Essential Skills

- **Schedule modeling** – convert backlog size & velocity into bars on the timeline.
- **What-if simulation** – adjust resource allocation, holidays, or scope to assess deadline feasibility.
- **Stakeholder storytelling** – present the timeline visually and narrate impacts clearly.
- **Integration tricks** – export PNG/PDF, embed in Confluence or email status updates.

---

6 Product Backlog & Story Points

6.1 Prioritization Techniques

Framework	How It Works	When Useful
MoSCoW	Must/Should/Could/Won't	Tight deadlines, simple weighting
WSJF (SAFe)	(Cost of Delay ÷ Duration)	Portfolio-level, economic focus
Kano	Delighters vs. Basic needs	UX-heavy products
RICE	Reach × Impact × Confidence ÷ Effort	Growth & product-led contexts
Cost of Delay	Quantify $ lost per time unit	Revenue-sensitive features

6.2 Assigning Story Points

- **Modified Fibonacci sequence**: 1, 2, 3, 5, 8, 13… promotes non-linear sizing.
- **Anchor Story –** pick a reference 3-point story everyone understands.

- Facilitate **Planning Poker**: conceal votes, reveal simultaneously, revote after discussion.
- **Velocity** = completed points / sprint; stabilize after 3–4 sprints for forecasting.

6.3 Backlog Hygiene

- Limit **work-in-progress (WIP)** by keeping only next 2 sprints "Ready".
- Weekly **Backlog Refinement** – clarify, split, de-duplicate >13-point stories.
- Archive items older than 6 months or with zero economic value.

6.4 Key Knowledge

- Estimation anti-patterns: *estimates as commitments*, *anchoring*, *t-shirt sizes stuck*.
- Velocity variance & throughput distribution for capacity planning.
- Definition of Done (DoD) vs. Definition of Ready (DoR).

6.5 Essential Skills

- **Consensus building** – moderate estimation disagreements to convergence.
- **Visualization** – maintain burn-down/up & cumulative-flow diagrams.
- **Technical-debt management** – surface, size, and schedule debt items.
- **Backlog pruning** – decisive removal of obsolete or low-value stories.

---

7 Tooling Workflow – Notion + GitHub

7.1 Notion (Requirements & Road-map)

- **PRD Database**: properties = ID, Title, Priority, Status, Epic Link, Owner.
- **Timeline View** for Gantt; **Board View** for Kanban snapshot.
- **Synced Blocks** to embed GitHub Issue status in Notion.

Key Knowledge

- Database **relations & roll-ups**; formula properties for auto-priority scoring.

- Notion **template buttons** – spin-up new story pages pre-populated.
- **Notion API** & Zapier for integrating with Slack or Jira.

Essential Skills

- **Schema design** – structure databases to eliminate duplicate entry.
- **Advanced filters & sorts** – craft stakeholder-specific dashboards.
- **Bulk edits & templates** – accelerate backlog creation & updates.

7.2 GitHub Issues & Projects (Execution)

1. **Enable Projects V2** – custom fields (`Story Points`, `Sprint`, `Risk`).
2. **Issue Templates**: *user-story.yml*, *bug.yml*, *change-request.yml* with YAML front-matter.
3. **Automation Rules**: when PR merged ➜ move card to *Done*, auto-add label `needs-qa`.
4. **Labels**: `epic`, `story`, `tech-debt`, `spike`, `blocked`.

Key Knowledge

- GitHub Actions for auto-labeling & Slack notifications.
- Linking commits/PRs to issues via keywords (`closes #123`).
- Board views: **table**, **board**, **timeline** for different audiences.

Essential Skills

- **Workflow automation** – triage new issues, enforce DoR through status checks.
- **Reporting** – export CSV, query with GitHub GraphQL for custom metrics.
- **Issue template authoring** – build checklists that embed DoD.

7.3 Cross-Linking Strategy

- Paste GitHub Issue URL in Notion to create a bi-directional backlink.
- Use `notion-doc` label in GitHub containing the Notion page URL.

7.4 Integration Patterns

- Webhooks/Zapier: sync status or create cross-system notifications.
- Embed GitHub Projects board inside Notion for single-pane tracking.

8 Activities

Activity	Tool	Outcome
Requirement Walk-through	Notion comments	Clarity rating per requirement
Baseline Sign-off Simulation	Notion approval property	PRD v1.0 frozen
Story Mapping Workshop	Miro or whiteboard	Draft user-journey map
Planning Poker	GitHub Issues	Story points assigned
Gantt v1 Build	Notion timeline	Shareable high-level plan

9 Glossary (Quick Reference)

- **Baseline** – Frozen approved requirements set (scope contract).
- **CCB** – Change Control Board governing baseline changes.
- **Backlog** – Ordered list of work items awaiting development.
- **Sprint** – Time-boxed iteration (usually 1–4 weeks) delivering a potentially shippable increment.
- **Velocity** – Average story points a team completes per sprint.
- **WSJF** – Weighted Shortest Job First; priority metric.

10 Reference Links

- **Notion Gantt Chart Template:** https://www.notion.com/templates/gantt-chart
- **Notion Product Requirement Document Guide:** https://www.notion.com/help/guides/building-a-product-requirement-document-in-notion

- **GitHub Issues Quick-Start:** https://docs.github.com/en/issues/tracking-your-work-with-issues/configuring-issues/quickstart
- **IEEE 830 Software Requirements Specification Overview:** https://en.wikipedia.org/wiki/Software_requirements_specification
- **INVEST User Story Criteria (Bill Wake):** https://xp123.com/articles/invest-in-good-stories-and-smart-tasks/
- **SAFe WSJF Explanation:** https://www.scaledagileframework.com/wsjf/