What is a team?

A group of people formed to achieve a goal
•May be temporary or indefinite
•Individuals share responsibility
•Takes advantage of all the collective talent, knowledge, and experience of each team member.
•Team members feel they are valuable to the project

Positive team behaviour

•Active listening
•Summarising
•Open body language, eye contact
•Encouragement
•Enhancing and maintaining the self-esteem of others.

Negative team behaviour

•Interrupting
•Silence
•Whispering to other members
•Aggression
•Ridicule
•Withdrawal, either mentally or physically, from the group
•Personal attacks

Characteristics of successful teams

Successful teams will demonstrate most, if not all, of the following characteristics:
•There is a clear specification and understanding of the purpose, objectives and goals to which they are working
•All team members work collaboratively
•Team members understand and agree on the role of each person
•Good chemistry is evident
•Individuals create connections that engender commitment, respect and responsibility
•Personal attacks

Requirements of successful teams

•Good communication
•Courage to confront and resolve conflict
•Ability to give positive feedback
•Ability to empathise with other team members
•Willingness to put aside personal goals in order to achieve team goals

Team types

Traditional

•Shared understanding and purpose
•Mutually agreed operating principles
•Interdependent – all working for the good of the team
•Distinguish task from process

Self-directed

•The team as a whole is responsible for whole product or process
•Team plans and performs work, including supervision and management
•A facilitator helps team get started and stay on track

Tuckman's Four Phase Model:

Forming – Storming – Norming - Performing

Forming stage

•Orientation •Identify boundaries -[Interpersonal & Task]
•Establishment of dependency relationships - [Team leaders, Team members, Standards – group norms]

Storming stage

•Conflict. •Polarisation around interpersonal issues. •Emotional response to task
•Resistance to group influence and task requirements

Norming stage

•Resistance is overcome. •Group feeling and cohesiveness develop

•New standards (group norms) develop. •New roles adopted. •Personal opinions are expressed

Performing stage

•Interpersonal structure becomes the tool of task activities
•Roles become flexible and functional. •Group energy is channeled into the task
•Structured issues have been resolved •Structure supports task performance

Belbin's Team Roles

•Roles may be filled by more than one person
•Each person may fill more than one role
•Team roles are not personality types but are clusters of characteristics

- People Orientated:
  - •Coordinator – sees big picture, confident, stable, recognizes abilities in others; good at delegation, clarifies decisions
  - •Team Worker – keeps team running, good listeners, smooth over conflicts and help people understand; often not noticed until they are absent; will not take sides so may not be able to take decisions
  - •Resource Investigator – focuses outside the team

- Action Orientated:
  - •Shaper – task focused with high motivation, keen to "win", committed to achieving goals
  - •Implementer – turns ideas into action; efficient and self-disciplined, deliver on time; loyal
  - •Completer Finisher – perfectionist, strong need for accuracy, has high standards, worries about minor details
- Thought Orientated:
  - •Shaper – task focused with high motivation, keen to "win", committed to achieving goals
  - •Implementer – turns ideas into action; efficient and self-disciplined, deliver on time; loyal
  - •Completer Finisher – perfectionist, strong need for accuracy, has high standards, worries about minor details

Benn and Sheat's Group Roles:

Identified 3 different types of roles:

- Task Roles: those centred on the task
- Personal Roles: personal / social
- Dysfunctional Roles: dysfunctional / individualistic

Flexibility of role play by members will improve performance
Not pessary to have all roles all the time
Play right role at right time

How to use Benne and Sheat's Group Roles when forming your team

- Determine what stage you are at in team formation and what roles are helpful for that stage
- Develop capacity in necessary and missing roles
- Flexibility of roles played by members will improve performance
- Identify dysfunctional roles
- Wherever possible eliminate this behavior
- Very important – these behaviours are disruptive and damaging
- Not necessary to have all the roles all the time

People skills

A set of skills enabling a person to get along with others, to communicate ideas effectively, to resolve conflicts, and to achieve personal or business goals.
People skills include:
•Communication

Understanding how people communicate/ Expressing your thoughts and feelings clearly/ Speaking up/ Asking for and giving feedback

- Collaboration
- Development and maintenance of productive relationships
- Leadership
- Motivation
- Influencing

Persuasion, articulation / Active listening / Multiple perspectives /Enquiry / advocacy, trust

- Effective decision-making

Goal-focus / Decision-making process / Environmental factors / Develop personal qualities of team-members / Stimulate team creativity / Manage opportunity and risk

Personality & Understanding its Impact

- **Team work** often **brings out** both the **best** and the **worst** of team members
- **Know**ing **your own preferences** helps you learn to work effectively with others helps you understand your own strengths and weaknesses
- **Know**ing **team member personalities and preferences** lets the team leverage personality differences and mitigate and manage inhibitor personalities, so as to achieve their common goal.
- High performing teams demonstrate high levels of "Emotional Intelligence" (Peter Salovey and Jack Mayer (1990); Daniel Goleman, (1990))
- "Emotional Intelligence" (EQ) is the ability to use emotions effectively
- Daniel Goleman, Working with Emotional Intelligence, estimates that **IQ accounts for only 4% to 25% of how well people perform at work** and that the **other 75%** to 96% left unexplained can be, largely, attributed to **emotional intelligence**.

Synergy

- a combined effect greater than the sum of their separate effects

1 + 1 + 1 = 10 (positive synergy) 1 + 1 + 1 = 2 (negative synergy)

Characteristics of high performing teams

Share a sense of **common purpose**
Make effective use of individual talents and expertise
Have balanced and shared roles
Maintain a problem solving focus
Accept differences of opinion and expression
Encourage risk taking and creativity
Sets high personal performance **standards**
Identify with the team

Conditions Favouring Development of High Performance Project Teams

Ten or fewer team members
Voluntary team membership
Continuous service on the team
Full-time assignment to the team
An organization culture of cooperation and trust
Members report only to the project manager
All relevant functional areas are represented on the team
The project has a compelling objective
Members are in speaking distance of each other

How to Manage different types of conflict within the Project Team

Encourage Functional Conflict Encourage dissent by asking tough

questions.
Bring in people with different points of view.
Designate someone to be a devil's advocate.
Ask the team to consider an unthinkable alternative

Manage Dysfunctional Conflict
Mediate the conflict.
Arbitrate the conflict.
Control the conflict.
Accept the conflict.
Eliminate the conflict.

Rejuvenating the Project Team

Informal Techniques

Institute new rituals./ Take an off-site break as a team from the project./ View an inspiration message or movie./ Have the project sponsor give a pep talk.

Formal Techniques

Hold a team building session facilitated by an outsider to clarify ownership issues affecting performance. / Engage in an outside activity that provides an intense common experience to promote social development of the team

Defining Characteristics of Leadership & Management [3]

Category	Leadership	Management
Thinking Process	Focuses on people Looks outward	Focuses on things Looks inward
Goal Setting	Articulates a vision Creates the future Sees the forest	Executes plans Improves the present Sees the trees
Employee Relations	Empowers Colleagues Trusts and develops	Controls Subordinates Directs and coordinates
Operation	Does the right things Creates change Serves subordinates	Does things right Manages change Serves superordinates
Governance	Uses influence Uses conflict Act decisively	Uses authority Avoids conflict Acts responsibly

Essential Management Skills

Management responsibilities are task oriented, therefore good project managers require

- **Technical skill** -
  knowledge about and proficiency in a specific type of work or activity
- **Human skills** -
  knowledge about and ability to work with people often considered "people" skills
- **Conceptual skill** -
  the ability to work with ideas and concepts and is focused on ideas

A Comparison of Management & Leadership Competencies [5]

Management produces order and consistence	Leadership produces change and movement
Planning and budgeting	Establishing direction
Establishing agendas	Creating a vision
Setting timetables	Clarifying the big picture
Allocating resources	Setting strategies
Organising and Staffing	Aligning People
Provide structure	Communicating goals
Making job placements	Seeking commitment
Establishing rules and procedures	Building team and coalitions
Controlling and Problem Solving	Motivating and Inspiring
Developing Incentives	Inspiring and energise
Generating creative solutions	Empowering subordinates
Taking corrective action	Satisfying unmet needs

A Comparison of Managing or Leading a Project [6]

Managing = coping with complexity	Leading = coping with change
Formulate plans and objects	Recognise the need to change to keep the project on track
Monitor results	Initiate change
Take corrective action	Provide direction and motivation
Expedite activites	Innovate and adapt as necessary
Solve technical problems	Integrate assigned resources
Serve as a peacemaker	
Make tradeoffs among time, costs, and project scope	

Elements of a Feasibility Study

Definition:

> *"An analysis to determine if a course of action is possible within the terms of reference of the project. Work carried out on a project or alternatives to provide a basis for deciding whether or not to proceed"*

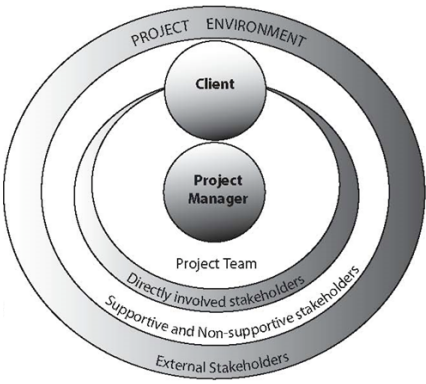
A feasibility study has six key elements:

1) Clear & well-understood definition of the scope of proposal
2) What is the current situation
   - Statement of the "world" as it is currently
3) Requirements
   - Statement of the problem
   - State of the world after project implemented
   - Constraints - project, organisational and external
4) Approach
   - Considers the options / various alternatives
   - Build versus buy / In-house versus contract
   - Explanation of why preferred option was selected
5) Evaluation
   - Cost-effectiveness of selected approach
   - Includes estimations for other potential approaches

High-level project schedule

6) Review & decision

Q: What does the term ‘stakeholder’ mean?

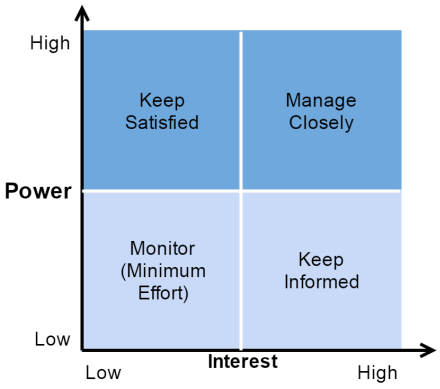


External Stakeholders
(who may not be directly involved)
- Regulatory authorities
- Unions
- Special interest groups – society at large
- Lobby groups
- Government agencies and media outlets
- Individual citizens

Directly involved:
- Originator
- Owner
- Sponsor
- Functional managers
- Contracts
- Suppliers
- Support companies
- Users
- Customers

Stakeholders Matter!

High power, interested people: these are the people you must fully engage and make the greatest efforts to satisfy.
High power, less interested people: put enough work in with these people to keep them satisfied, but not so much that they become bored with your message.
Low power, interested people: keep these people adequately informed, and talk to them to ensure that no major issues are arising. These people can often be very helpful with the detail of your project.
Low power, less interested people: again, monitor these people, but do not bore them with excessive communication



Project Success Factors

To have an **increased likelihood of success** it is important to have the following key factors in place from the start of the project:
Clear, well-understood project objective(s):
- What are the client's needs?
  - They are often high level and strategic
- Constraints
  - Internal
  - External
- Factors within and outside PM control
- Strong business case
  - Why are we doing this project?
- Sufficient planning detail
  - Agreed deliverables
  - Realistic timescale
  - Accurate cost estimates
  - Enables progress to be measured
- Motivated and committed team
- Controlling the scope
- Good risk management
- Well defined, measurable critical success factors agreed with the client
  - Helps determine whether a project should be cancelled

It is often very ***difficult to determine project success*** (***or otherwise***) on completion. For example, a project may have been undertaken to save a company money – but it may be some time, before the company can tell.

Product Vs. Project Scope

Project scope: The work that must be accomplished to deliver the product scope. It is sometimes also called the Statement of Work (SoW)
Product scope: The features and functions characterising a desired product, service or result. It is the outcome of the project

What is Project Scope?

A definition of the end result or mission of the project—a product or service for the client/ customer—in specific, tangible, and measurable terms

Purpose of the scope statement

To clearly define the deliverable(s) for the end user.
To focus the project on successful completion of its goals.
To be used by the project owner and participants as a planning tool and for measuring project success.

Project Scope – Checklist

Project objective - what, when, cost
Deliverables - progressive
Milestones - usually align with deliverables; control points
Technical requirements - product performance constraints
Limits and exclusions - explicit out-of-scope items
Reviews with customer - agree on scope

Project Scope – Terms and Definitions

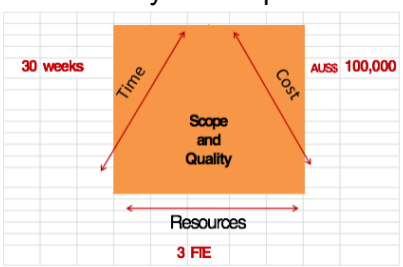
- Scope Statements
- Also called **statements of work (SOW)**
- Also called **Project Charter**
  Can contain a brief or expanded version of scope statement
  A document authorizing the project manager to initiate and lead the project.
- **Scope Creep**
  The tendency for the project scope to expand over time due to changing requirements, specifications, and priorities

Project Scope – establishing priorities [1]

Remember the iron triangle!
When determining the project scope it is important to determine priorities because they will likely lead to project trade-offs. These are caused by shifts in the relative importance of criteria related to cost, time, and performance parameters.
When managing the priorities of potential project trade-offs there are three possibilities:

Constrain: a parameter is a fixed constraint that must be satisfied.
Enhance: a parameter that would be preferentially optimised ahead of others if there is opportunity.
Accept: a parameter for which failure to satisfy is acceptable in a tradeoff situation.

		units		ó		Effect
Scope	100%		*	0.00%		100%
Quality	100%		*	0.00%		100%
Time	30	weeks	+	0		30
Resources	3	FTE	+	0		3
Cost	100,000	Aus$	*	1		100000



When developing the **communication plan** you need to consider the following **factors**:

- What information needs to be collected and when?
- Who will receive the information?
- What methods will be used to gather and store information?
- What are the limits, if any, on who has access to certain kinds of information?
- When will the information be communicated?
- How will it be communicated?

Additionally, you need to consider the sorts of information that you might need to communicate to stakeholders.
This will include:

- Project status reports
- Deliverable issues
- Changes in scope
- Team status meetings
- Gating decisions
- Accepted request changes
- Action items
- Milestone reports

To develop the communication plan you **must:**

- Conduct a **stakeholder analysis** so you can determine their information needs
- Decide the sources of required information
- Determine appropriate dissemination modes and
- Determine who has responsibility and what the timing will be

What is the Project Charter?

It is team documentation that defines:
the purpose of the team, how it will work, and, what the expected outcomes are.
It captures and makes public the agreements between team members. The aim is to:
make sure that all involved are clear about where they're heading, and to give direction when times get tough.

Outlines
Purpose
Objectives including critical success factors
Scope: Major milestones / deliverables / statement of work

Provides
Baselined and agreed point of reference
Preliminary Costs / Schedule / Resources

Includes
Assumptions
Constraints
Requirements / specifications / interface

*This content is alternatively called a **team charter,** as **project charter** sometimes refers to a scope statement or an authorisation for project resources, including the team.*

Why is the project charter important?

The project charter is developed through consensus early in the project, and helps the team “get off on the right foot.” It:

- Speeds the process of forming, storming, norming and performing, so that the team is likely to become effective much more quickly.
- Works towards developing team norms relating to acceptable behaviour and performance
- Helps develop a culture of trust, respect and support

The project charter is developed through consensus early in the project, and helps the team “get off on the right foot.” It:

- Usually posted in a highly visible location to ensure that team members can use it as a reference.

- Should include a definition of what team norms are, for:
  performance,
  attendance, and
  conduct.
- Provides guidance for team behaviour and performance throughout the life of the project

During the Project:

- Serves as a contract between the team and the sponsor
- Defines objectives and intent of the team - assures a common objective among team members
- Defines the work effort and its intended results to the rest of the program - avoids redundancy and "holes“
- Keeps the team focused - allows the team to determine if its activity is relevant and on-track or off on a tangent.
- Defines boundary conditions and helps the team determine when to raise an issue
- Helps control scope of team's efforts and re-negotiate its objectives or boundary conditions

3 basic types of software development:

Traditional (Waterfall) Incremental Integrative

Traditional (Waterfall)

Inflexibility and rework risks only beneficial / justifiable in developing software systems such as:

- systems embedded in hardware
- safety and security critical systems
- large, multi-developer systems

Avoid using traditional development where teams can communicate informally and requirements change quickly.

Incremental

Three major advantages over traditional development:

- reduced cost of implementing requirements changes
- better customer engagement and feedback
- early delivery of usable software

But…

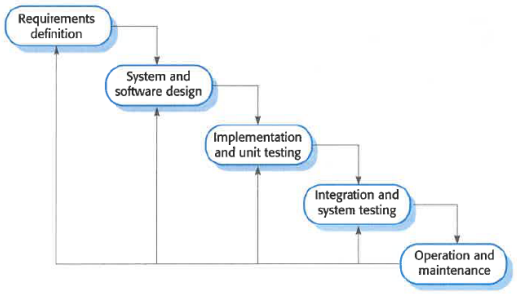
Process can be less visible - more developing and less documenting

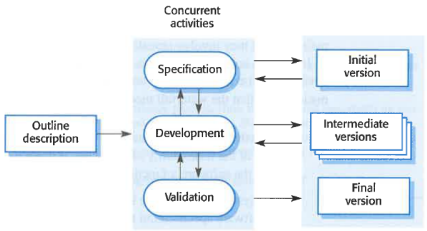
and System structure can get 'messy' so regular refactoring is needed

Integrative

Best for:

- configuring stand-alone multi-purpose applications
- collections of objects developed as a component or package for integration into a framework
- web services developed in accordance with standards for invoking over the Internet





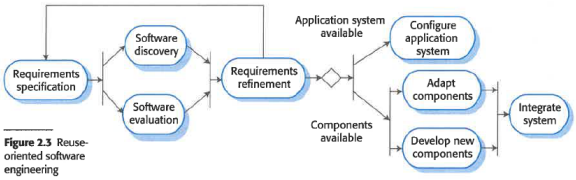


Figure 2.3 Reuse-oriented software engineering

Principles of APM

- Focus on customer value
- Iterative and incremental delivery
- Experimentation and adaptation
- Self-organisation
- Continuous improvement

APM is related to the **rolling wave planning** and **scheduling project methodology**.

It uses **iterations** ("time boxes") to develop a **workable product** that **satisfies** the customer and other key stakeholders. Stakeholders and customers review progress and re-evaluate priorities to ensure alignment with customer needs and company goals.

APM Methodologies

APM Methodologies view a **project as a non-linear, complex adaptive system** where change is normal and take a "barely sufficient" approach to plans, process and control while focussing on delivering customer value.

The **focus** of APM is on **people and their interactions**, giving individuals the power to make quick decisions. They are adaptive rather than predictive, even self-adapting their own processes.

APM are **chaodic** – that is, there is both **chaos and order.**

Product goals are achievable but they are not predictable.

Processes aid consistency, they are not repeatable.

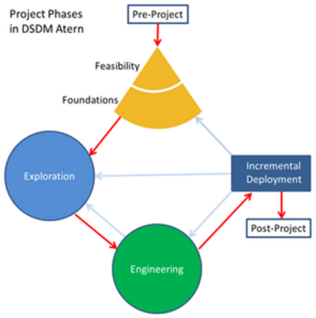
APM values collaboration and barely sufficient methodology, which is based on practices (**what happens in reality**) **not** processes (**what is described in manuals**).

Small releases – helps manage complexity, provides early feedback. 1 to 3 months

Iterative and incremental development – plans, requirements, design, code and tests are evolved incrementally through multiple passes or iterations. Iterations are fixed length (usually 2 weeks), which maximises feedback. The fixed scope retains stability.
Collocation – all team members, including on-site customer, are collocated. This arrangement facilitates communication & integration, encourages impromptu meetings, and design sessions.
Release plan / feature backlog – desired features are defined at a high level and prioritised by the customer. Estimation is done collaboratively in a release planning game (based on game theory). Developers provide effort estimates & customers decide the business priority.
Iteration plan/task backlog – high-level features from the release plan are elaborated upon and prioritised along with their implementation tasks in an iteration plan or task backlog. Estimation is done collaboratively with developers in an iteration planning game. Developers provide effort estimates & customers decide business priority.
Tracking – features and tasks are tracked within an iteration. They only count as complete when they are 100% done. There is no concept of partial completion. What constitutes "done" is agreed before project starts.
Self-organising teams – team members self-organise by completing tasks collaboratively from backlogs without top-down management control.
Simple, lean and adaptable – all aspects of work, including processes, are kept simple, lean (low on waste) and adaptable to maximise customer value and to accommodate change.

Popular Agile Methodologies:

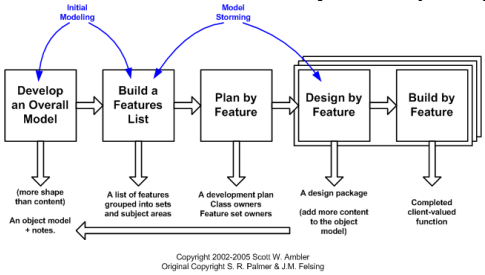
Dynamic Systems Development Method (DSDM)



Lean Development (LD)

Adapted from manufacturing; by seven principles:
Eliminate waste
Amplify learning
Decide as late as possible
Deliver as fast as possible
Empower the team
Build integrity in
See the whole

Feature-Driven Development (FDD)



Extreme Programming (XP)

Rational Unified Process (RUP)

Inception - Idea for project stated.
Development team determines if project worth pursuing / resources needed.
Elaboration - Project's architecture and required resources further evaluated.
Construction - Project developed and completed.
Transition - Software released to the public.

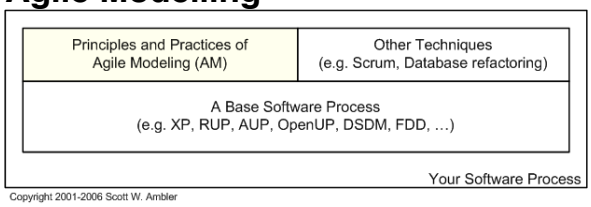
Adaptive Software Development (ASD):

speculate, collaborate, and learn cycles

Rapid Product Development (PRD)

- cross-discipline seminars & impromptu meetings, entire engineering team understands all functions
- streamlined processes, flexibility in quality control systems and paperless approval systems
- robust library of reusable software code

Agile Modelling



Agile PM in Action - Scrum Methodology

Is a holistic approach for use by a **cross-functional team** collaborating to develop a **new product**. It defines product features as deliverables and prioritizes them by their perceived highest value to the customer. Priorities are re-evaluated after each iteration (sprint) to produce fully functional features. Scrum consists of four iterative phases: (1)analysis (2)design (3)build (4)test
1sprint = 1 - 4 weeks

Agile Project Management: Key Scrum Concepts

- outcomes(The way a thing turns out; a consequence.) vs outputs(The amount of something produced by a person, machine, or industry) :compare
- documenting the scope -- the product backlog: control workflow & issues
- agile requirements using user stories: origin, (who, what, why),(3C:Card, Conversation, Confirmation), generate acceptance criteria, big picture, template zombies
- using a visual mapping tool to provide an overview of the total scope of the project: user story map, planning with story map

Scrum concept help move the process from one of the requirements delivery to one of requirement discovery:

FALLACY - Requirements delivery process

•The customer knows what she/he wants
•The developers know how to build it
•Nothing will change along the way

REALITY - Requirements *discovery* process

•The customer discovers what he/she really wants (not what he/she first thinks he/she wants)
•The developers discover how best to build it
•There is acceptance there will be many changes along the way.

In APM: the scope of the project is captured in the "Product Backlog"
The agile product backlog isa prioritized features list, containing short descriptions of all functionality desired in the product.
Unlike many traditionally managed projects, agile projects don't start a project with a lengthy, upfront effort to document all requirements. Instead, an agile team and its product owner typically begin by writing down everything they can think of to include in the backlog. A typical Scrum backlog 4 types of items:
(1)*Features.* (2)*Bugs*(*no diff btw new feature*).
(3)*Technical work.* ("Upgrade all developers' workstations to Windows 10.")
(4)*Knowledge acquisition* (*"Scrum backlog item about researching various JavaScript libraries and making a selection.")*
As a [Who] I want [What] because [Why]. ===> user story == requirements <Less details, no how; no why == no value>

Controlling work flow with Product Backlog

- The **product owner** shows up at the **sprint planning meeting** with the **prioritized agile product backlog** and **describes the top items** to the team.
- The **team then determines which items they can complete** during the coming sprint.
- The team then moves items from the product backlog to the sprint backlog. In doing so, they expand each Scrum product backlog item into one or more sprint backlog tasks so they can more effectively share work during the sprint.
- Conceptually, the **team starts at the top of the prioritized Scrum backlog** and draws a line after the lowest of the high-priority items they feel they can complete. In practice, it is not unusual to see a team select, for example, the top five items and then two items from lower on the list that are associated with the initial five.

Product Backlog Issues

1. Long lists of “stuff to do” - flat structure.
2. No better than poorly developed WBS / PBS – that **focus on delivering the items on the list rather than delivering value.**
3. A list of stories **without any connection to the value** that is to be delivered to users who are trying to solve problems.
4. Arranging user stories in the order you'll build them **doesn't help to explain to others what the system does**.
5. You need context in order to really tell a story about the system. BUT typical product backlogs have no context.
6. The typical project has a minimum of dozens of user stories, frequently over a hundred. **Stepping through each one** making an in-out decision is **tedious and time-consuming**.

Backlog Description	Initial Estimate	Adjustment Factor	Adjusted Estimate	work remaining until completion						
				1	2	3	4	5	6	7
Title Import				256	209	193	140	140	140	140
Project selection or new	3	0.2	3.6	3.6	0	0	0	0	0	0
Template backlog for new projects	2	0.2	2.4	2.4	0	0	0	0	0	0
Create product backlog worksheet with formatting	3	0.2	3.6	3.6	0	0	0	0	0	0
Create sprint backlog worksheet with formatting	3	0.2	3.6	3.6	0	0	0	0	0	0
Display tree view of product backlog, releases, sprints	2	0.2	2.4	2.4	0	0	0	0	0	0
Sprint-1	13	0.2	15.6	16	0	0	0	0	0	0
Create a new window containing product backlog template	3	0.2	3.6	3.6	3.6	0	0	0	0	0
Create a new window containing sprint backlog template	2	0.2	2.4	2.4	2.4	0	0	0	0	0
Burndown window of product backlog	5	0.2	6	6	6	0	0	0	0	0
Burndown window of sprint backlog	1	0.2	1.2	1.2	1.2	0	0	0	0	0
Display tree view of product backlog, releases, prints	2	0.2	2.4	2.4	2.4	0	0	0	0	0
Display burndown for selected sprint or release	3	0.2	3.6	3.6	3.6	0	0	0	0	0
Sprint-2	16	0.2	19.2	19	19	1.2	0	0	0	0
Automatic recalculating of values and totals	3	0.2	3.6	3.6	3.6	3.6	0	0	0	0
As changes are made to backlog in secondary window, update burndown graph on main page	2	0.2	2.4	2.4	2.4	2.4	0	0	0	0
Hide/automatic redisplay of burndown window	3	0.2	3.6	3.6	3.6	3.6	0	0	0	0
Insert Sprint capability ... adds summing Sprint row	2	0.2	2.4	2.4	2.4	2.4	0	0	0	0
Insert Release capability ... adds summary row for backlog in Sprint	1	0.2	1.2	1.2	1.2	1.2	0	0	0	0
Owner/assigned capability and columns optional	2	0.2	2.4	2.4	2.4	2.4	0	0	0	0
Print burndown graphs	1	0.2	1.2	1.2	1.2	1.2	0	0	0	0
Sprint-3	14	0.2	16.8	17	17	17	0	0	0	0
Duplicate incomplete backlog without affecting totals	5	0.2	6	6	6	6	6	6	6	6
Note capability	6	0.2	7.2	7.2	7.2	7.2	7.2	7.2	7.2	7.2
What-if release capability on burndown graph	15	0.2	18	18	18	18	18	18	18	18
Trend capability on burndown server	2	0.2	2.4	2.4	2.4	2.4	2.4	2.4	2.4	2.4
Publish facility for entire project, publishing it as HTML web pages	11	0.2	13.2	0	0	13	13	13	13	13
Future Sprints	39	0.2	46.8	34	34	47	47	47	47	47
Release-1				85	70	65	47	47	47	47

Agile Story

- Shared understanding and alignment are the objectives of collaborative work
- User stories are supposed to spark conversations.
- Stories get their name from how they should be used, not what should be written!
- A story is supposed to be a promise to have a conversation around the subject of the story, to eliminate misunderstandings through speech, not writing.
- Users stories help develop a shared understanding between the customer and the development team and should be focused on the big picture. As stories are developed they are talked about, discussed and documented. They provide a light-weight approach to managing requirements for a system. The details are figured out in future conversations between the team and the product owner or customers.
- This approach facilitates just-in-time requirements gathering, analysis and design by the following activities:
- -Slicing user stories down in release planning

- -Tasking user stories out in sprint planning
- -Specifying acceptance test criteria for user stories early in development.

When decomposing user stories following the INVEST guideline will help you create good user stories.

I – Independent - The user story should be self-contained, in a way that there is no inherent dependency on another user story.

N – Negotiable - User stories, up until they are done, are part of an iteration (Sprint in Scrum), can always be changed and rewritten.

V – Valuable - A user story must deliver value to the end user.

E – Estimable - You must always be able to estimate the size of a user story.

S - Sized appropriately - User stories should not be so big as to become impossible to plan/task/prioritise with some certainty

T – Testable - The user story or its related description must provide the necessary information to make test development possible.

Planning with user story map

- Activities are the **backbone**
  The essential activities required to deliver minimum viable product (MVP)
  Do not prioritise: Without any of these you do not have an MVP
- The **walking skeleton** consists of
  The highest priority tasks immediately below the backbone
  The smallest possible system that provides end to end functionality

Prioritize tasks below the backbone: Move activities up or down

- Releases can be marked out in “swim lanes”
  Stories can be moved in or out of releases
  Stories can have different heights within a release
- Build all the major features a little at a time: Each release **always** adds value
- **Test** by walk/talk through, left to right;

Benefits of User Story Maps

- big picture in your backlog,
- better tool for making decisions about grooming and prioritizing the backlog
- promote brainstorming and a collaborative approach to generating user stories
- encourage an iterative development approach where early deliveries validate your architecture and solution
- a great visual alternative to traditional project plans
- a useful model for discussing and managing scope
- visual dimension to planning, & real options for your project/product
- Display as an info radiator : good for sprint or iteration planning, mark of progress

Activities are the **backbone**

The essential activities required to deliver **minimum viable product (MVP)**

Do not prioritise - without any of these you do not have an MVP

The walking skeleton consists of

The highest priority tasks immediately below the backbone

The smallest possible system that provides end to end functionality

Tips to help minimise some of the issues

•The right stakeholders need to be involved in problem definition.
•Stakeholders need to clearly understand their operational/business needs.
•Stakeholders need to **define the problem**, not the solution.
•**Solution and delivery constraints** need to be real, applicable and negotiable.
•**Development strategies** (the selected life cycle model/s) should cater for the evolving nature of business and product requirements.
•Progressive/evolving understanding of a required outcome/technical solution need is the norm.
•**Effective project management** is **not static** – it needs to be disciplined and agile
•It is important to **focus on the entire scope of change** to the business that is affected by the project.
•The **initial plan** is only good for a **limited time** – things always change.
•As **things change during a project**, information has to be gathered, decisions made, actions taken and implementation effectiveness monitored.
•**Accountabilities and authorities** for decision making and product development should be established prior to the commencement of development work.
•**Measures of success** should be defined prior to commencement of development work.
•A **consistent and coherent approach** should be applied to all **technical and management activities** on complex and high-risk development work.
•**Proof of quality** should be sought and delivered incrementally throughout a development activity.

The Agile Project Plan

Exists at multiple levels:

Product Vision
Product Road Map
Product Backlog
Iteration Backlog (also called sprint backlog)
Daily task planning.

Agile consider ‘fractal’ because the “phases” in an agile project are nested within one another

Phases of the Agile Project Management Lifecycle – Initiating Project Initiation Phase

An agile project begins with the following, all of which are what identified in the PMBOK as the "Project Charter" and the "Preliminary Scope Statement"

• Visioning and development of the product roadmap
• Planning = the **evolution of the product over time**
• Product backlog definition -- the broad scope of the project

The initiation phase in an agile project also includes the following activities (many of which are also completed during initiation in a traditionally managed project)

• Setting up the environment
• Documenting the architecture
• Release planning

• Team coding standards
• Definition of “done”

Agile initiation **may also include development of the business case & feasibility study**. Sometimes this is considered to be outside the boundaries of an agile project, and so either not done at all -- frequently to the detriment of the project -- or is done prior to project commencement and the gathering of the team.

Phases of the Agile Project Management Lifecycle – Product Release

Consists of one or more iterations
- Iterations = PMBoK Sub-Phases

Begins with Release Planning
- One or two day event
- Involves whole team, including customer
- Output = Release Plan = high level STRATEGIC plan

Goals, assumptions, and decisions to guide the team so that it delivers value to the customer

Ends with Release Retrospective

Inputs to release planning
- Prioritised and estimated product backlog
- Backlog = PMBoK User Requirements
- Team’s velocity
  - Velocity = rate at which a team completes work (user stories) and turns them into running, tested and documented features
- Product vision & road map

Phases of an Agile Project – Agile Iteration

Iteration = one **timebox** of work; Timebox has a fixed start & end date

Iterations are usually between 1 & 6 weeks long

Managed by the team: Consists of daily tasks for individual team members

Begins with a planning meeting with customer
- The team as a whole agrees what features need to / can be delivered
- The team breaks the features/functions/activities into tasks
- There is a clear commitment by team to delivery

Each agile iteration begins with Iteration planning
- Inputs and attendees as for release planning
- Output = detailed task list = tactical guide
- Requires negotiation and collaboration with customer!
- Team manages work within an iteration
- Iteration plan updated to reflect reality

Iteration Review

Collaborative decision making about the product
- Review progress
- Walkthrough of the completed work as well as uncompleted work
- May review iteration metrics

Iteration Retrospective

Decisions flow into next iteration
- What when well
- What needs improvement

Updated priorities for backlog

Phases of an Agile Project – Daily Work

Lowest level of the “agile fractal”

Involves daily ‘stand up’ meetings (15 minutes) for progress updates

Team collaborates to manage, direct, and
complete its work

Interim Phase – Releases

The interim phases or fractals of the agile project management lifecycle are known as "releases".
Agile releases are essentially project milestones within a traditionally managed project.
The principal difference, however, is that **agile project ALWAYS delivers a working (set of) feature(s)**.
Within each release, there are a series of “iterations".

Final Phase -- The Project Retrospective (= PMBoK Closing process)

The final phase in an agile project is the project retrospective, which equals the project ‘post-mortem’ in a traditional project.
The project retrospective involves the entire project management team and the key stakeholders, including customers. The purpose of the retrospective is to reflect on the complete project with the aim of improving management processes for next project.

Scheduling

‘The process used to determine the overall project duration and when activities and events are planned to happen. This includes identification of activities and their logical dependencies, and estimating of activity durations, taking into account requirements and availability of resources.’

Schedule Development

Determining planned start and end dates for project activities
Subject to change and frequent revision
Revisions improve accuracy, account for the changes
Schedules commonly have *time-based constraints*. These might be contractual obligations, government legislation or directives, weather or climatic conditions, imposed or agreed milestones

Important concept - Effort versus Duration

Effort
The number of labour units required to complete a schedule activity or work breakdown structure component. Usually expressed as staff hours/days/weeks.
Duration
The total number of work periods (not including holidays or weekends) required to complete a schedule activity or WBS component. Usually expressed as workdays or work-weeks.
Converting Effort to Duration
Planning involves estimating the effort required and then converting it to or calculating the duration required. We do not estimate duration

If a project team member tells you it will take 100 hours of effort to complete an activity, and she has two people who are available 30 hours a week, how many days’ duration will it take?
100 hours/(2*30) = 1.67 weeks or 8.35 days

Product Owner: manage vision, the ROI, release
Scrum Master: manage the process
Team: manage the development iteration

Your *roadmap* needs to answer these questions:
- What are the relative priorities of each theme?
- When do we intend to work on each theme?
- How many team members are working on each theme?

Use the map to manage your budget: High risk, deal first

Slice through functionality

Use the story map to create a development plan by "slicing" horizontally through the stories that represent the functionality.

First slice

The first slice shows the functionality working from end-to-end. This first slice doesn't work in all needed situations and if shipped users would be upset, but it allows real data to be used to see how it performs and helps the team learn about technical risk

Second slice

The second slice helps build functionality and gets closer to being releasable. It enables to the team to learn things that they couldn't predict and helps identify things that have been overlooked, that weren't explored in the prototype.

This slice starts to indicate how well the system performs in reality so that the team can understand the "**predictably unpredictable**", that is things you can't know, but you know will exist.

Later slices

Later slices help refine the picture and make it polished. These slices add in some of the unpredictable things.

Agile planning takes a "**just-in-time**" approach to developing details.
As a result, user activities (software features) are captured in a largely imprecise manner.
Only enough detail is known about them at any point during the development process to allow the tasks to be completed at that time.

What does it mean to be "ready"?

Feature(s) to be developed in sprint iteration need to be:
•Immediately actionable by the team
•Product backlog is designed to be a negotiation between the product owner and the team – so talk it through before planning the sprint
fully meets the INVEST

A story is not "ready" if it depends on something outside the team's control.
Such stories **greatly increase the risk of an iteration failing** to achieve its goal, and you can't do anything about it!

Development 'done-ness': Well-written code, Documentation complete
Acceptance criteria 'done-ness': User can request a password reminder

Done, DoD:

The **DoD may vary** depending on whether it relates to a **story**, an **iteration**, a **release** or **the whole product**
Establishing "done" up front can save hours of refactoring work
No non Done story in iteration meeting
Done is about quality not approval from external parties

Functional Decomposition: break down large process

User story Size:

Business > User/Customer > Development

When to use *Epic* not user story?

•undertaking long-term planning;
•during initial estimation as part of a business case or project initiation, when a perfect, very precise estimate is not required. Instead, a high-level estimate, provided through a high-level product backlog, is sufficient, where big user stories – epics – describe large swaths of functionality; and
•before you are ready to develop them -- **remember just-in-time decomposition is the way to go.**

Epic(big story) + Conversation = multiple Smaller story + More conversation = more even smaller stories

How to break down epics into stories

Often splitting stories **separate high-value parts from low-value parts**, allowing the team time to focus on the valuable parts of a feature. This helps to provide value by enabling development of a minimal, end-to-end solution and then filling in the rest of the solution. Frequently splitting stories is called decomposition.
Decomposition takes practice and experience. Some stories are more difficult than others to split. Discovery conversation is one of the best tools for breaking down big stories.

Remember to only decompose epics only when more detailed estimation is required

Flowchart for Splitting Epics

•Prepare the input story
•Apply the splitting pattern
•Evaluate the split

Functional Decomposing Nine Patterns for Splitting Epics

1. Workflow steps
Build a simple end to end case first, then add the middle steps and special cases.
For example, As a content manager, I can publish a news story to the corporate website.

2. Business rule variations
As a user, I can search for flights with flexible dates.
...as “n days between x and y.”

3. Major effort
Where a story has several parts, but most effort goes to the first one.
For example, splitting a story by credit card type: most effort goes to creating the first one

4. Simple / Complex
Capture the simplest version along with some acceptance criteria.
Then add variations and complexities into their own stories. For example,
As a user, I can search for flights between two destinations.

5. Variations in data
Build the simplest first. For example,
As a content manager, I can create news stories.

6. Data entry methods
Complexity is often in the UI, not the story. For example,
As a user, I can search for flights between two destinations.

7. Defer performance
Start with the easiest and provide value. Make it work, make it fast.
As a user, I can search for flights between two destinations.

8. Operations
Often stories begin with the word “manage”. For example
As a user, I can manage my account.

9. Break out a “spike”
When the implementation is poorly understood a time-boxed spike to resolve uncertainty around implementation is a good approach.

How many stories is enough?

Teams need to understand how stories contribute to building and realising business goals and objectives and delivering value to stakeholders.
Stories need to be prioritised by more than just the user’s perspective. Don't overlook business, compliance, and technology.
Stories should only be written within view of the planning horizon -- the next development cycle.
Backlog grooming lets teams refine a smaller number of stories shortly before they are needed. This approach leads to a team being more efficient and their estimates more accurate.

Too many stories leads to

Bloated backlogs
Extra work (possibly mis-work) on stories
Team taking its eyes off business value
Loss of focus on the user

The **conversation** starts out at a **high level**: a who-what-why discussion.

At this point in the discussion, a **decision to go ahead** with further investigation is made; is this a **feature/product** that is **worth spending time** on? This is called discovery.
Discovery isn't about building software. It's about building a **deeper understanding** of **what we could build**.

To develop quality, "ready" stories

in addition to the usual product owner you should aim to include a tester, a member of the core product discovery team and a user experience designer in your story-writing workshops.

Persona

an example of your target user assembled from the facts (and sometimes assumptions) you have about the users.

client / developer relationship

like the good patient-doctor one where you tell the doctor what hurts and the doctor works out the best way to treat the problem as it is discovered.

Your job is to take the **vision of the business stakeholde**r and **help** them make it a success. Sometimes this may mean telling those stakeholders things they don't want to hear.

Working with stories is a **continuous process of *conversation* and *discussion*** to break them down from big things to small things.

During these conversations the **focus must be kept on what is being built, for whom and why**.

Story mapping: helps the decomposition process while keeping the focus of the conversation on the people who will use the product and what makes those people successful.

No matter how "ready" your stories appear, the conversation needs to keep going even when you're building because no matter how good they are, you won't have predicted everything you'll learn once you start to build.

For each release, develop an Opening-, Mid-, and Endgame Strategy

Opening game

- Focus on the essential
- Step through the entire product
- Focus on things that are technically challenging or risky
- Skip optional things
- Skip sophisticated business rules
- Build just enough to see the product working end-to-end

Midgame

- Fill in and round out features
- Add stuff that supports optional steps users might take
- Implement tough business rules
- Start testing product end-to-end things like performance, scalability and usability
- Think about quality concerns
- Constantly test them

Endgame

Refine your release
Make it sexier, more efficientUsing it with real data and at scale will help find improvement opportunities that you couldn’t see from a prototypeObtain feedback from users that you can apply

Project
A project is a sequence of finite dependent activities whose successful completion results in the delivery of the expected business value that validated doing the project.

Program
A collection of related projects, which may need to be completed in a specific order for the program to be complete. Programs may have more than one goal.

Portfolio
A collection of projects that share a common link to one another.

Software development models

The waterfall model
This takes the fundamental process activities of specification, development, validation, and evolution and represents them as separate process phases such as requirements specification, software design, implementation, and testing.

Incremental development
This approach interleaves the activities of specification, development and validation. The system is developed as a series of versions (increments), with each version adding functionality to the previous version.

Integration and configuration
This approach relies on the availability of reusable components or systems. The system development process focuses on configuring these components for use in a new setting and integrating them into a system.

Defining Characteristics of Leadership & Management [3]

Category	Leadership	Management
Thinking Process	Focuses on people Looks outward	Focuses on things Looks inward
Goal Setting	Articulates a vision Creates the future Sees the forest	Executes plans Improves the present Sees the trees
Employee Relations	Empowers Colleagues Trusts and develops	Controls Subordinates Directs and coordinates
Operation	Does the right things Creates change Serves subordinates	Does things right Manages change Serves superordinates
Governance	Uses influence Uses conflict Act decisively	Uses authority Avoids conflict Acts responsibly

Tuckman's Four Phase Model

Forming stage

- Orientation
- Identify boundaries
  - Interpersonal
  - Task
- Establishment of dependency relationships
  - Team leaders
  - Team members
  - Standards – group norms

Storming stage

- Conflict
- Polarisation around interpersonal issues
- Emotional response to task
- Resistance to group influence and task requirements

Norming stage

- Resistance is overcome
- Group feeling and cohesiveness develop
- New standards (group norms) develop
- New roles adopted
- Personal opinions are expressed

Performing stage

- Interpersonal structure becomes the tool of task activities
- Roles become flexible and functional
- Group energy is channelled into the task
- Structural issues have been resolved
- Structure supports task performance

PESTLE(external):

Political, Economic, Sociological, Technological, Legal, Environment

Project Scope – establishing priorities

iron triangle!
Determining the project scope: determine priorities because they will likely lead to project trade-offs. These are caused by shifts in the relative importance of criteria related to cost, time, and performance parameters. When managing the priorities of potential project trade-offs there are three possibilities:
Constrain: a parameter is a fixed constraint that must be satisfied.
Enhance: a parameter that would be preferentially optimised ahead of others if there is opportunity.

Accept: a parameter for which failure to satisfy is acceptable in a tradeoff situation.

Plan end-to-end experiences NOT features

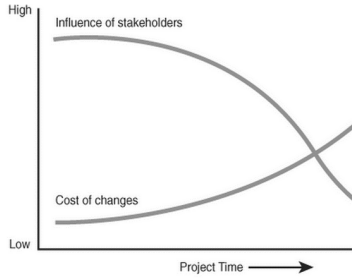
All they want is a smooth seamless solution.
Features are part of the product development process: they are not part of what users see as the solution to a problem.Developers need to have a **good understanding of user needs** so they can stitch together features to provide end-to-end solutions. **requires insight into the customers motivations and overall situation**.
Big picture first, details later

So what is an end-to-end experience?

End-to-end experience is what the customer sees, feels, and does when he or she uses your product, device, or service in a real-life situation, from the very beginning to the very end.
Examples:

- **The total purchase experience from an online App store.**
  This includes how the app is discovered, chosen, and then made available to use on the customer's devices.
  If the user/customer sees it as one connected job – or really wishes it could be all connected – then the development team should be thinking that way too.
- **Buying a milkshake to drink on your way to work**
  In this case, the end-to-end experience is identified by understanding when the majority of sales were made -- on the way to work. This meant the customer needed something that could be handled single handed, no mess, and the contents needed stave off hunger pangs.

Taking a customer focused iterative approach generally facilitates achieving this goal. This can be achieved by: Using a fast feedback cycle: Prototyping ideas, Experimenting, Learning fast, Iterating often



Measurable Organisation Value (MOV)

The **Measurable Organisational Value (MOV)** guides all decisions & processes and should be clearly tied to overall mission, goals and strategy of the organisation. MOV forms the basis for evaluating the project's achievements.
The MOV must be: clear, attainable and specific and measurable
Achievement of the project objective must be easily recognisable
(the tangible end product that the project will deliver)
Helps identify assumptions
Need to be evaluated in terms of criticality to the project
Significant source of risk to a project
Process is as important as the plan itself

Minimum Viable Product (MVP/MVS)

A great way to remember what the MVP should be is M U R F S

Specifically Marketable Useful Releasable Feature Set

the smallest product release that successfully achieves its desired outcomes
the smallest solution release that successfully achieves its desired outcomes

At this point, we can't measure outcomes to determine whether we will meet them once the software is delivered.

Prior to the release, we're just hypothesising. You're guessing who will buy your product, what are their characteristics, and if they decide to use it what's feasible to build within the required timeframe. You have to guess how much (or how little) will make them happy.

If you hedge your bets too much and guess high, you may have spent too much money and risk not getting everything done. On the other hand, if you go to low (that is less than minimal) you fail because you haven't delivered something that will meet their needs.

User scenarios Vs. User Stories

User stories are simple, and written from the perspective of the end user.

User scenarios are richer descriptions of the user story. They take into consideration the motivation of the user and the environment within which the user story lives.

> **User scenarios are a valuable tool that helps you envision your solution**.
> Use these scenarios to develop prototypes and test out your ideas with the users to learn whether your solution is valuable and usable, from their point of view.
> **Be especially careful of what people say they want. Frequently, what really matters are the things they and you haven't thought about.**
> It is this that gives rise to the concept of change in software development projects. The project and the desired outcomes haven't changed, but the complete problem has not been fully understood.

Opening game

- Focus on the essential
- Step through the entire product
- Focus on things that are technically challenging or risky
- Skip optional things
- Skip sophisticated business rules
- Build just enough to see the product working end-to-end

Midgame

- Fill in and round out features
- Add stuff that supports optional steps users might take
- Implement tough business rules
- Start testing product end-to-end things like performance, scalability and usability
- Think about quality concerns
- Constantly test them

Endgame

- Refine your release
- Make it sexier, more efficient
- Using it with real data and at scale will help find improvement opportunities that you couldn't see from a prototype
- Obtain feedback from users that you can apply

Estimation supports and enables good decision-making

During initiation of the project -

During initiation it is **used to determine how long the project should take** and its **likely cost**, which is used to determine whether the project is worth doing. This information can also be used to help organisations **determine cash flow needs**.

Project underway -

1. Once the project is underway, estimation is used by the team to help to **determine how well the project is progressing**, to **develop time-phased budgets** and **establish** the **project baseline**.
2. Estimates of time and resources required facilitate the project manager's **scheduling of project work** and **enable selection of appropriate scope** for the next release and iteration (sprint).
3. The process of estimation promotes discussion and collaboration among team members which leads to team buy-in and commitment and builds trust and increases acceptance of responsibility by individual team members.

Estimation is hard

- Estimation is an art, not a science. Your estimates will never be "correct", but they will get better as you revisit and refine your estimates throughout the life of the project.
- The greater your experience with estimation the better your estimates will become.
- Many project teams encounter real problems when estimating. The most significant is lack consistency between individual estimates.
- One person may be **optimistic** while another may be **pessimistic**, but we may not know who is who.
- Teams often get the person who will be responsible for the work to estimate the duration and effort.
- But, the person might provide a pessimistic estimation so that they will always be able to meet the proposed deadline.
- Using more than one person to do the estimation can help alleviate this issue.

Problems with traditional approaches to estimation:

Large-scale and detailed estimation before a great deal is known about the project

Project leads do the estimating:

- they may have more experience so they don't allow for less experienced team members
- they are motivated to minimise estimations to bring in the project in less time with less $
- they may not have enough understanding of the project to give a meaningful answer at the start of the project and underestimate the effort required for the specifics

So just from a logic point of view, using a model where the technical team lead and/or project manager estimates time is not the best.

- ***Project manager*** is asked to provide a quick "ballpark" estimation of time required to complete the project which then becomes a commitment despite informing management that it is only a rough estimate.
- The uncertainty of scope, technology and the team itself, all work together to increase the difficulty of estimation.

Barry Boehm's 'Cone of Uncertainty'

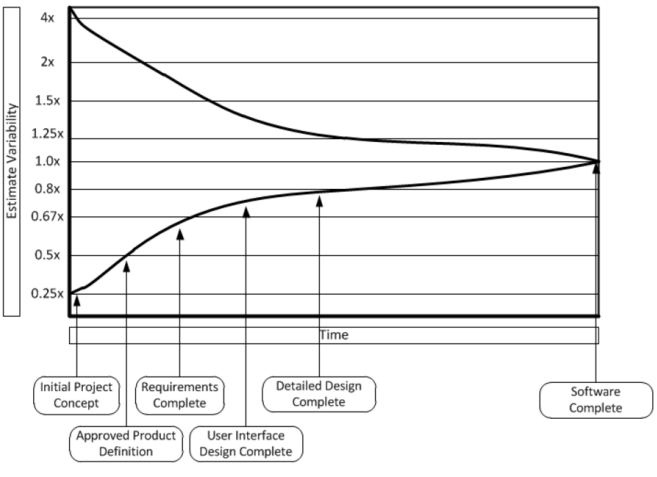
Minimise uncertainty with:

✓ 'walking skeleton'

✓ Increased frequency of estimation feedback

✓ Near term not distant future estimation

Start of project – least is known, but precision is requested a 12 months project could be estimated at anywhere from 3 – 48 months!



Elapsed time (or duration)

the amount of time that passes on the clock to do "something"

example -- International soccer game = 120 minutes

If you plan on only spending 120 minutes to watch the game, you will miss the end of the game, just how much you miss will depend on the individual game. The game appears to last longer because while the game itself is 120 minutes, additional time is added to this to take account of time-out for injuries, substitutions, etc. And don't forget the extra time added for TV advertisements

Velocity

the measure of the team's rate of progress. the amount of work that a team can complete during an iteration, that is how quickly a team can create working features.

Relative units of measure

Story points: unit of measure used to express the overall size of a user story, feature or other piece of work.

The value of a story point is related to how hard it is (risk, complexity, and definition of done) and how much of it there is (effort). **It is NOT related to the amount of time or the number of people.** Raw value is unimportant – what matters are the relative values!

Fibonacci sequence: Relative units of measure are unit-less but numerically- meaningful. Humans intuitively understand relative sizes of things. *Story points are additive, time is not.*

Ideal time / ideal days

Ideal time is the amount of time it takes to do "something" when it is stripped of all peripheral activities. It is a pure estimation of the effort required. Estimating in ideal days ignores overheads such as phone calls, dealing with emails, attending meetings, holidays, etc. and **assumes** that the **individual is working solely on that task** rather than dealing with multiple tasks (which is more likely).

When estimating ideal days you need to assume:

the story being estimated is the ONLY thing you'll work on everything you need will be on hand and will be immediately available; there will be no interruptions

Ideal time is not additive – it is still a ***relative unit*** of measure

Remember the law of diminishing returns. Vary the effort allocated to estimation according to the purpose of the estimate -- all you need is adequate results!

Derive your estimates collaboratively rather than doing them on your own as they will be much more accurate.

Approaches to Estimation

Some are used more often in traditional project management while others are used more with agile approaches to project management. There is not one right way.

Expert opinion --

used more frequently with traditional rather than APM
can be individual or group
relies on "gut feel" based on (extensive) experience
Disadvantage for APM - Need to consider all aspects of developing a story, so one expert will likely not be enough

Analogy --

estimator compares the story to an assortment of other stories which have already been estimated - known as triangulation
a little bit bigger than '3' and a little bit smaller than '8'
NOT against a single baseline

Disaggregation --

by splitting the feature/project into smaller features and estimating each

need to make sure you don't miss any tasks, which it is very easy to do

sanity check - does the sum of the parts make sense?

Planning Poker --

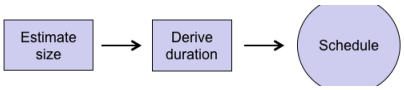
combines expert opinion, analogy, and disaggregation

enjoyable

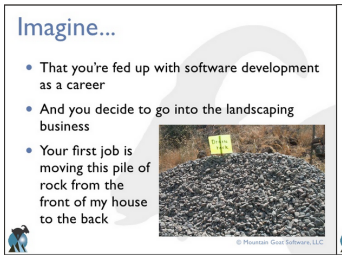
produces quick and reliable estimates

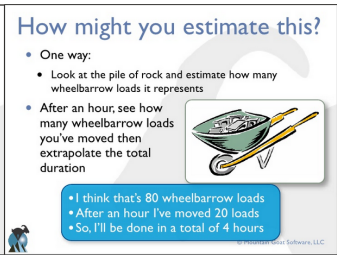
The **goal of estimation** is not to derive an infallible estimate but to achieve a valid estimate cheaply.
The goal is **not absolute precision** but reasonableness.

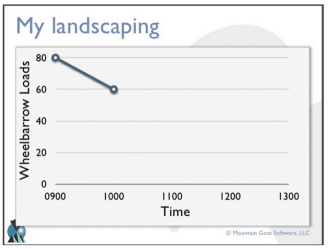
Estimate size NOT duration



It is important to remember when using relative estimation that: Size does NOT equal duration and Size is relative
In other words, we estimate the size of the task from which we derive the duration and it is the duration that we use to determine the schedule.







Story points

Story points are an **abstract unit of measure** which **indicate the size** of a story. The team determines how story points translate to effort and establish their own fixed comparison of points.
A story point is an **amalgamation** of the

- amount of **effort** involved in developing the story
- the **complexity** of the story and its development
- the **risk** involved
- the **quality** required (the definition of "done")

Size is relative. Story points are relative.
Estimating size using story points is relatively straight forward. There are two approaches:
The smallest story = 1 story point

OR A medium sized story = (somewhere in the middle of the range you expect to use) perhaps 5

- Once the team has decided upon the **baseline story and its size**, each new story is compared with this story and its size in story points is decided. When estimating, the team decides whether a particular story is *larger* or *smaller* than the *baseline story*.
- The number of story points allocated to a story indicates the overall size of that story and the number of story points a team can deliver per iteration is called team velocity.
- Size is relative. Story points are relative.

Depending when (and why) you are estimating, stories can be incomplete. Sometimes you really need more information BUT you're required to have an estimate for each story.
In this instance, you may have to make assumptions. Take a guess BUT note it down! Sometimes

when a team has difficulty using story points, because they can't visualise the size, they may use t-shirt sizes or "dog points". A dog point represents the height of the dog at the shoulder.

Estimation using Ideal days

- If you really must use time, then use the notion of ideal days, that is using **effort** rather than **elapsed time (or duration)**. Ideal time equals the amount of time a task takes when we strip away all peripheral activities such as "overheads".
- If you must use time, it is easier and more accurate to estimate in **ideal time** rather **duration (elapsed time)**. However, you must remember that ideal time will vary for different levels of staff, e.g. a manager versus a developer.
- Estimating using time is risky. What exactly do you mean when you answer "5 days" to the question "how long will this take?". Most non-agile practitioners will assume “elapsed time” which is very very different!
- Estimation using ideal days is based on a number of assumptions
  - the story is the only thing you will work on
  - everything is available at the start and
  - there will be no interruptions.
- As a measure of size, ideal deals do not consider overheads. However, they can be converted using velocity (just as with story points) into an estimate of duration.

Choosing between Story Points and Ideal Days

Each approach has advantages and disadvantages. Considerations favouring story points

1. Story points help drive **cross-functional behaviour**
2. Story point estimates do not **decay**
   1. Ideal days can change based on a team's experience, technology, domain etc.
   2. A programmer starting out with a new language will take longer than when that same programmer has experience
3. Story points are a **pure measure of size**
   1. Cannot be compared with reality (actual days) as often happens with ideal days.
4. My ideal days are not your ideal days
   1. Often have to compare the "time it takes"
   2. Need to have developers who are all of approximately the same skill level
5. **Ideal days are easier to explain** to people outside the team
   1. They have an intuitive feel -- "this is how long it would take if I worked on nothing but it"
   2. Need to explain the "cone of uncertainty" and progressive refinement of planning/ estimation accuracy
6. At first, **ideal days are easier to estimate**
   1. For new developers, ideal days are easier to get started with
   2. However, usually within an hour, teams arrive naturally at estimating with story points

Key advantages of using story points to estimate size

Forces the use of *relative estimating* -- studies have shown we are better at this than estimating time (Lederer and Prasad, 1998. A Causal Model for Software Cost Estimating Error and Vicinanza et al., 1991. Software Effort Estimation: An Exploratory Study of Expert Performance)
Focuses on *estimating size*, not duration -- we derive duration empirically by seeing how much we complete per iteration
Puts estimates in units that can be added together -- time-based estimates are not additive

How to play…Planning Poker

1. The team (estimators) are given a deck of cards. Each card has a valid value (according to the units of measure that the team is using) that represents an estimate.

2. The customer/product owner reads a story and it is discussed until each member of the team (estimators) believes they fully understand what is involved in its development.
3. Each estimator selects a card according to his estimate of size / complexity
4. When each estimator indicates they have selected a card, all cards are turned over at the one time.
5. Differences, especially outliers, are discussed until the team again believes they fully understand what is required to develop the story.
6. Re-estimate until convergence is reached.

Hints for playing planning poker

Don't move on until there is agreement.

Reason:Disagreement is valuable and the discussion needed to help the team move towards agreement helps them gain understanding of the various positions that each team member is taking, and importantly why they think differently.

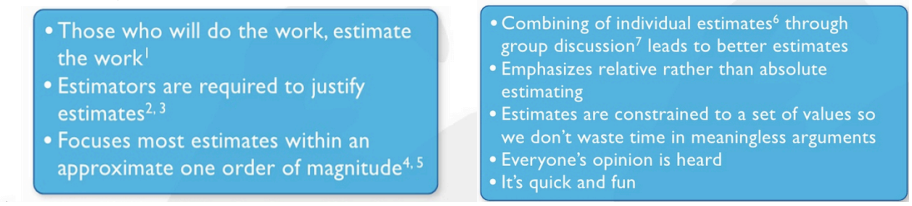
Don't average estimations **– come to consensus**.

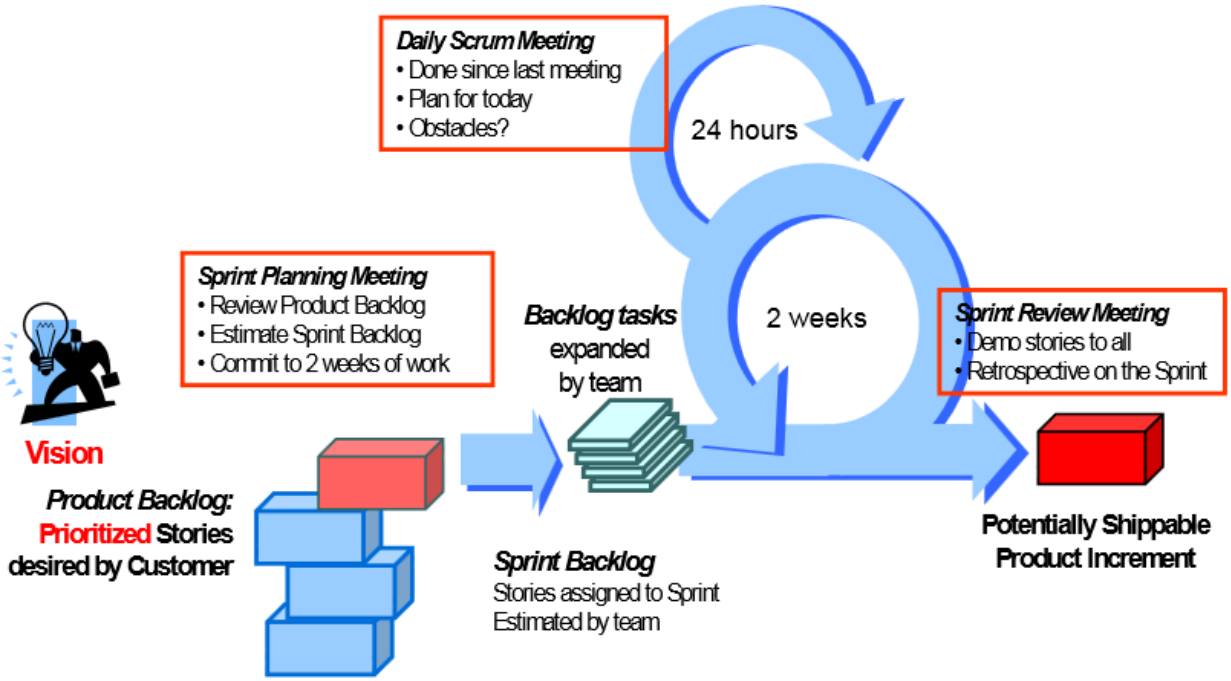
Reason: Averaging is too easy. Consensus requires a robust discussion – that is one of the most significant benefits of planning poker. If you must average, take the median value (middle value) not the mean. Teams lose out when they average. Some agreement will be stronger than others – "close enough" rather than the "perfect number"

When should planning poker be used?

- Planning poker is ideally suited to estimating the product backlog. It is often used shortly after the initial Product Backlog is created.
- Estimation workshops may take multiple days and are used to create initial estimates for scoping and sizing the project.
- Planning poker should be used at least once per iteration because stories are added throughout the project. This is often done a few days before the end of an iteration, immediately following the daily standup meeting.

Planning Poker works because…





Velocity

Velocity = Story points per iteration

The notion of velocity is based on the assumption that if a team completed 10 story points last iteration, then they'll complete 10 again this and subsequent iterations.

Velocity is a **measure** of the team's **rate of progress** and is **calculated** by **summing the total number of story points** that have been **completed during the iteration**.

To determine the total size estimate for the project, sum the story point estimates for each story. Using the team's velocity we can then estimate the number of iterations required. Duration is calculated by mapping the number of iterations required onto a calendar.

Estimate size but Derive duration. Using relative estimation and velocity to derive duration means that planning errors are self-correcting.

Only **re-estimate** when it is believed that a story's relative size has changed

-- we have more knowledge about it than we had when we did the original estimation.

Approaches to calculating velocity

There are three principal options when calculating velocity. These are to

1. Use historical values
2. Run an iteration
3. Make a forecast

While all of these approaches may be appropriate at different times, you have little chance of being "correct".

It is, therefore, best to express velocity as a range.

Estimating velocity using historical values

- There are two principal issues related to using this approach.
- Firstly, you can't use the approach if you don't have the historical data; and secondly, what about when there has been significant change? This changes includes changes related to
- Technology. Domain. Team. Product owner. Tools. Working environment. People estimating
- This approach is best when the same team is moving to a new release of the same product. Because of uncertainty you may need to provide a larger range to reflect that uncertainty. Calculating velocity using historical values is a simple calculation.
- ***For example:*** In the last release the team completed 150 story points and it was comprised of 10 iterations. Therefore, Velocity = 15 points, (i.e. 150 / 10 = 15)

Note: factor in the cone of uncertainty you were introduced to earlier in the course

Estimating velocity by running an iteration

- The **ideal way to forecast velocity is to run one (or three) iterations.**
- This should always be the default approach. Hold off giving an estimate for one iteration and measure the velocity for that iteration, then calculate the range around that one data point by multiplying by 0.60 and 1.6.
- It is best to run three or more iterations before giving an estimate of velocity as this provides additional options for calculating range.
- Simple approach:
  - ***Example1:*** Imagine you've completed 3 iterations with velocities of 12, 15 and 16. The range therefore is 12 to 16
  - ***Example2:*** Calculate average velocity for the iterations you have run, in this case, it is 14.3. Round it up to 15. (NEVER round down!) For each completed iteration, move one step to the right on the cone of uncertainty, to a maximum of 4 steps (Product Design Specification). To calculate the range, multiply the average by the number on the y axis representing the step.
    - Three iterations with an average velocity of 20. For 3 iterations range is 85% to 115%.

Iterations completed	Low multiplier	High multiplier
1	0.6	1.6
2	0.8	1.25
3	0.85	1.15
4	0.9	1.10

Estimating velocity by making a forecast

- Use this approach when **you don't have historical data** and **it isn't feasible to run a few iterations** to observe velocity. For example, when a project will start in 12 months, or won't start until the contract is signed. Any estimate in these situations reflects considerable uncertainty.
- Expand user stories into constituent tasks, as when planning an iteration, work out what fits in an iteration and then calculate velocity.
- Estimate available hours per person per day.
- Determine total hours per iteration.
- Select some (arbitrary & random) stories and expand to constituent tasks to fill iteration. Convert velocity into a range.
- **Remember – available hours do not equal hours worked per day**

Which approach should I use?
Circumstances will regularly determine your choice!
In order of desirability

1. Run one or more iterations.
   Actual data is always the best choice.
2. Use actual velocity from a related project by this team.
3. Estimate velocity by seeing what fits.

Regardless of which approach you use to start, switch to using actual, observed values for velocity as soon as possible.
Always give a range that reflects the uncertainty inherent in the estimate.
Once the project is underway and you have observed velocity always use those metrics when discussing likely range of completion dates
Words of wisdom
"Remember it is better to be roughly right than precisely wrong"
John Maynard Keynes

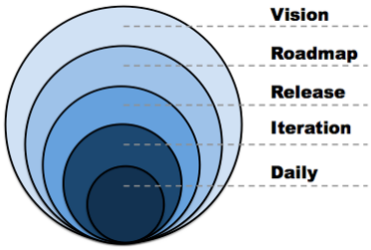
Selecting an Appropriate Iteration Length

Iteration lengths

- An iteration is usually between two and four weeks long. Occasionally they may be longer, or shorter.
- There is no one length is right in all cases -- the right length for a team on one project may not be the right length for the same team on another project!
- While iteration lengths may vary between teams and between projects, the iteration length usually remains constant throughout the project, though they may be different for different teams on the same project (but that is unusual).

Factors in selecting an iteration length

There are a number of factors which influence the choice of iteration length. These include



• the length of the release being worked on
• the amount of uncertainty involved
• the ease of getting feedback
• how long priorities can remain unchanged
• the willingness of the team to go without outside feedback (relates to uncertainty and risk)
• the overheads associated with iteration and
• how soon a feeling of urgency is established

1. Length of the release being worked on

Short (small) projects benefit from short iterations because there is always a need to measure and to show progress. Another factor is how often priorities can be refined, which is usually done between iterations. **Most projects benefit from having four or five opportunities to change course and assess progress.**

2. The amount of uncertainty

When thinking about this aspect, you need to understand how well does
the customer know what they want?
the team understand the customer requirements?
the team know itself and its velocity?
Shorter iterations provide greater feedback allowing the team to
Measure progress
Gather feedback from stakeholders
Confirm course

3. The ease of getting feedback

It is important to choose the iteration length that maximises feedback to the whole team. To do so you need to consider how much feedback is required, the frequency and timeliness. Affecting this consideration is whether the project is an internal or an external project
Your aim should always be to maximise value received from feedback

4. How long priorities can remain unchanged

Remember that priorities are not changed ***during*** an iteration only ***between*** iterations. Ask yourself, how long does it take to turn a good idea into working software? Usually, at an absolute minimum it is one and a half iterations. So consider how certain you are that

• you're on the right track,
• you and the customer understand what they really need
• you understand the risk involved
• you understand the level of uncertainty surrounding the project.

5. Willingness to go without outside feedback

No matter how good a team, without outside feedback the work could be useless. Poor communication may cause misunderstandings as may infrequent communication.

6. The overhead of iterating

Each iteration costs (usually in terms of time) so consider what can be achieved within an iteration. For example, regression testing must be completed for EACH iteration!

7. How soon a feeling of urgency is established

The deadline needs to be soon enough to motivate but not so soon as to create stress. Equally you don't want to go so long that your iterations fall into the pattern of having a clear beginning, middle and end.

Deciding on an iteration length

An appropriate iteration length is one that encourages everyone to work at a consistent pace throughout the whole iteration

- If it is **too long**
  - you find there is often a feeling of slack at the start with stress and overtime at the end to meet the deadline
- **Extremely short** – one week –
  - it never lets a team recover from leave (sick or holiday)
- **Longer** – four weeks
  - Longer iteration lengths allow teams to be more creative and can be appropriate for an exploratory phase.
  - However, longer iterations have a distinct feeling of beginning, middle and end. Longer iterations allow the team to be more relaxed (perhaps too much so), just right, or very pressured. Team and organisational needs will determine which.
- **Ideal?** – two week
  - Planning and testing overheads are manageable

BUT: Iterating constantly can be stressful

The deadline is never further away than next week!

To overcome this you can use a **macro-cycle**: 6 x 2 + 1

Six 2-week iterations followed by a 1-week iteration

The team chooses its own work during the 1 week iteration

- They focus on what they view as priority to the project, e.g. refactoring, automating legacy manual tests, additional analysis
- It is NOT a dumping ground for sloppy work from earlier iterations
- It allows the team to work off technical debt from earlier iterations

Issues for project execution

Project execution always faces a number of significant issues. These issues include

- the evolving nature of requirements and understanding
- the complexity of business needs and solutions
- the criticality of technology to business success
- the uncertainty of component maturity/quality
- the uncertainty of staff/resources availability
- the variability of supplier capability and workmanship
- the expense of failure or delay – business and financial
- the expense of maintenance/ total cost of ownership
- the cheapest bid often wins

It is possible to minimise the impact of these issues

For example, **ensuring that the right stakeholders are involved from the start**, especially during problem definition. Stakeholders need to clearly understand their own operational/business needs.

It is important that **stakeholders define the problem**, rather than attempting to define the solution. When considering possible solutions and delivery constraints, these must be **real, applicable and negotiable.**

Development strategies (the selected lifecycle model/s) should cater for the evolving nature of business and product requirements because a **progressive and evolving understanding of a required outcome or technical solution need is the norm**.
Effective project management is not static – regardless of whether traditional or agile approaches have been adopted, the **project management itself needs to be disciplined and agile**.
It is important to **focus on the entire scope of change to the business** that is affected by the project. As things always change, the **initial plan is only good for a limited time**.
In response to change during a project, information has to be gathered, decisions made, actions taken, and implementation effectiveness monitored.
Prior to the commencement of the development work, **accountabilities and authorities for decision making and product development should be established**. **Measures of success** should also be defined prior to the commencement of development work.
On complex and high-risk development work, a **consistent and coherent approach** should be applied to all **technical and management activities**.
Proof of quality should be sought and delivered incrementally throughout a development activity.

Some project failure statistics

Mike Cohn (2005) in Agile Estimating and Planning (p. 11)1 notes the following statistics (citations for those statistics here are from Cohn1): nearly two-thirds of projects significantly overrun their cost estimates 64% of the features included in products are rarely or never used the average project exceeds its schedule by 100% (Planning often ignores uncertainty and estimates become commitments.)

Why planning fails

- Much of the **planning is done by activity rather than by features**, which means that when an activity finishes late, that **delay is passed down as the activities are not independent**.
- Requiring team members to multitask (work on multiple projects) slows productivity even further. Furthermore, features that are of the greatest importance and priority are not always developed first, meaning the **client has to wait longer to receive value**.

Scheduling

- According to the Association of Project Management "scheduling is the process used to determine the overall project duration and when activities and events are planned to happen.
- This includes **identification of activities and their logical dependencies**, and **estimation of activity durations**, taking into account requirements and availability of resources."
- Developing the schedule involves determining the planned start and end dates for project activities. The schedule will be revised a number of times during the project to improve accuracy and to account for changes. Schedules commonly have time-based constraints, such **as contractual obligations, government legislation or directives, weather or climatic conditions, imposed or agreed milestones**.

The Schedule -- AKA Release Planning

The purpose of release planning is to **answer questions** such as:

- How much will be done by 30 June?
- When can we ship this set of features?
- How many people or teams should be on this project?

When all teams have completed the last iteration for the release, the full product set of functionality (the next product increment) is released to the client.
Releases can be internal, or to customers. **Planning doesn't stop**, it continues in anticipation of the next release.

Planning the release

- Progress is measured by the delivery of working software, i.e. concrete evidence that the team is progressing.

- The schedule is developed using prioritised requirements or features, i.e. the prioritised product backlog. Scheduling a project is done at a high level of abstraction of user stories.
- Don't forget to include activities such as training in the schedule and make sure you choose an approach which reflects your environment
- A **Release Plan is a high-level plan that covers multiple iterations**, 3 to 12 (or even more) iterations. Exactly how many will depend on how long each iteration is and how long the overall project will run. Each release covers many months, usually somewhere between 3 to 6 months.
- Release planning is an iterative process and is **updated at the start of each release**.

Timing:
Release planning happens after initial approval of the project and then again at the start of every release cycle

Who is involved:
The Product owner, all delivery teams, the system architect, and any other experts, including subject matter experts.

Preparations:
Epics on the backlog are inspected by the participants in the release planning meeting. When necessary new stories are written and epics decomposed. Stories to be included in the release are estimated and the delivery team familiarizes with themselves with the stories.

What:
The highest priority stories are then moved to the best iteration within the release that will allow the release to meet its goal.

Approaches:
There are **two principal approaches** to developing a release plan.
The first is to start with an end date and see how much can be completed within that time frame.
The second is to take a set of user stories and see how long it will take to develop them. Which approach is better at any time will depend on the project context and the constraints under which it is operating.

Release planning helps you:
- decide WHAT must be developed for the product to be releasable – i.e. the MVP.
- determine HOW LONG development will take
- convey expectations to management
- with strategic planning

Furthermore, release planning provides a baseline against which to measure progress.
It also provides context – showing how each iteration leads towards the whole.

The Release Plan
When planning the release, the stories (epics) to be included must be **assessed against the overall organisational goals** - as set out in the agreed MVP.
For example:
Will it make required money for the company?
Will it save enough money?
Will it capture large enough market share? If not would a longer or shorter project work?
The release plan **does not indicate**
- which developers work on which stories or tasks,
- engineering tasks to be performed

REMEMBER: **User stories are simply descriptions of functionality to be delivered**

The release plan **may communicate**
- Some underlying assumptions / additional information
  - Who is on the team
  - Length of iteration

Date first iteration starts / date last iteration finishes

Agile Metrics

Velocity is a measure of a team's rate of progress. Velocity is calculated by summing the number of story points assigned to each user story that the team completed during the iteration. It is assumed that in future iterations, the team will produce at the rate of their past average velocity. In other words, we base our estimation on "**Yesterday's weather**".

REMEMBER

When calculating velocity always remember the difference between effort and duration and that you estimate effort and calculate duration.

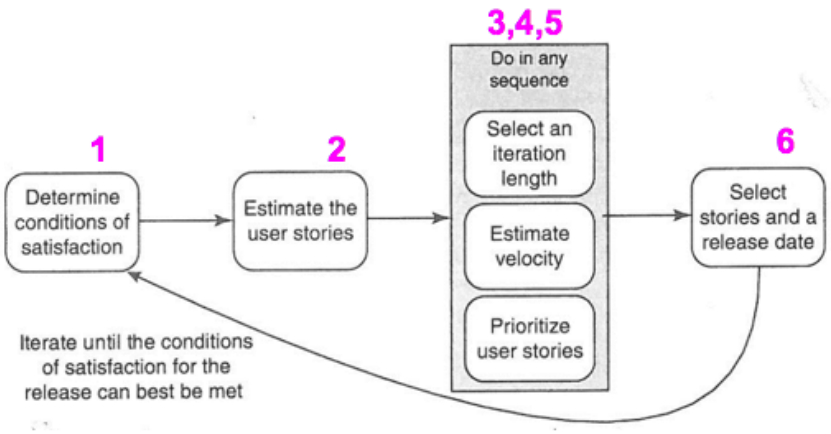
Effort is the number of labour units required to complete a scheduled activity, eg a user story or work breakdown structure (WBS) component. Effort is usually expressed in terms of staff hours/ days/weeks.

Duration is the total number of work periods (not including holidays or weekends) required to complete a scheduled activity, eg a user story or WBS component. Duration is usually expressed in terms of workdays or work weeks or sprints of known length.

Converting Effort to Duration: Example

If a project team member tells you it will take 100 hours of effort to complete an activity, and she has two people who are available 30 hours a week, how many days' duration will it take? Remember that effort does not equal duration. If the working day is 8 hours, productivity is unlikely to be greater than 6 hours out of 8. When calculating duration always factor in non-productive time. This will vary according to role; a project manager will have greater "non-productive" time than a programmer who is engaged on the project 100% of the time.

100 hours / 2 = 50 hours work per person
50 / 30 = 1.67 weeks or 8.35 days
PLUS 25% non-productive time = 2 days
Duration is greater than 2 weeks' work per individual



Step 1 - Determine the conditions of satisfaction

- What are the criteria you will use to evaluate success or failure against? These are often a combination of schedule, scope, and resource goals
- Projects are either date or feature driven
    - Date = must be delivered by a set date
    - Feature = must include a set of features (which are more important than the actual date or release, which is usually as soon as possible)

Step 2 – Estimate the user stories

- Cost of developing each story
- Prioritise

    Estimate only features that are likely to be included in THIS release

Step 3 – Select an iteration length

- Usually between 2 - 4 weeks
- Occasionally longer or shorter
- Depends on size of project, the team, organisation, work to be completed etc

Step 4 – Prioritise user stories - MVP

- Most projects have too many features
- Impossible to do all, so do those that deliver the most value
- Sequencing is important

Step 5 – Select user stories and a release date

- Using estimate of team's velocity per iteration and assumption of how many iterations plan a release that meets conditions of satisfaction.
    - Feature driven
        - Sum the estimates of needed features and divide by expected velocity
        - This will give you the number of iterations to complete desired functionality
    - Date driven
        - Determine number of iterations based on the calendar
        - Multiply the number of iterations by expected velocity
        - This will give you the number of story points in a release

Baseline your release plan

The baseline is your promise. It helps to keep rhythm and momentum and makes sure the team can commit to delivering the release. Each release usually contains a number of features.

Remember: As the project progresses, always compare your estimates with your actuals. *If you fail to do this your estimates will never improve in accuracy and you will never become a better estimator!*

Wisdom = Experience + Reflection

Scrum iterations are Sprints

Iterations (or Sprints) have consistent durations throughout the project. They are a “time-box” during which a complete (or “Done”), usable, and potentially releasable product increment is created.
The next iteration commences immediately following the conclusion of the preceding iteration.

Iteration VS. Release Planning

Release planning

The release plan looks forward through the product for some months out from the start

Iteration planning

The iteration plan looks ahead ONLY for the period of the iteration.
During iteration planning user stories are decomposed into tasks and estimation is in hours rather than story points. Iteration planning helps to refine understanding and leads to discussion about the product and the software design.

What is the goal of Agile iteration planning?

When planning the iteration, the **team must always have a view of “the whole”** so that individual team members always know the part their work plays in delivering the big picture goal.
Keeping the big picture in mind helps the team achieve the agreed goal for each iteration.
An important goal of iteration planning is to ensure that **team members are only loaded with the right amount of work**, not too much that they will burn out and not so little that they become bored.
The aim is to create the best flow of delivered stories. This will require the discovery and management of dependencies, and risks.
Iteration planning also serves to confirm the existing estimates and priorities and for the **team to commit to a delivery - not just once, but continuously**.

Iteration planning

- **Timing**: Iteration planning is conducted at the start of every iteration, during the "iteration planning meeting."
- **Who is involved**: All delivery teams, product owner(s), system architect(s), and any other experts, including subject matter experts.
- **Preparations**: Prior to the planning meeting, there needs to be enough detail known and available to enable the delivery team to **decompose stories into tasks and estimate these**.
- **What**: During iteration planning the Iteration plan itself is created.

It contains detailed stories **broken into tasks**, estimates for those tasks, commitments from team members, and makes note of all dependencies.

- **Tangible**: The output of the planning meeting is tangible; a spreadsheet or set of cards that provides details about which tasks go with which stories.

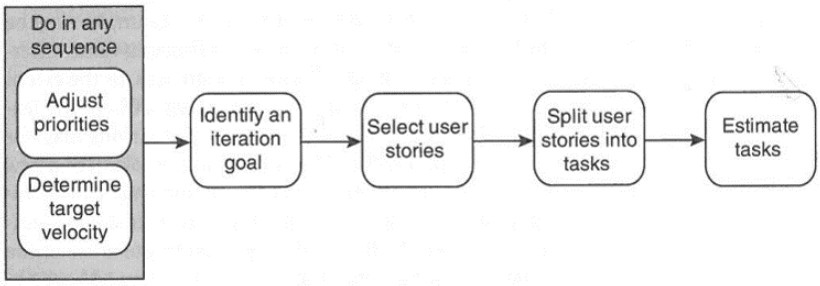
Simple planning:

- To be successful, iteration planning should be **simple**, involving cards or “post-it” notes. It should also be democratic and collaborative and must encourage conversations, inquiry and the development of understanding by the whole team.
- By encouraging conversations and using cards or post-it notes instead of recording information electronically in the first instance, it **removes “power” from the typist**, who might otherwise simply type what they believe the case to be.
- Successful planning allows and enables **everyone to participate in the process**.
- Involving the whole team in the planning leads to better results.

- During planning, many teams do not allocate tasks. This is the ideal situation but it assumes that members of the team are fairly equally skilled. It allows for steady development and progress regardless of what hidden issues are uncovered as the iteration progresses.
- Cards / notes made during the meeting may be entered into software AFTER the meeting.

Two main approaches to iteration planning

Velocity-driven iteration planning



Velocity-Driven is based on previous velocity.
Steps 1 & 2 - Adjust priorities and determine the target velocity: can be carried out in any order

1. Adjust priorities

This is done collaboratively and is based on learning from earlier iterations. It allows for the emergence of good new ideas. Prioritisation is driven by organisational value (usually financial), cost, significance, risk, etc.

2. Determine target velocity

The default position is the same as the previous iteration. On the other hand, the target velocity can be moving average over last 3 iterations. Which is more appropriate will depend on the team and the variation in velocity that they experience between each iteration.

5 iterations of velocity 12, 12, 14, 14, 10, ?
Moving average for Sprint 4 = 13, Sprint 5 = 14, Sprint 6 = 13

Step 3 - Identify iteration goal

Concise description of what the team will achieve this iteration. Unifying statement, agreed to by each member of the team, and is not overly specific.

Step 4 - Select User Stories

Working together, the product owner and the team select stories that will help the team meet the iteration goal. Story selection is based on the priority assigned to each story

Step 5 - Split user stories into tasks

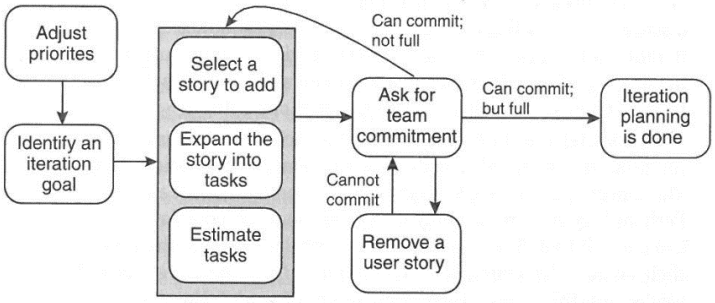
When splitting stories into tasks, it is important to include only work that adds value to **THIS** iteration. Tasks must be specific, and must include unit tests. Time must also be allowed for meetings (participating AND preparing) and to allow for bug fixing. It is important also to ensure that all dependencies are accounted for. This can be difficult when dependencies don't allow for development in their natural order

Step 6 - Estimate tasks

Task estimation can be done either AFTER all tasks are identified or AT THE SAME TIME as each task is identified.
Estimation should be a collaborative activity as it helps to identify misunderstandings and so leads to greater accuracy of estimation. It is OK to have some design discussion during estimation, but this is usually only about how something will be implemented or UI design. Guesses are all that is needed at this point.
The right size task is something that is able to be completed in a single day.

Commitment-driven iteration planning

Commitment-Driven is based on adding stories.



Steps 1 & 2 - Adjust priorities & Identify iteration goal

As for velocity-driven planning.

Steps 2, 3, 4 & 5 - Select a story, Expand the story into tasks and estimate the tasks

As for velocity-driven planning, except these steps are done iteratively, story-by-story.

Step 6 - Seek Team Commitment

At this point team commitment to the individual story is sought. The team is asked, “Can you commit to delivering the features we’ve discussed?”

The significant difference with this approach is that the team is asked to **commit to deliver new functionality not a set of tasks**.

This commitment to delivery of functionality includes team commitment to delivering any new tasks discovered during the iteration. This can be problematic as it may not be possible to complete all the new tasks within the iteration. If this is the case, the situation is discussed with the product owner.

The team and the product owner collaboratively determine how to meet the iteration goal. This may involve reducing functionality to be delivered or by dropping a story in its entirety.

Steps 2,3,4,5 & 6 are carried out iteratively until the iteration is “full”.

Developing commitment

- Developing team commitment to the iteration involves summing the estimates for each task while remembering to include hidden tasks such as email and participating in meetings. Once the sum is known the team is asked whether they can commit to that.
- At this point it is very important to remember that these are ESTIMATES not GUARANTEES.
- When planning always use less than the total number of working hours per day. Use 4-6 hours per day NOT 8. This allows for the fact that people are not productive 100% of the time.
- It is also very important to ensure appropriate distribution of work for the team given the skills of the team. Make sure the work is shared and allow team members to develop multiple skills.

Maintenance versus development work and commitment to the iteration

As well as development work the **team may need to support an existing system or systems**. This may be a prior version of the software the team is currently developing or it may be totally unrelated.

It is important to keep maintenance and support workload in mind when committing to developing a set of user stories. Maintenance work surround support and fixes is very unpredictable. However, the team must allow for this BEFORE determining what it can commit to.

Value of Commitment-Driven Planning

Overcomes issues with velocity-based planning:

•As the name suggests, velocity-based planning is based on the team's past velocity. Velocity is a measure of coarse-grained estimates and is **valid for estimating overall amount of work**.

•Velocity is, however, **not valid for estimating finer-grained, shorter-term work**, as it is too hard to “average out” the variability of velocity over a number of iterations.

•There is only a weak relationship between story-points and hours. **Not all story-points are equal!** - Hidden tasks and complexity -
Evaluate with business stakeholders & users/customers

Iteration Review Meeting

- Iteration review meetings are held at the end of each iteration. They are very informal; often they even “prohibit” PowerPoint slides! They require only a short preparation time and usually last no more than 2 hours. They are used to showcase the work completed during the iteration.
- **Iteration Review Meetings are attended by everyone** -- the whole team, the product owner, and anyone else who is interested, for example, management, customers, engineers.
- The purpose of the meeting is to assess the software developed against the iteration goal that was determined during the iteration planning meeting. Ideally, each product backlog item brought into the iteration is complete. However, more important is that the overall goal of the iteration has been achieved.

Iteration Retrospectives are:

- Iteration retrospectives usually the last thing done in an iteration. May immediately follow on from the iteration review meeting. Are attended by full team and the scrum master. The product owner may also attend (good idea to include the product owner as it helps avoid the "them and us" syndrome).
- Iteration retrospectives are usually **short**, about **an hour or so** unless there is conflict or a particularly difficult topic to deal with. The focus of an Iteration
- Retrospective is on improvement —
  - What went well and why?
  - What could we do better? And how?
- Construct a list of specific things team thinks it should:
  - Start doing
  - Stop doing
  - Continue doing

Team votes on what to focus on during the next iteration. Action items are noted. The next retrospective begins by reviewing the action items from the previous retrospective.
Retrospectives are not a witch hunt!
“Regardless of what we discover, we understand and truly believe that everyone did the best job they could, given what they knew at the time, their skills and abilities, the resources available, and the situation at hand.”
Giving feedback
There are two approaches to giving constructive feedback
Straight and cold

> “Suzy, I noticed you did some great work on the print module last iteration, but your unit tests were really lacking”

Or sweeten it up a little

> “Suzy awesome work on the print module. Apply that same level of detail to your unit tests and you are soon going to be world-class”

By not using “but” the whole tone and delivery of the message changes.
Acknowledging good behaviour and recognising those who deserve it can encourage more of the “right” kind of behaviour.
Retrospectives and sharing ideas help refocus and energize teams

Evaluate after release.

1. Learn to use metrics to learn if and how people use your product. Use face-to-face conversation to learn why they do or don’t use it.

2. If you predicted people would use your product; or that the company would benefit from it don't just assume that is what has happened.
3. Use metrics and conversations to really learn if your target outcomes were met.

So how do all these reviews and retrospectives happen? MEETINGS!

A meeting is **effective** when it:

- ☑ Achieves the meeting's objective.
- ☑ Takes up a minimum amount of time.
- ☑ Leaves participants feeling that a sensible process has been followed.

Your meeting is in trouble when there is:

- •No agenda
- •No stated time limit
- •No one is responsible for keeping the meeting on track
- •No one is taking notes
- •No agreement on what will happen once the meeting breaks, i.e. no action items
- •The person who can make decisions is not there.

How to hold effective meetings

Don't call a meeting when an email will do. Transmission of information works best in email format. (But be careful not to overuse email in documenting your project!)

Only call a meeting when you need to make a decision.

Always have an agenda and stick to it. Additionally, ensure you invite the right people and make sure that everyone understands the purpose of the meeting. Importantly, **make sure the invitees have the authority to make the required decisions**.

Make people show up on time. Everyone's time is valuable. Regularly showing up late indicates a lack of respect.

Get everyone talking. **Use meetings to argue and discuss things**.

Record your decisions and always include a brief background to the decision.

Always keep minutes (meeting notes). Given a meeting is about decisions, it should always have action items associated with each decision. Action items should always be assigned to someone. Always follow up on action items from the last meeting at the start of this one.

Cardinal rule of meetings: End on Time!

What is project governance?

Governance is the management framework within which project decisions are made. It provides a bridge between management and IT.

Why do we need it?

Someone has to make project decisions within the context of the business operation

Communications work better when the people sending the messages have recognised authority

Why is project governance important?

Project governance seeks to answer the following two questions:

1. Are we getting **value for money**?

   It is important to measure the **effectiveness of the money spent**. Use **fact-based measurement** based on outstanding scope, work completed, total expenditure and trends

2. Do the proposed solutions meet our full expectations?

   How **complete is the solution**? Does it address the full range of corporate policies, including security, architecture, quality, risk etc.

Quality Governance leads to

- a reduction of "surprises,"
- increased trust and confidence, and
- execution that is aligned with strategy

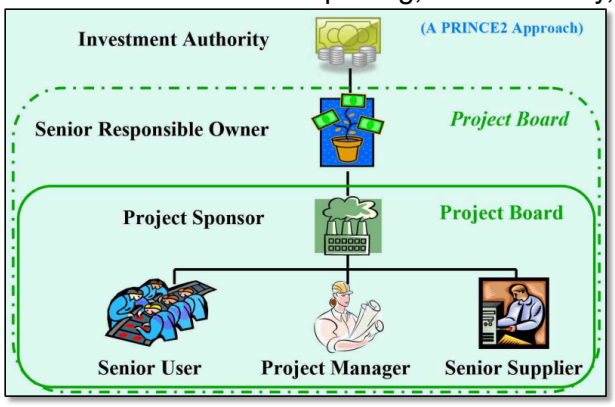
Quality governance makes IT less mired in its own problems and, consequently, better able to be responsive to the business.

Core Governance Principles

- ☑ Single point of accountability for project success (someone with authority, may be called the Senior Responsible Owner)
- ☑ Service delivery ownership determines project ownership (because the primary reason for a project is to deliver a service outcome)
- ☑ Separate stakeholder management and project decision-making (so that decision making team is small and focussed)
- ☑ Separate project governance and organisational governance (because the project is about change, not business as usual)

The Three Pillars of Governance

1. **Structure** : Determines how the decision-making is organised (e.g. decisions about investment, business change, timing etc.)
2. **People** : Roles, skills, authorities, recognition, influence, commitment
3. **Information** : Lines of reporting, accountability, communication



Key Governance Roles

A. **Establish** the basis for project governance (roles accountabilities, policies, standards, management/reporting processes)
B. **Evaluate** project proposals
C. **Enable** projects through resourcing and harnessing business support
D. **Define** the desired business outcomes, benefits, and value from the project
E. **Control** scope, contingency funds, overall project value
F. **Monitor** progress, commitment, results
G. **Measure** outputs, outcomes, benefits & value
H. **Steer** the project, remove obstacles, managed critical success factors, remedy shortfalls
I. **Develop** the organisation's project delivery capability

What can go wrong?

Collectively, governance (structure, people and communication):

- ⌑ Does not understand projects
- ⌑ Does not understand its role
- ⌑ Does not have the skills or knowledge for the role
- ⌑ Is not visible, active and committed to the project's success

- Does not enjoy the respect or trust of the business
- Is not adequately informed

ITIL

Information **T**echnology **I**nfrastructure **L**ibrary (2011) – supports IT Service Management and underpins ISO/IEC 20000 – The International Service Management Standard.
ITIL is used to help improve the **quality of services** delivered by providing a standard and repeatable set of functions, processes and procedures for service management.

PMBoK

Project **M**anagement **B**ody **o**f **K**nowledge (2013) – Developed from work done by the Project Management Institute.
PMBoK is used to help improve **project management** of projects. It includes a separate extension focussing on Software.

COBIT

Control **OB**jectives for **I**nformation and Related **T**echnologies (2012) - “aims to research, develop, publish and promote an authoritative, up-to-date international set of generally accepted ***information technology control objectives for day-to-day use by business managers***, IT professionals and assurance professionals”.
COBIT is a **governance** methodology - helps organisations govern their projects, do self-assessment, and manage culture and “best practice” behaviour.

PRINCE2

PRojects **IN** a **C**ontrolled **E**nvironment (2009) – Is a ***project management methodology***, developed by the UK Government.
PRINCE2 is a variant **project management** methodology - helps improve project management of projects.

We manage quality

So that we
•understand what quality means to the customer
•can take pre-emptive action as well as be responsive when problems with quality arise
•understand the impact of changing project scope, schedule and budget

Because it
•is orders of magnitude cheaper to prevent problems than fix problems
•avoids the situation where “Nothing Works Until it All Works”
•we don’t like to redo something we think is already done!

Stakeholder	What is their stake?
Us (Development team)	We're still working on the project We want a good reference We're professionals
Customer / Client	Wants a good solution that is fit for purpose that doesn't cost the earth
System Users	Rely on our solution to perform some action with accuracy time after time
Maintainers and Support Staff	Maintain our code and extend our solutions after we‘ve left the project/team/ company

Assumptions underlying Quality Management

- Customer satisfaction is a key component of project success. Customer satisfaction can be described as conformance to requirements and fitness for use.
- It is better to plan quality in and prevent defects and errors from occurring rather than to find errors during the inspection process.

Plan – Do – Check – Act, as defined by Deming and Shewhart is a strong basis for quality improvement, but is **not explicitly defined** in quality processes

Much of the investment in quality **comes from the organisation**.

Quality processes and procedures – certifications - investment in proprietary methodologies (TQM or Six Sigma) - quality audits – are the **responsibility of the organisation**. It is the **responsibility of the project** to **follow these processes**.

Different ways we manage quality

Pro-active Approaches
• Planning
• Processes and product descriptions
• Standards and templates
• Process and product metrics
• Communication
• Reporting

Re-active Approaches
• Testing -- just one way to manage quality
• Reviews, inspections and walk-throughs
• Defect and issue tracking
• Comparison with software quality factors

Indicators that Quality standard is at risk

LEAD INDICATORS

- No Implementation Plan or plan not independently reviewed
- Lack of process (even informal processes can be recognised)
- No coding standards
- Infrequent contact with client and end-users
- No accountability among team members or unable to articulate their contribution
- Inability to demonstrate progress
- Unexpected risks are being realised
- Team members only work independently and no or few peer-reviews occur

- Lots of technical “workarounds”
- Insufficient attend given to key dependencies between project roles
- Insufficient expertise or all necessary people are not providing input

LAG INDICATORS

- System instability (small changes have wide-ranging impacts)
- Inability to demonstrate progress
- Defects are only being discovered by users
- Defects take a long time to correct

Practical techniques you can do now

- Have a plan that identifies the sequence of work that gets you to the end.
- Have frequent meetings with your client – one a week if possible
- Allow time to fix bugs in your schedule.
- Have a coding standard and ensure you inspect each other’s code as part of your weekly responsibilities.
- Build continuously
- Plan your test strategy: who, how, when, where
- Use proper bug tracking software. It's free and it will help ensure issues don’t slip through to production
- Ensure you agree the process of handing over to the client ***before*** you finish development

Some observations:

Most people don’t recognise poor quality until its too late to fix.
You can differentiate yourself by being able to recognise when things aren’t right, and be able to suggest how to make them better.
Quality is a habit, it relies on continuous, ongoing effort.
Get into the habit of following quality processes and implementing quality controls and your clients will be a whole lot happier.

- Quality is a **whole-team responsibility**
- The **concept of “done”** is fundamental to quality development
- **Testing is integral to the process** rather than something that is left until later
- Testing is automated, but this is not the only form of testing
- Continuous integration

The importance of having a Definition of Done (DOD)

- Comprehensive collection of value add deliverables that assert the quality of a feature not the functionality of that feature
- Is informed by reality
- Exists at 3 levels: Feature Iteration Release

Relationship between “done” and “conditions of satisfaction”

- Definition of “done” (DoD) is a special set of conditions which is applied to every user story before any story is considered complete
- “Conditions of Satisfaction” (COS) are specific to an individual user story

The importance of “done” for quality software

- Overcomes our tendency to indicate a task is complete, when what we really mean is that the programming is complete
- Sets the minimum acceptable standard for the team
- Enables the team to deliver potentially shippable software at the end of each iteration
- Definition changes over time
- **UI** is at the top of the pyramid because you want to **do as little automated UI testing as possible** because:
  A small change to the UI can break many tests – it is brittle
  Expensive – writing a good UI test that remains useful and valid takes time

Tests take a long time to run
- **Service layer** - something the application does in response to input(s)
  Test services separately from the UI
  Write scripts that pass the data to the right services
- **Unit testing is the foundation upon which a solid automation strategy is built**

The Role of UI testing

- To confirm that
  the services are hooked up to the right buttons
  the values are displaying properly in the results field
- Automated unit testing cannot cover all of an application's testing needs
- Service-level testing fills the gap between UI and unit testing

Automate within the iteration

✓ To deliver value quickly relies on automated testing
✓ Provides insight into the state of the product and the process
✓ Automate test early – always in the same iteration in which the code is written
✓ Retrofitting automated tests is difficult
✓ Adding early means design of tests could influence design of product

There is still a role for manual testing…

- Impossible to fully automate all tests for all environments
- Generally a way of doing exploratory testing
  - Rapid cycle through test planning, test design, and test execution
  - Finds missing test cases
  - May uncover ideas missing from a user story

Resolve technical debt as quickly as possible

Technical debt is the cost of working on a system that is poorly designed, poorly written, includes unfinished work or is deficient in any number of ways
It is often the result of rushed implementation
Technical debt gets in the way of quality development and testing
Difficult to run short sprints with manual testing
Find ways to automate testing

- *Customers suffer if product is of low quality*
- *Entire team suffers if testing is not integrated into the process or it is not done at the right level*
- *A good agile team is constantly vigilant of the state of its testing practices*

Benefits

- An improved reputation for quality
- Lower post-release maintenance costs
- Smoother release cycles
- Increased confidence that the system will work well and satisfy customers.
- Protection from lawsuits initiated by unhappy customers, users, or those otherwise affected by system failure.
- Reduce risk of loss of entire missions and even lives.

Costs

Conformance
- testing (finding bugs, a.k.a defects)
- quality assurance (preventing bugs)

Non-conformance
- Fixing bugs
- Retesting
- Dealing with angry customers
- Damage to company image
- Lost business, etc.

Ensuring Quality

Reactive
Quality Control
Review, inspect, analyse and test products

Pro-active
Quality Assurance
Plan, manage, implement and monitor processes

PMBOK Quality Management

Plan for Quality

Process of identifying quality requirements and/or standards for the project and product, and documenting how the project will demonstrate compliance

Perform Quality Assurance

Process of auditing the quality requirements and results from quality control to ensure use of appropriate quality standards and operational definitions

Perform Quality Control

Process of monitoring and recording results of executing the quality activities to assess performance and recommend necessary changes

Quality Control Methods

Formal and informal reviews such as: Peer reviews, walk throughs, inspectionsTesting, Prototypes, Demonstration, Modelling, Simulation, Audits

What are we quality controlling for?

Functionality

Existence of a set of functions and their specified properties

Security/ Accuracy/Regulatory compliance/Technical compliance/Interoperability

Reliability

Capability of the software to maintain its level of performance under stated conditions for a stated period of time

Data Integrity/ Error Handling/Fault Tolerance/Recoverability

Usability

Effort needed for use by a stated or implied set of users

Software Quality Characteristics

User Friendliness/User Guidance/Homogeneity/Adaptability/Clarity of Control/Error Handling/ Conciseness/Ease of learning/Documentation quality/East of installation

Efficiency

Relationship between the level of performance of the software and the amount of resources used, under stated conditions:

Throughput/ Acceptable response time/ Data storage requirements/ Acceptable memory capacity/ Acceptable processing speed

Maintainability

Effort needed to make specified modifications

Analysability/ Changeability/ Testability/Vendor support/ Expandability / Scalability

Portability

The ability of software to be transferred from one environment to another / used in multiple environments

Portability to different hardware platforms/Compatibility with different operating systems

Conformance/Replace-ability/Languages supported

Testing and Maintenance

Testing can reduce the cost of quality

•Successful testing is not a few people in a dark lab at the end of the project …
•Concurrent – starts with requirements and continues throughout the lifecycle
•Cross-Functional – many people play a role
•Testers are the only ones who design, write and run tests.
•Collaborative – testing has dependencies throughout the project team

… winning test strategies pervade software development

The Paradox of Software Testing

Testing software tells us very little about its quality!
A failed test only provides evidence of non-quality
A successful test is only relevant to quality assessment if it previously failed

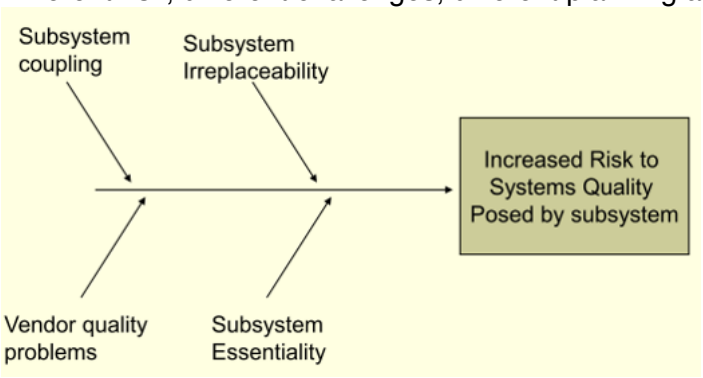
“Testing shows the presence of bugs not their absence” (Edsger Dijkstra)

System, subsystem COTS and Component Integration

Not every major system project involves a lot of system development or maintenance. Often centres on acquiring, integrating, or customising existing software or hardware. Increase one or more of these **risks**:

•Criticality of integrated component to success.
•Coupling between integrated component and any critical component.
•Large degree of component customisation or adaption required.
•Large number of supported configurations.
•Broken by new releases.
•Poor vendor testing practices.
•Poor vendor support.
•Maybe small user base sees component go obsolete.

Different risk, different challenges, different planning and estimation required.



Summary

•Quality is a measure of conformance with requirements
Requirements can be set for products, processes and projects
•Quality management activities can be both reactive and proactive
•There are costs associated with ensuring quality and the failure to ensure quality
•Testing can help reduce the cost of quality
•However, successful tests don’t necessarily provide evidence of software quality
Testing shows the presence of bugs not their absence
•Software requirements are usually more than just functional requirements

Risk

Uncertain or chance events that planning can not overcome or control.
The combination of the probability of an event and its consequence.

NOTE:
The term "risk" is generally used only when there is at least the possibility of negative consequences.
In some situations, risk arises from the possibility of deviation from the expected outcome or event.

According to PMBOK：
An uncertain event or condition that, if it occurs, has a positive or negative effect on a project’s objectives

According to AS/NZS 4360:2004 Risk Management:
Risk is the chance of something happening that will have an impact on objectives (Risk may have a positive or negative impact)

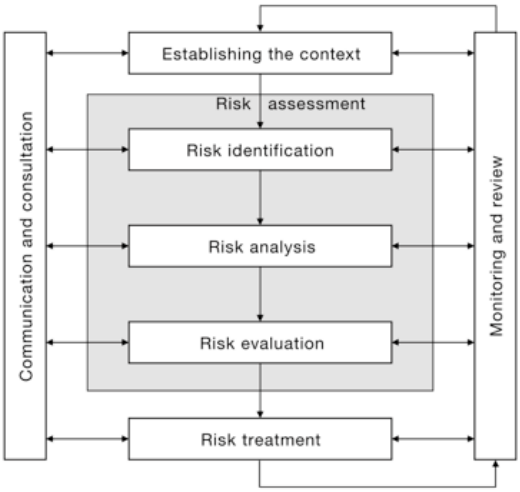
According to PRINCE2:
A risk is an uncertain event or set of events that, should it occur, will have an effect on the achievement of objectives

Risk Management

A proactive attempt to recognize and manage internal events and external threats that affect the likelihood of a project's success.

What can go wrong (a risk event)

- How to minimize the risk event's impact (manage consequences)
- What can be done before an event occurs (anticipation)
- What to do when an event occurs (contingency plans)



Uncertainty – Likelihood, Impacts/Consequences, Triggers

Impact - Negative and positive (risks and opportunities), Immediate, Delayed, Ongoing

Table 4.1: Example of qualitative risk quantification

Likelihood	*Impact/Consequence*
A = Almost Certain L = Likely M = Moderate U = Unlikely	C = Critical H = High M = Medium L = Low

Table 4.2: Example of semi-quantitative risk evaluation

Likelihood	*Impact/Consequence*
A = within 12 months L = within 1 to 2 years M = within 2 to 5 years U = Unlikely	C > $10M H = $5M to $10M M = $1M to $5M L < $1M

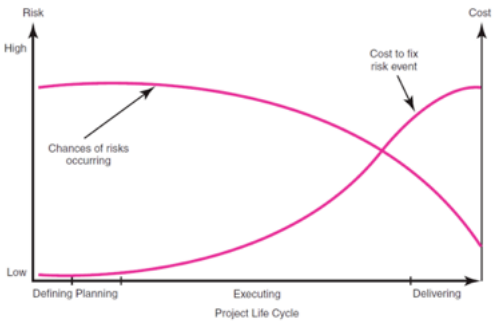
Risk Event	Response	Contingency Plan	Trigger	Who Is Responsible
Interface problems	Mitigate: Test prototype	Work around until help comes	Not solved within 24 hours	Nils
System freezing	Mitigate: Test prototype	Reinstall OS	Still frozen after one hour	Emmylou
User backlash	Mitigate: Prototype demonstration	Increase staff support	Call from top management	Eddie
Equipment malfunctions	Mitigate: Select reliable vendor Transfer: Warranty	Order replacement	Equipment fails	Jim

Options for risk treatment

Options, which are not necessarily mutually exclusive or

appropriate in all circumstances, include the following:

a) risk avoidance;
b) reduction of likelihood;
c) reduction of consequences;
d) risk transference; and



Benefits of Managing Risk

A proactive rather than reactive approach.

Reduces surprises and negative consequences.

Prepares the project manager to take advantage of appropriate risks.

Provides better control over the future.

Improves chances of reaching performance objectives within budget & on time

Areas of risk common to all projects

Intrinsic schedule flaw - Estimates are wrong and undoable – often based on "wishful thinking"

Specification breakdown - Failure to achieve stakeholder consensus on what to build

Scope creep - Additional requirements that inflate the original agreed set

Personnel loss
Productivity variation - Difference between assumed and actual performance

Risk is dynamic

Risks themselves change throughout the life of the project
Risk profile (likelihood and consequence) will change
Risk treatments must change in response to profile change
Require constant monitoring and updating

PMBOK Risk Management

Plan risk management
Identify risks
Perform qualitative risk analysis
Perform quantitative risk analysis
Plan risk responses
Monitor and control risks

There is currently uncertainty about risk in agile projects
There us no definite consensus on the need for risk management within agile methodologies.
Therefore, many people believe that risk management is irrelevant with an iterative model
However, Agile approaches do not **"risk proof" your projects**!… But they help…..

Benefits of Agile when managing risk

- Risk is owned by the team
- Agile approaches encourage proactive risk management
  Identified and reviewed in all planning meetings: release, iteration and daily stand-up
- Testing is an integral part of each iteration
- Can consider agile methods as "risk-driven"
  Pull user stories with risks forward in the backlog
- Increased engagement
  Done by the whole team not just the PM
- Revisited frequently – with each iteration
  Rather than only at the start of a project when we know least, and only reviewed occasionally
- Integrated into lifecycle
- High visibility
  Stakeholder engagement because risk management is a team activity

Proactive Risk Management – reduces risk

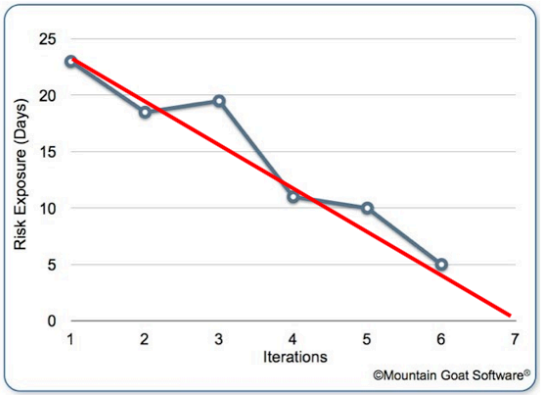
- Rigorous application of agile methodologies inherently reduces risk in software development
- Attack risky work early
- Use existing artefacts and meetings to help manage risk
- Large scale, explicit risk management is not necessary with agile. ***Why?***
  1. Short iterations
  2. Focus on working software
  3. Automated tests and constant builds
  4. Frequent deliveries to the customer

Risk area	Problem	Agile helps because…
Intrinsic schedule flaw	Likely has the biggest impact due to tendency to underestimate	Continual evaluation of estimation leads to improved accuracy release planning and iteration planning Plan is continually updated and refined

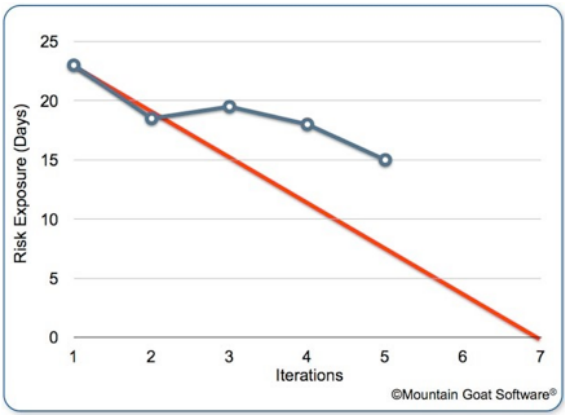
Specification breakdown	Multiple customer contacts may not agree on what is needed and its priority	Product owner is empowered to make decisions Increased communication Disagreement within team about the how resolved in “set-based design” – work through multiple potential solutions
Scope creep	Longer delivery cycles encourage this	Agile accepts change as necessary Remember to keep release plan highly visible so impact of change is always obvious
Personnel loss	Loss of resource/knowledge	Self-organising teams lead to a) lead to higher morale and b) share knowledge
Productivity variation	Actual performance vs Planned performance	End of iteration review and demonstrations make this visible, early Shorter feedback loops lead to less deviation

Create risk burndown chart

Plot the sum of the risk exposure values from the census
Only sum the **top ten risks**, even if more have been identified
Top ten will change over the course of the project



As with regular release burndown chart, there should be a linear drop in risk over the course of the project (red line)



When risk doesn’t come down at an appropriate rate budget some time in the next sprint to work directly on risk mitigation
Risk not reducing quickly enough across iterations
The burn-down chart represents the status of the risk across the iterations.
From a project management perspective, this is an excellent indicator of how the risks are managed and controlled.