PACE UNIVERSITY
Seidenberg School of Computer Science and Information Systems

CS631G Software Verification

CHAPTER 3:

AGILE SOFTWARE DEVELOPMENT (W2)

CHAPTER 22:

PROJECT MANAGEMENT (W2)

Class instructor: Yuri Chernak, PhD

Outline

Part I. Introduction to Agile Methods

- Agile Manifesto and Principles
- Extreme Programming Overview
- XP Development Practices
- User Stories
- Agile Project Management
- Scrum Overview
- Agile Methods Applicability

Part II. Project Management Practices

- General Project Management Overview
- CMMI: Project Planning Goals and Practices
- CMMI: Project Monitoring and Control
- Risk Management Overview

Key Points

Exercise

Class Objectives

The objective of this class is to introduce agile software development methods.

At the end of this class, you will:

- understand the rationale for Agile software development methods, the Agile Manifesto, and differences between agile and plan-driven (traditional) development;
- learn about important Agile development practices such as user stories, refactoring, pair programming and test-first development;
- understand the Agile management practices and general project management discipline and process.

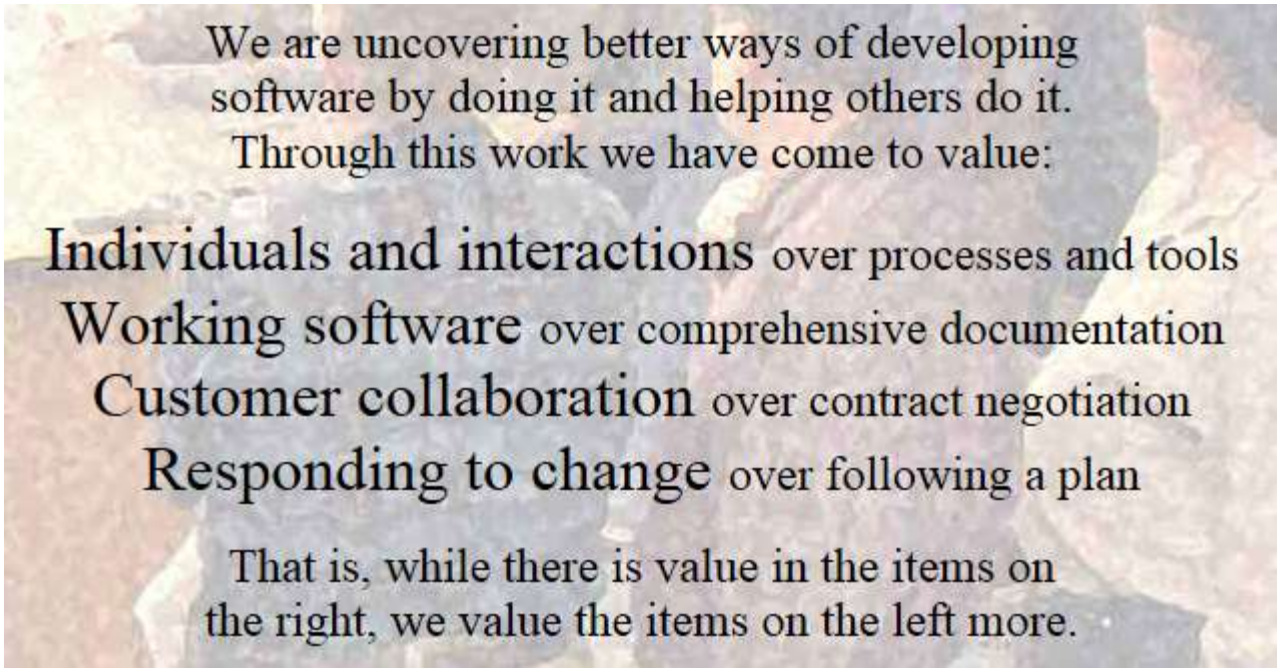
Part I.
Introduction to Agile Methods

Motivation for Agile Methods

- Dissatisfaction with the overheads involved in software design methods of the 1980s (Waterfall) and 1990s (RUP) led to the creation of Agile methods.
- Agile Software Development is a set of software development methods in which requirements and solutions evolve through collaboration between self-organizing, cross-functional teams.
- These methods:
  - focus on the code rather than the design; there is no detailed system specifications, design documentation is minimized.
  - are based on an iterative approach to software development; the system is developed in a series of increments.
  - are intended to deliver working software quickly and evolve it to meet changing requirements.
- The aim of agile methods is to reduce overheads in the software process (e.g., by limiting documentation) and to be able to respond quickly to changing requirements without excessive rework.

Agile Manifesto

- In February 2001, seventeen software developers met at the Snowbird resort in Utah to discuss lightweight development methods.
- They published the *Manifesto for Agile Software Development:*



Twelve Principles of Agile Methods

Principles behind the Agile Manifesto:

1. Our higher priority is to satisfy the customer through early and continuous delivery of valuable software.
2. Welcome changing requirements, even late in development. Agile processes harness change for the customer's competitive advantage.
3. Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale.
4. Business people and developers must work together daily throughout the project.
5. Build projects around motivated individuals. Give them the environment and support they need, and trust them to get the job done.
6. The most efficient and effective methods of conveying information to and within a development team is a face-to-face conversation.

Twelve Principles of Agile Methods

7. Working software is the primary measure of progress.
8. Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely.
9. Continuous attention to technical excellence and good design enhances agility.
10. Simplicity – the art of maximizing the amount of work not done – is essential.
11. The best architectures, requirements, and designs emerge from self-organizing teams.
12. At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly.

Popular Agile Methods

Over the years, various agile software development methods and/or process frameworks have been developed that include (but are not limited to):

- Adaptive software development (ASD)
- Agile modeling
- Agile Unified Process (AUP)
- Business analyst designer method (BADM)
- Crystal Clear Methods
- Disciplined agile delivery
- Dynamic systems development method (DSDM) Feature-driven development (FDD)
- Lean software development
- **Extreme programming (XP)**
- **Scaled Agile Framework (SAFe)**
- **Kanban**
- **Scrum**

Discussed in this lecture.

Extreme Programming Explained

- Extreme programming (XP) is a software development methodology which is intended to improve software quality and responsiveness to changing customer requirements.
- As a type of agile software development, it advocates frequent "releases" in short development cycles, which is intended to improve productivity and introduce checkpoints at which new customer requirements can be adopted.
- The name “Extreme Programming” was coined by Kent Beck in 1998, because the approach was developed by pushing common software practices to “extreme” levels.
- Extreme Programming was controversial when it was developed, as it introduced agile practices that were quite different from the conventional practices at that time.

XP Four Values

The four values of XP are:

Communication

- Problems with projects can be frequently tracked back to bad communication.
- Bad communication does not happen by chance, there are many circumstances that lead to a breakdown in communications.
- XP aims to keep the right communications within the team.

Simplicity

The second XP value is simplicity, which means it is better to do a simple thing today and pay a little more tomorrow to change it if it needs, than to do a more complicated thing today that may never be used anyway.

XP Four Values (continues)

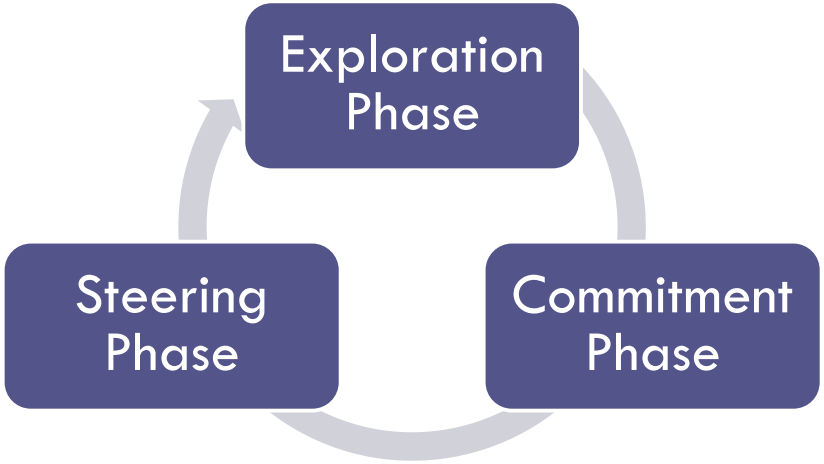
Feedback

- Concrete feedback about the current state of the system is priceless.
- Feedback works at different time scales:
  - Unit tests and user story estimation provide feedback at the scale of minutes and days.
  - Functional tests, schedule tracking, and velocity review provide feedback at the scale of weeks and months.

Courage

- Responding to change needs courage.
- XP projects can face various decisions like changing architecture, throwing away bad code, design alternatives, etc. that require courage to make the right choice.

Phases of the XP Process Framework



1. Exploration	2. Commitment	3. Steering
• Write a user story • Estimate a user story • Identify, record tasks • Split/combine tasks	• Sort stories by value • Set velocity • Choose iteration scope • Estimate and accept tasks	• Plan and implement iterations • Add new user stories and re-estimate • Implement tasks • Record progress • Verify, test stories

XP Development Practices

Practice	Description
Pair programming	Developers work in pairs, checking each other's work and providing the support to always do a good job.
Collective ownership	The pairs of developers work on all areas of the system, so that no islands of expertise develop and all the developers take responsibility for all the code. Anyone can change anything.
Continuous integration	As soon as the work on a task is complete, it is integrated into the whole system. After any such integration, all the unit tests in the system must pass. *This practice later has evolved into the DevOps methodology.*
Incremental planning	Requirements are recorded on story cards (*currently in Jira*) and the stories to be included in a release are determined by the time available and their relative priority. The developers break these stories into development 'Tasks'.
On-site customer	A representative of the end-user of the system (the customer) should be available full time to interface with the XP team. In an extreme programming process, the customer (*or Product Owner*) is a member of the development team and is responsible for bringing system requirements to the team for implementation.

XP Development Practices (continued)

Practice	Description
Re-factoring	All developers are expected to re-factor the code continuously as soon as potential code improvements are found. This keeps the code simple and better maintainable.
Simple design	Enough design is carried out to meet the current requirements and no more.
Small releases	The minimal useful set of functionality that provides business value is developed first. Releases of the system are frequent and incrementally add functionality to the first release.
Sustainable pace	Large amounts of overtime are not considered acceptable, as the net effect is often to reduce.
Test-first development	An automated unit test framework is used to write tests for a new piece of functionality before that functionality itself is implemented.

DevOps Explained

Over the years, the original XP methodology has been extended by the DevOps methodology, which is a mainstream in the IT Industry these days.

DevOps Definition

- **DevOps** is a combination of software developers (dev) and operations (ops).
- It is defined as a software engineering methodology which aims to integrate the work of software development and software operations teams by facilitating a culture of collaboration and shared responsibility.
- DevOps takes a development cycle to the **extreme of efficiency** and is based on the concept of CI/CD (Continuous Integration/Continuous Delivery).
- DevOps implementation is heavily dependent on tools.

CI/CD Explained

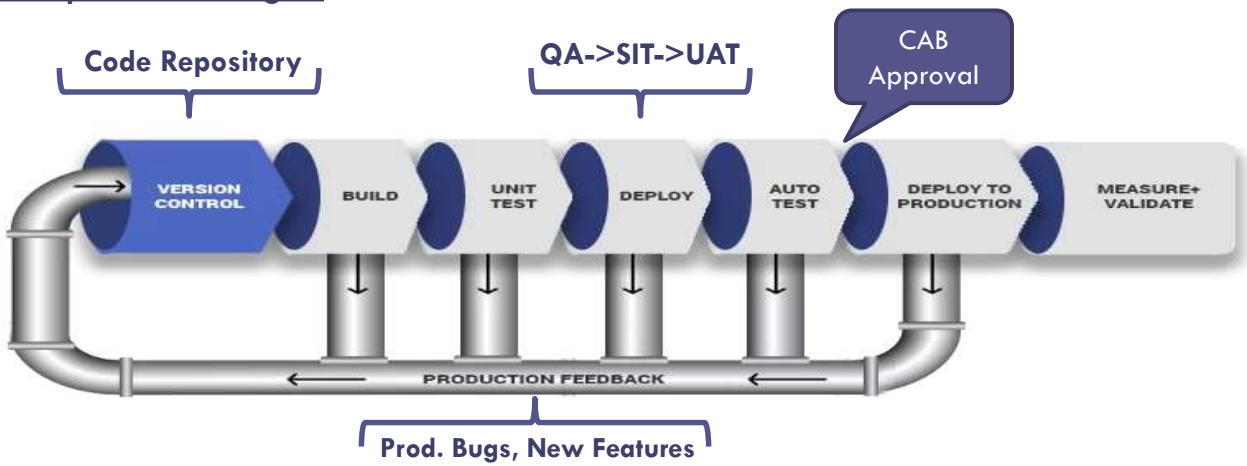
What is CI/CD?

- CI/CD is a way of developing software in which you're able to release updates at any time in a sustainable way. When changing code is routine, development cycles are more frequent, meaningful and faster.

What is the CI/CD Pipeline?

- The continuous integration/continuous delivery (CI/CD) pipeline is an Agile DevOps workflow focused on a frequent and reliable software delivery process.
- A key characteristic of the CI/CD pipeline is the use of automation at every stage of the delivery process to ensure code quality.

CI/CD Pipeline Stages



What is a User Story?

- In agile software development and product management, a user story is a description of functionality that will be valuable to a user of a software system.
- User stories are used within agile software development methodologies as the basis for defining the functionality a business system must provide, and to facilitate requirements management.
- A user story consists of one or more sentences in the everyday or business language of the end user of a system that captures what a user does or needs to do as part of his or her job function.
- It captures the 'who', 'what' and 'why' of a requirement in a simple, concise way, often limited in detail by what can be hand-written on a small paper notecard.

Purposes of a User Story

User stories are written for the following purposes:

- **Planning**. User stories can be used for planning project iterations.
- **Development**. Once user stories are written, then the development team breaks them into development tasks to implement.
- **Testing**. User stories provide a basis for developing and executing acceptance tests to determine when a story's implementation is complete.

Where are the details?

- Details of a given user story can be captured as additional user stories, supplementary requirements, or story acceptance tests and acceptance criteria (*discussed on next slide*).
- It is better to have more stories than to write stories that are too long and might not fit a given iteration.

User Story Acceptance Criteria

- User story Acceptance Criteria are a set of conditions or requirements that must be met for a user story to be considered complete and ready for implementation.
- They serve as a detailed specification that outlines the functional and non-functional expectations of a user story, ensuring a shared understanding between the development team and stakeholders.
- Acceptance Criteria are crucial in Agile development as they define the boundaries and conditions under which a feature or functionality is considered done.
- Acceptance criteria provide detailed requirements for the user story, breaking down the desired functionality into specific, actionable tasks or conditions.
- Each acceptance criterion should be measurable and testable. This means that it should be clear when the condition has been met and can be objectively tested.
- Acceptance criteria define the conditions under which the user story implementation can be considered successful. That includes story constraints, exceptions, etc. These conditions often serve as a checklist for testing.
- Acceptance criteria are not static and can evolve over time. As the team gains more insights or receives feedback, criteria may be added, modified, or refined.

Acceptance Criteria Example

User Story “Reset Password”
As a registered user, I want to reset my password.

Acceptance Criteria Example

1. When on the login page, there should be a "Forgot Password" link.
2. Clicking the "Forgot Password" link should take the user to a password reset page.
3. The password reset page should ask for the user's email address.
4. After entering a valid email address and clicking "Submit," a password reset email should be sent to the user.
5. The password reset email should contain a link to a page where the user can create a new password.
6. The new password should meet the specified complexity requirements.

Six Attributes of Good User Stories

- User stories are brief statements of intent that describe something the system needs to do for some users. They are commonly captured in a standard "user-voice" form:

  As a <user role>, I can <activity> so that <business value>

INVEST is the mnemonic representing the six attributes of a good user story:

- **I**ndependent
- **N**egotiable
- **V**aluable to users
- **E**stimatable
- **S**mall
- **T**estable

- A right-sized story from a user's perspective is one that fulfills a need.
- A right-sized story from a team's perspective is one that fits the iteration (Sprint).

User Story Examples

Title: Search for customers

- As a user, I want to search for my customers by their first and last names.

Title: Modify schedules

- As a non-administrative user, I want to modify my own schedules but not the schedules of other users.

Title: Run tests

- As a mobile application tester, I want to execute my test cases and report results to my management.

Title: Start application with last edit

- The application begins by bringing up the last document the user was working with. *Or*
- As a user, I want to start an application with the last edit.

User Stories are the Basis for Iteration Planning

- A user story contains a short description of user-valued functionality.
- The customer team, led by the Product Owner, writes the stories because they are in the best position to express the desired features.
- Stories are prioritized based on their value to the organization.
- Releases and iterations are planned by placing stories into iterations (iteration backlog).
- User stories are used by a team to estimate the iteration effort.
- If a story cannot fit in an iteration, it is split into two or more smaller stories.
- Acceptance tests are developed to validate stories.

XP: Code Refactoring

Code refactoring is the process of restructuring existing computer code without changing its external behavior to improve system maintainability and performance.

- A programming team looks for possible software improvements and makes these improvements even where there is no immediate need for them.
- This improves the understandability of the software and so reduces the need for documentation.
- Changes are easier to make because the code is well-structured and clear.
- However, some changes require architecture refactoring and this is much more expensive.

Examples

- Re-organization of a class hierarchy to remove duplicate code.
- Tidying up and renaming attributes and methods to make them easier to understand.
- The replacement of inline code with calls to methods that have been included in a program library.

XP: Test-first Development

- Test-first development is a software development process that relies on the repetition of a very short development cycle: first the developer writes an (initially failing) automated test case that defines a desired improvement or new function, then produces the minimum amount of code to pass that test, and finally refactors the new code to acceptable standards.
- Writing tests before code clarifies the requirements to be implemented.
- Unit tests are written as programs rather than data so that they can be executed automatically. The test includes a check that it has executed correctly.
- Development usually relies on a testing framework such as JUnit.
- All previous and new tests are run automatically when new functionality is added, thus checking that the new functionality has not introduced errors.

Issues with Test-first Development

- Programmers prefer programming to testing and sometimes they take short cuts when writing tests. For example, they may write incomplete tests that do not check for all possible exceptions that may occur.
- Some tests can be very difficult to write incrementally. For example, in a complex user interface, it is often difficult to write unit tests for the code that implements the 'display logic' and workflow between screens.
- It is difficult to judge the completeness of a set of tests. Although you may have a lot of system tests, your test set may not provide complete coverage.

XP: Pair Programming

- In XP, programmers work in pairs, sitting together to develop code.
- In pair programming, programmers sit together at the same workstation to develop the software.
- Pairs are created dynamically so that all team members work with each other during the development process.
- The sharing of knowledge that happens during pair programming is very important as it reduces the overall risks to a project when team members leave.
- This helps develop common ownership of code and spreads knowledge across the team.
- It serves as an informal review process as each line of code is looked at by more than 1 person.
- It encourages refactoring as the whole team can benefit from this.
- Measurements suggest that development productivity with pair programming is similar to that of two people working independently.

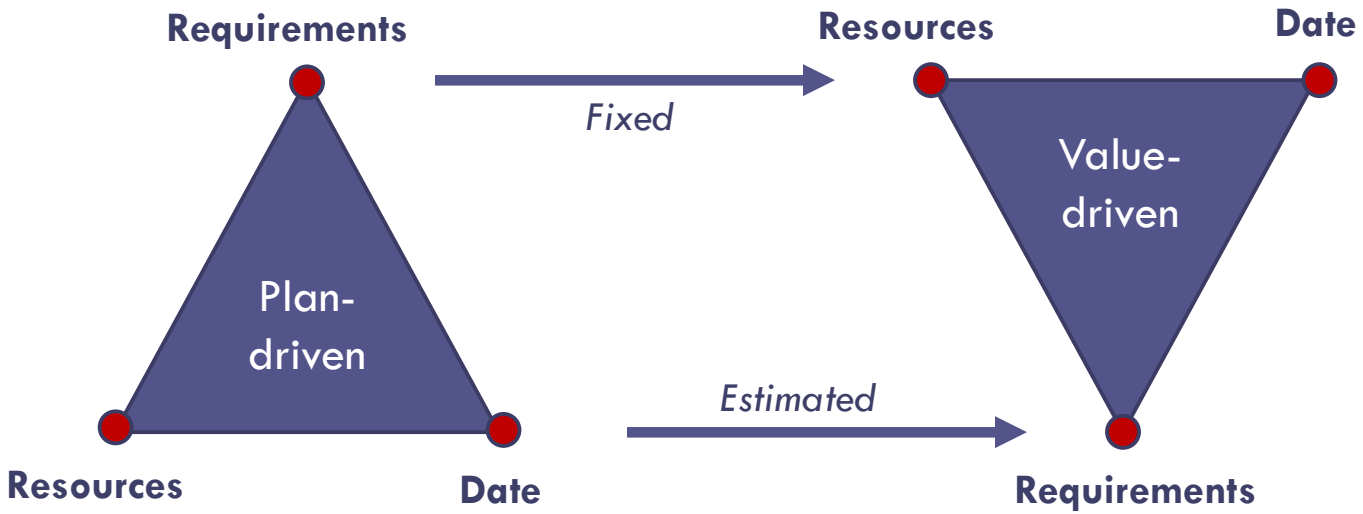
Agile Project Management

- The principal responsibility of traditional software project managers is to manage the project so that the software is delivered on time and within the planned budget for the project.
- A conventional approach to project management is plan-driven. Managers draw up a plan for the project showing what should be delivered, when it should be delivered and who will work on the development of the project deliverables.
- Agile project management requires a different approach, which is adapted to incremental development and the particular strengths of agile methods.
- Scrum is a popular project management methodology used for iterative and incremental agile development.

Agile Requirements Management

- With Agile, we take a much more flexible approach to requirements management.
- The approach is far more interactive and "just-in-time".
- Light requirements allow us to deliver on a fixed schedule and fixed resource basis.

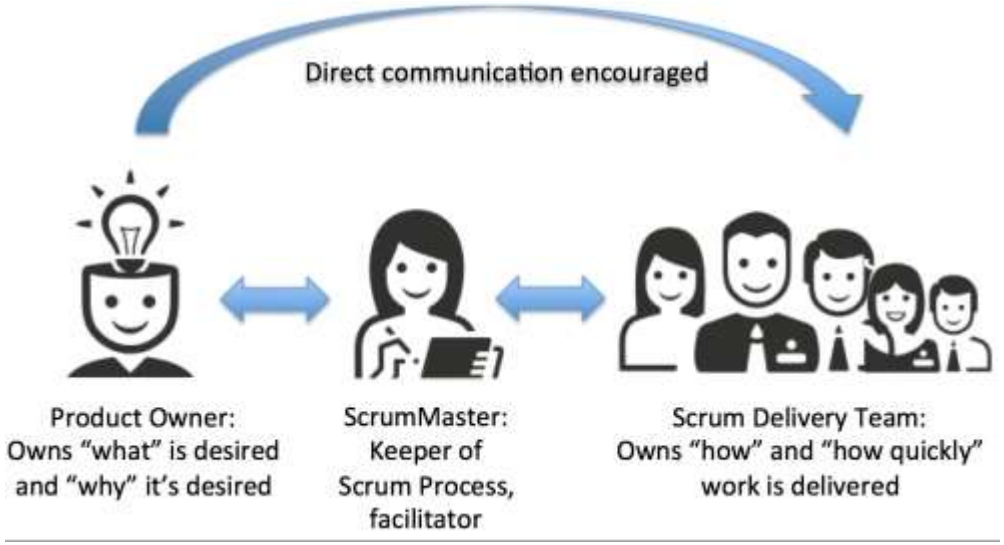
Plan-driven vs. Value-driven Approaches



Scrum Explained

- Scrum defines a flexible, holistic product development strategy where a development team works as a unit to reach a common goal.
- Scrum challenges assumptions of the "traditional, sequential approach" to product development, and enables teams to self-organize by encouraging:
  - physical co-location or close online collaboration of all team members
  - daily face-to-face communication among all team members and disciplines in the project.
- There are three roles in Scrum – Product Owner, Scrum Master, Delivery Team.

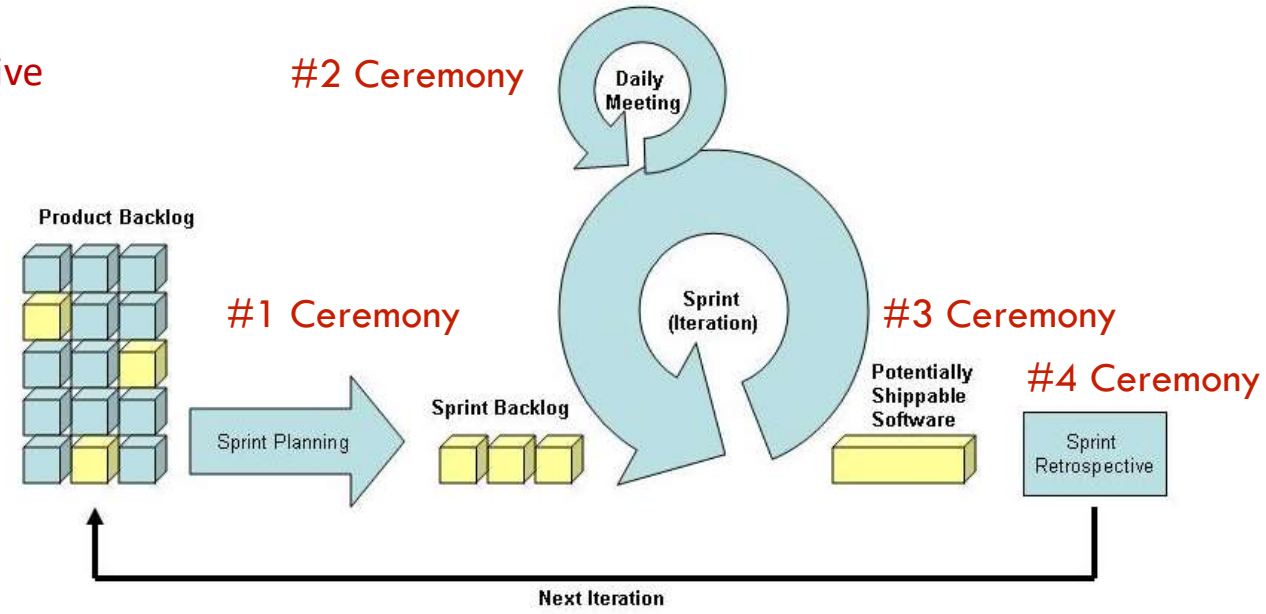
Product Owner sets a goal for a release.



Scrum Master ensures the team achieves that goal.

Scrum Workflow

- A Sprint (or iteration) is the basic unit of development in Scrum. The Sprint is a *timeboxed* effort; that is, it is restricted to a specific duration.
- The duration is fixed in advance for each sprint and is normally between one week and one month, with two weeks being the most common.
- Scrum workflow includes four Agile Ceremonies (four types of meetings):
  1. Sprint Planning
  2. Daily Scrum Meeting
  3. Sprint Review
  4. Sprint Retrospective



Four Scrum Ceremonies

- Meetings, or "ceremonies" are an important part of agile development.
- Scrum has four main ceremonies that bring structure to each sprint.

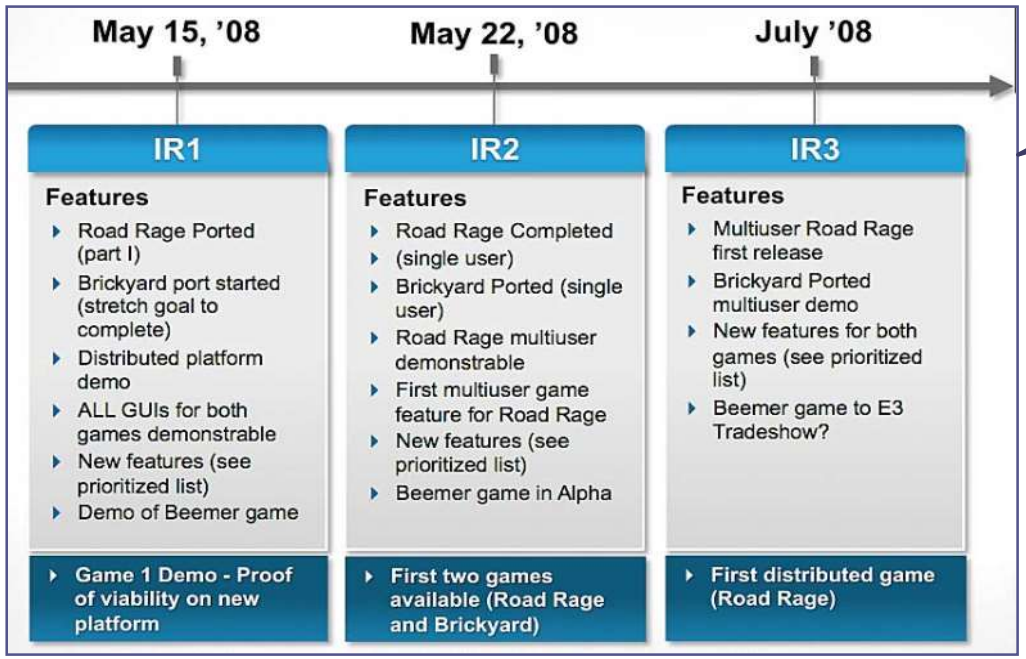
Ceremony Type	Ceremony Purpose
1. Sprint Planning	A team planning meeting that determines what to complete in the coming sprint. Coming into the meeting, the Product Owner will have a prioritized product backlog. They discuss each item with the development team, and the group collectively estimates the effort involved. The development team will then make a sprint forecast outlining how much work the team can complete from the product backlog. That body of work then becomes the sprint backlog.
2. Daily Stand-up Meeting	Also known as a daily scrum, a 15-minute mini-meeting for the software team to sync. Stand-up is designed to quickly inform everyone of what's going on across the team. It's not a detailed status meeting.
3. Sprint Review (Sprint Demo)	Sprint review is a time to showcase the work of the team. A sharing meeting where the team shows what they've developed in that sprint, whether they achieved the iteration objective.
4. Sprint Retrospective	Agile is about getting rapid feedback to make the product and development culture better. Retrospectives help the team understand what worked well–and what didn't to make the next sprint better.

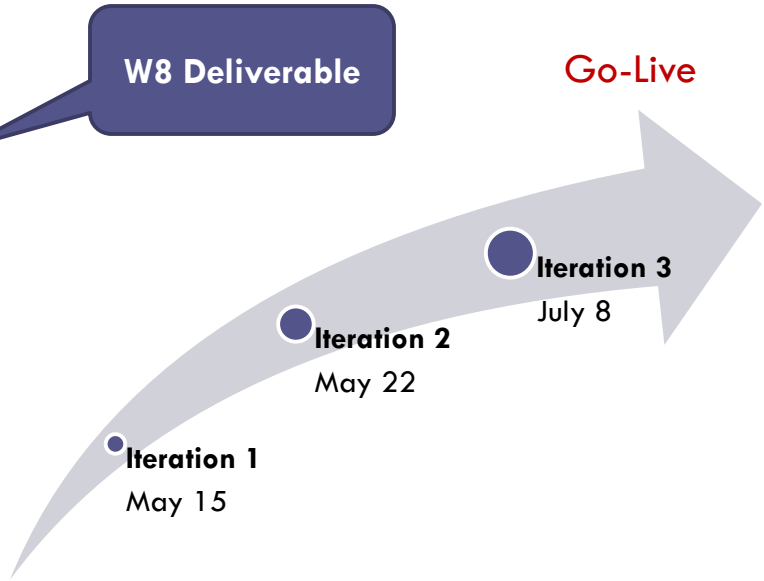
Daily Scrum Meeting

- After Sprint Planning, the Team gets to work and meets every day for the Daily Scrum. All team members, working on the Sprint Backlog, need to attend and should stand-up to help keep the meeting short (no longer than 15 minutes).
- The Daily Scrum is not a status report. This ceremony aligns the team and helps maintain open communication.
- During the Daily Scrum, each Team member answers three question:
  - What did I do yesterday that helped the Team meet the Sprint Goal?
  - What will I do today to help the Team meet the Sprint Goal?
  - Do I see any impediment that prevents me or the Team from meeting the Sprint Goal? (Risk and issue management)

Roadmap (Iteration Planning)

- When we describe the Vision, it is presented as time-independent.
- To set priorities and plan for implementation, we need a perspective that includes time – this is the purpose of the Roadmap (see an example below).
- The Roadmap consists of a series of planned release (or iteration) dates, each of which has a prioritized feature (or user story) set.
- For time-boxed iterations the workload (Velocity) should be similar.





User Story Mapping (Release Planning)

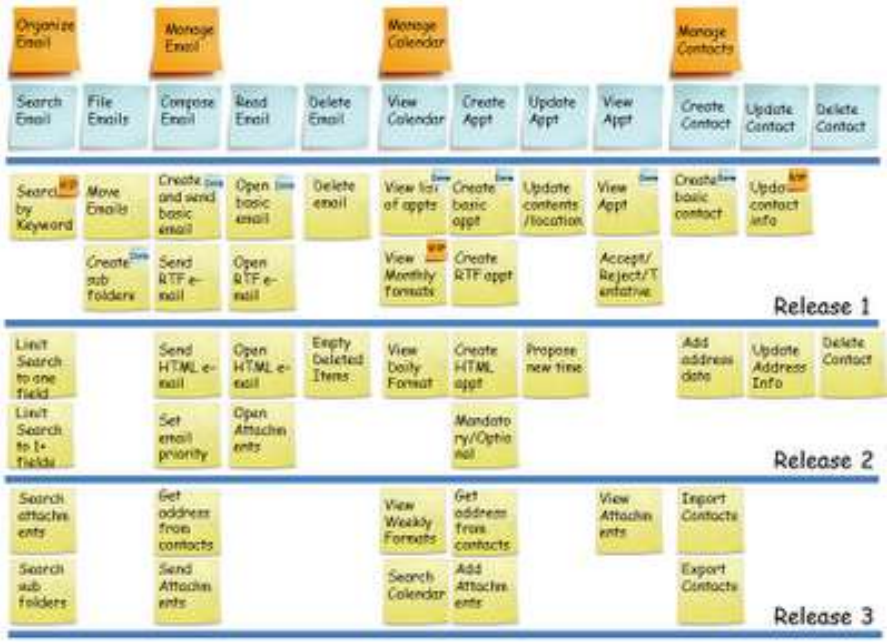
- User Story Mapping is an Agile technique that became popular about 10 years ago. It is used for a) discovering user stories, and b) planning stories by releases/iterations.
- A Story Map organizes user stories according to a narrative flow, called *Backbone*, that presents the big picture of the product. The Backbone can be of 1 level (Epic) or 2 levels (Epic, Feature).
- The horizontal axis corresponds to the major user activities, and the vertical axis corresponds to the needs of the individual users, organized by the activities.

Application Backbone

Epics

Features

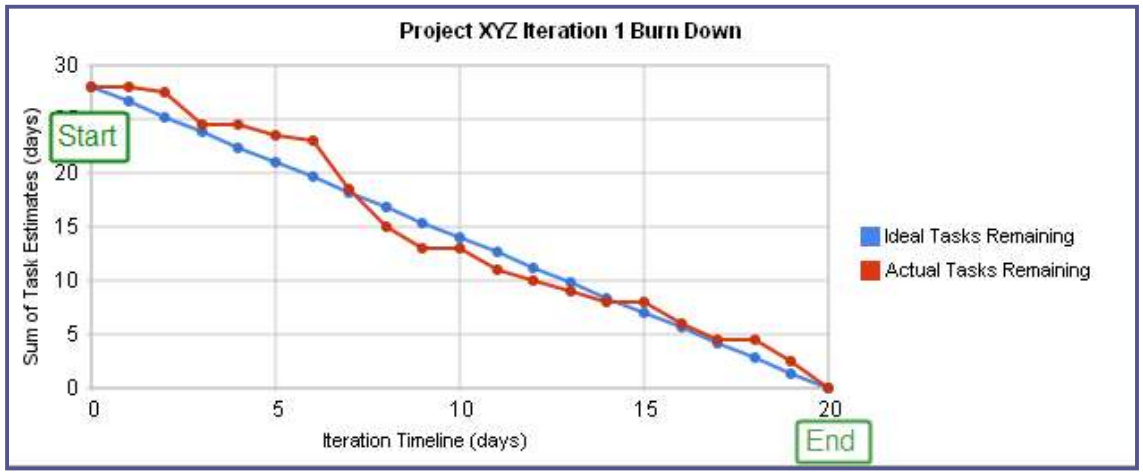
User Stories organized by Epics and releases (or Iterations)



Sprint Progress Tracking with Burndown Chart

- A Burndown chart is a graphical representation of work left to do versus time.
- The Sprint Burndown chart makes the work of the Team visible.
- The outstanding work effort (measured in story points) is shown on the vertical axis, with time shown along the horizontal axis.
- It is useful for predicting when all the work will be completed.
- The Burndown charts are commonly used on agile projects for tracking Sprint progress.
- However, Burndown charts can be applied to any project containing measurable progress over time.

Burndown Chart Example



Kanban Board for Tracking Iteration Progress

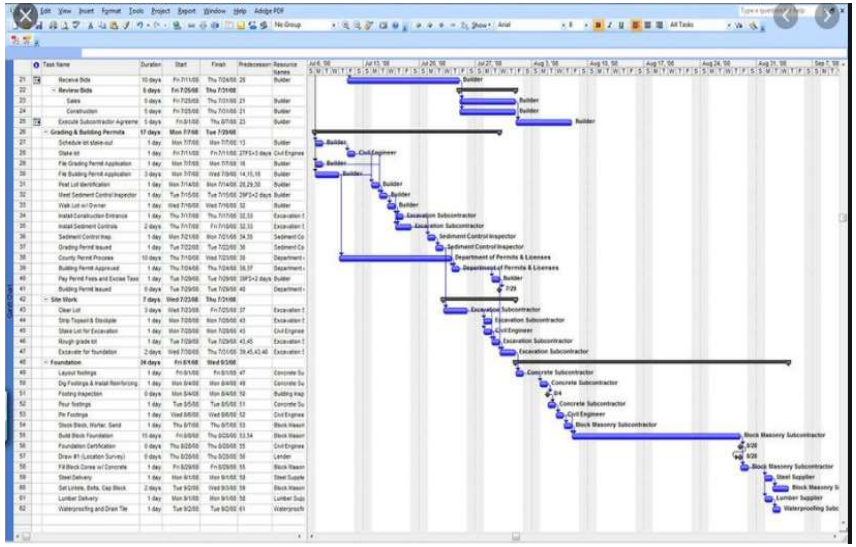
- A Kanban board is an Agile project management tool. It helps visualize work items in progress during a given iteration, limit work-in-progress, and maximize efficiency.
- Kanban boards complement the Burndown charts and visually depict work at various stages of a process using cards to represent work items and columns to represent each stage of the software process.
- Cards represent work items that can be user stories, tasks, bugs, etc.
- Cards are moved from left to right to show progress and to help coordinate team members performing the work.

Kanban Board Example

Traditional vs. Agile Project Planning Techniques

Traditional Projects

A Project Plan document is used for both – project planning and progress tracking.

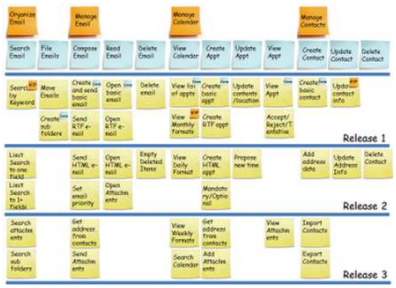


Agile Projects

Project Planning

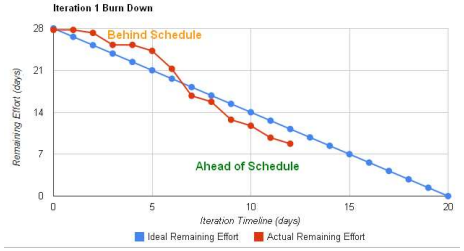
Roadmap

User Story Map

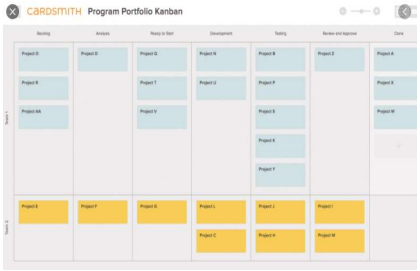


Progress Tracking

Burndown Chart



Kanban Board



Scrum Benefits

- The product functionality is broken down into a set of manageable and understandable chunks delivered as iterations.
- Unstable requirements do not hold up progress.
- All team members have visibility of everything and consequently the team's communication is improved.
- Customers see on-time delivery of increments (iterations) and gain feedback on how the product works.
- Trust between customers and developers is established and a positive culture is created in which everyone expects the project to succeed.

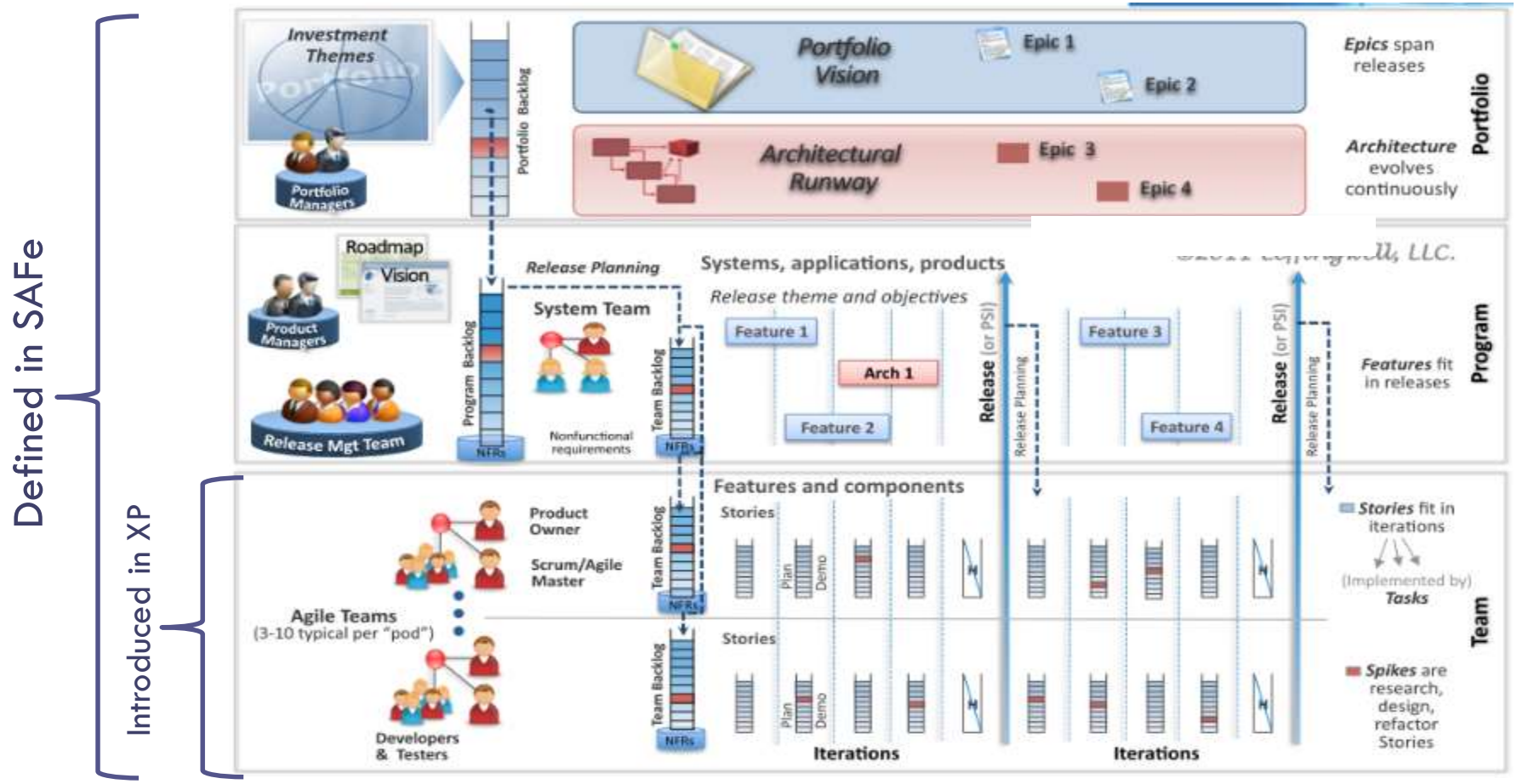
Agile Development has been adopted by large organizations and evolved into the Scaled Agile Framework (SAFe) discussed in the next slides.

Scaled Agile Framework (SAFe)

- Over the years, the original XP methodology has evolved into a SAFe Framework that includes three levels – Team, Program, Portfolio.
- **Scaled Agile Framework** (**SAFe**) is a set of organization and workflow patterns intended to guide enterprises in scaling lean and agile practices.
- In contrast with the original XP Agile methodology, SAFe promotes alignment, collaboration, and delivery across multiple Agile teams.

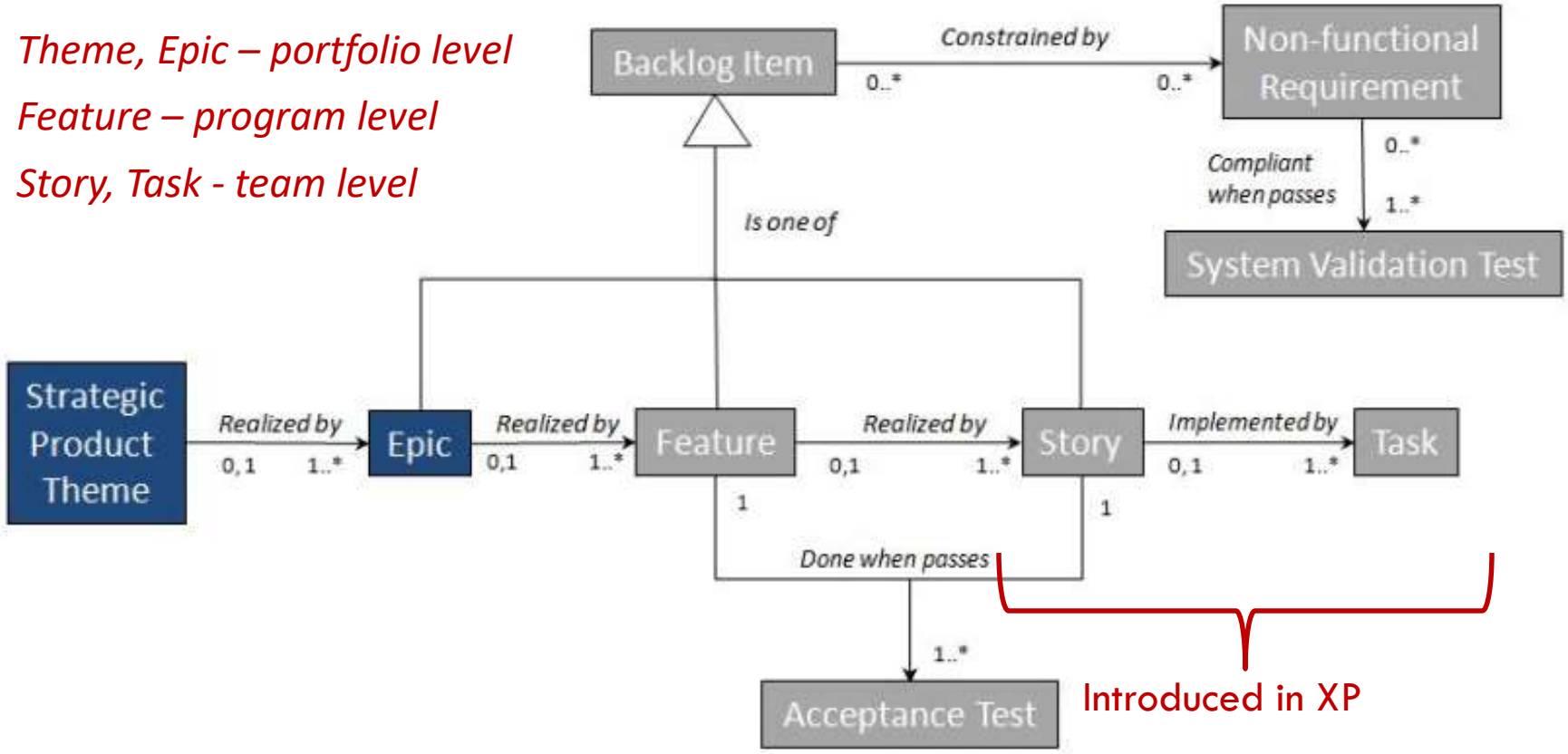
https://www.scaledagileframework.com/

SAFe: Scaled Agile Framework



SAFe: Types of Agile Requirements

- *Theme, Epic – portfolio level*
- *Feature – program level*
- *Story, Task - team level*



Portfolio Level → Program Level → Team Level

Ten SAFe Principles

1. Take an economic view
2. Apply systems thinking
3. Assume variability; preserve options
4. Build incrementally with fast integrated learning cycles
5. Base milestones on objective evaluation of working systems
6. Visualize and limit work-in-progress, reduce batch sizes, and manage queue lengths
7. Apply cadence (timing), synchronize with cross-domain planning
8. Unlock the intrinsic motivation of knowledge workers
9. Decentralize decision-making
10. Organize around value

SAFe Portal:
https://www.scaledagileframework.com/

Choice Between Agile and Plan-driven Methods

Most projects include elements of both plan-driven and agile processes. Deciding on the balance depends on:

- Is it important to have a very detailed specification and design before moving to implementation? If so, you probably need to use a plan-driven approach.
- Is an incremental delivery strategy, where you deliver the software to customers and get rapid feedback from them, realistic? If so, consider using agile methods.
- How large is the system that is being developed? Agile methods are most effective when the system can be developed with a small co-located team who can communicate informally. This may not be possible for large systems that require larger development teams so a plan-driven approach may have to be used.
- Plan-driven approaches may be required for systems that require a lot of analysis before implementation (e.g., real-time system with complex timing requirements).

Choice Between Agile and Plan-driven Methods

- What is the expected system lifetime?
  - Long-lifetime systems may require more design documentation, in order to communicate the original intentions of the system developers to the support team.
- How is the development team organized?
  - If the development team is distributed or if part of the development is being outsourced, then you may need to develop design documents, to enhance communications across the development teams.
- Are there cultural or organizational issues that may affect the system development?
  - Traditional engineering organizations have a culture of plan-driven development, as this is the norm in engineering.
- What type of system is being developed? Is the system subject to external regulation?
  - If a system has to be approved by an external regulator (e.g. the FAA approve software that is critical to the operation of an aircraft) then you will probably be required to produce detailed documentation as part of the system safety case.

Part II.
Project Management Practices

General Project Management Overview

Project management, as defined in the CMMI, includes two process areas:

Project Planning

- The purpose of project planning is to identify the scope of the project, estimate the work involved, and create a project schedule.
- Project planning begins with requirements that define the software to be developed.
- The project plan is then developed to describe the tasks that will lead to completion.

Project Monitoring and Control

- The purpose of Project Monitoring and Control (PMC) is to provide an understanding of the project's progress so that appropriate corrective actions can be taken when the project's performance deviates significantly from the plan.
- If the project deviates from the plan, then the project manager can take action to correct the problem.
- Project monitoring and control involves status meetings to gather status from the team. When changes need to be made, change control is used to keep the products up to date.

CMMI: Project Planning Goals and Practices

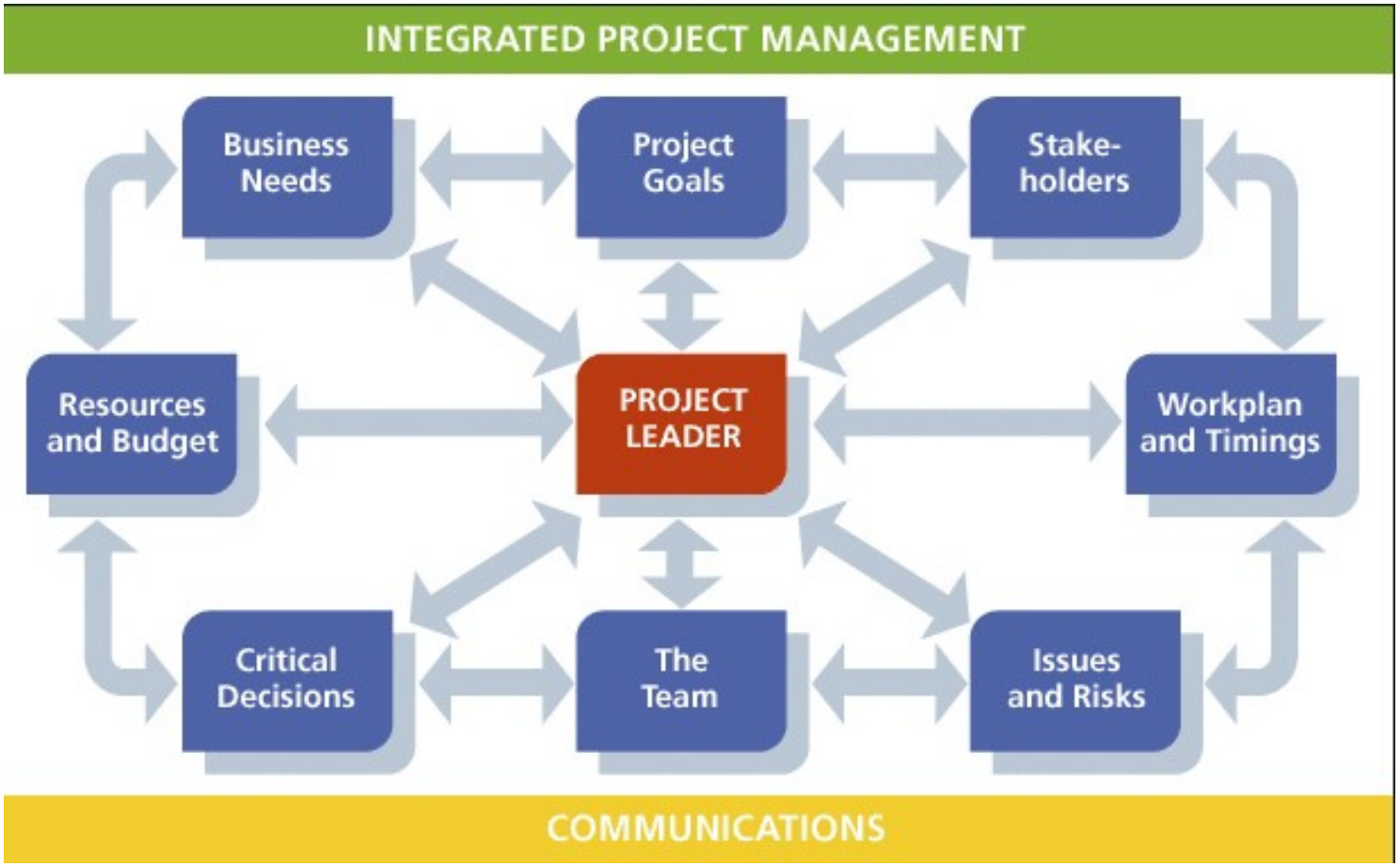
Process Goals	Specific Practices
SG 1 Establish Estimates	SP 1.1 Estimate the Scope of the Project
	SP 1.2 Establish Estimates of Work Product and Task Attributes
	SP 1.3 Define Project Lifecycle Phases
	SP 1.4 Estimate Effort and Cost
SG 2 Develop a Project Plan	SP 2.1 Establish the Budget and Schedule
	SP 2.2 Identify Project Risks
	SP 2.3 Plan Data Management
	SP 2.4 Plan the Project's Resources
	SP 2.5 Plan Needed Knowledge and Skills
	SP 2.6 Plan Stakeholder Involvement
	SP 2.7 Establish the Project Plan
SG 3 Obtain Commitment to the Plan	SP 3.1 Review Plans That Affect the Project
	SP 3.2 Reconcile Work and Resource Levels
	SP 3.3 Obtain Plan Commitment

CMMI: Project Monitoring and Control Practices

Process Goals	Specific Practices
SG 1 Monitor the Project Against the Plan	SP 1.1 Monitor Project Planning Parameters (performance, cost, deviations)
	SP 1.2 Monitor Commitments
	SP 1.3 Monitor Project Risks
	SP 1.4 Monitor Data Management
	SP 1.5 Monitor Stakeholder Involvement
	SP 1.6 Conduct Progress Reviews
	SP 1.7 Conduct Milestone Reviews
SG 2 Manage Corrective Action to Closure	SP 2.1 Analyze Issues
	SP 2.2 Take Corrective Action
	SP 2.3 Manage Corrective Actions

The Project Manager's Areas of Concern

On large projects, a project manager is responsible for many aspects of project delivery:



Project Plans

- In a plan-driven development project, a project plan defines necessary resources, the work breakdown into project tasks, schedule, etc.
- The details of project plans vary depending on the project context and normally include the following sections:

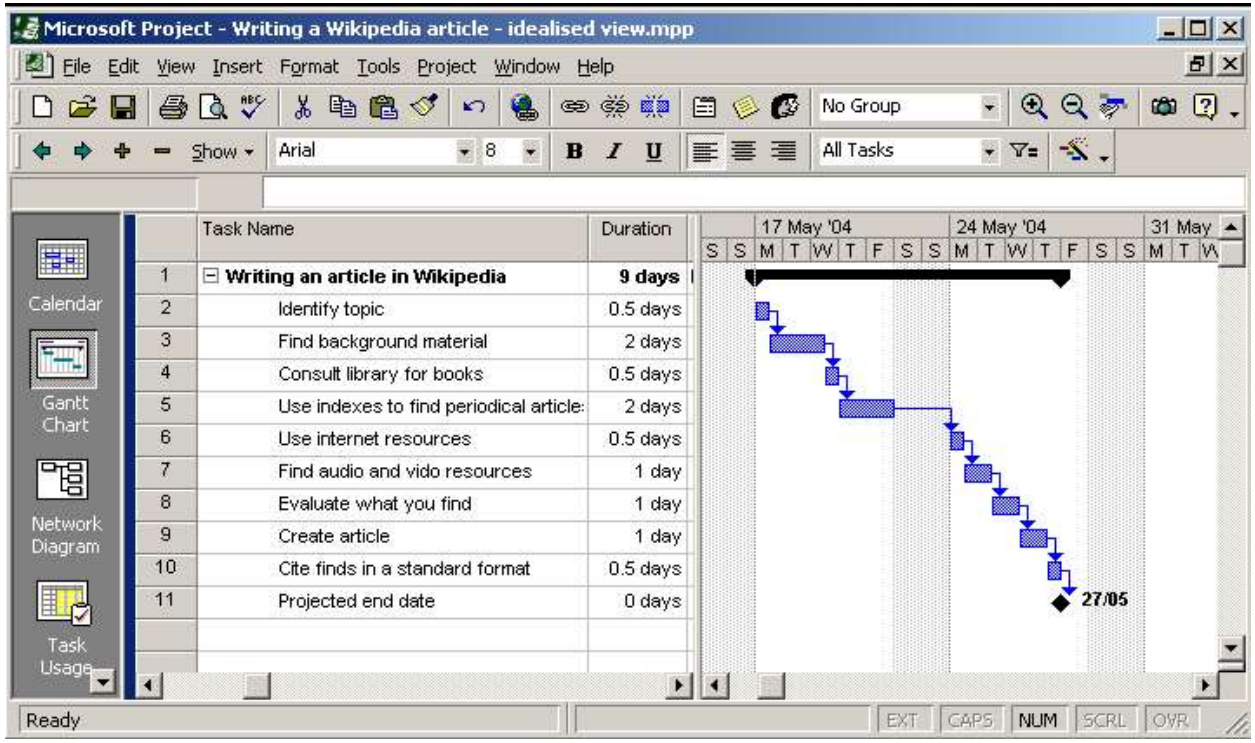
Plan Section	Description
Introduction	Describes the objectives of the project.
Project Organization	Describes the organization of the development team and project stakeholders.
Risk Analysis	Describes possible risks and mitigation strategies.
Required Resources	Specifies the resources necessary to deliver the project.
Monitoring and Reporting	Defines the management reporting procedure, PMO practices.
Communications Plan	A communications plan is a road map for getting your message across to your project stakeholders.
Work Breakdown	Defines the project phases, tasks, resources, and milestones.
Project Schedule	Defines the dependencies between the tasks and time required to complete each task.

Project Initiation Document

MS Project

Project Plan Example

- Microsoft Project is a common project management software program. It is designed to assist a project manager in developing a plan, assigning resources to tasks, tracking progress, managing the budget, and analyzing workloads.
- A software project plan also includes milestones. **Milestones** are specific points along a project timeline used in project management to better track progress and for management reporting.

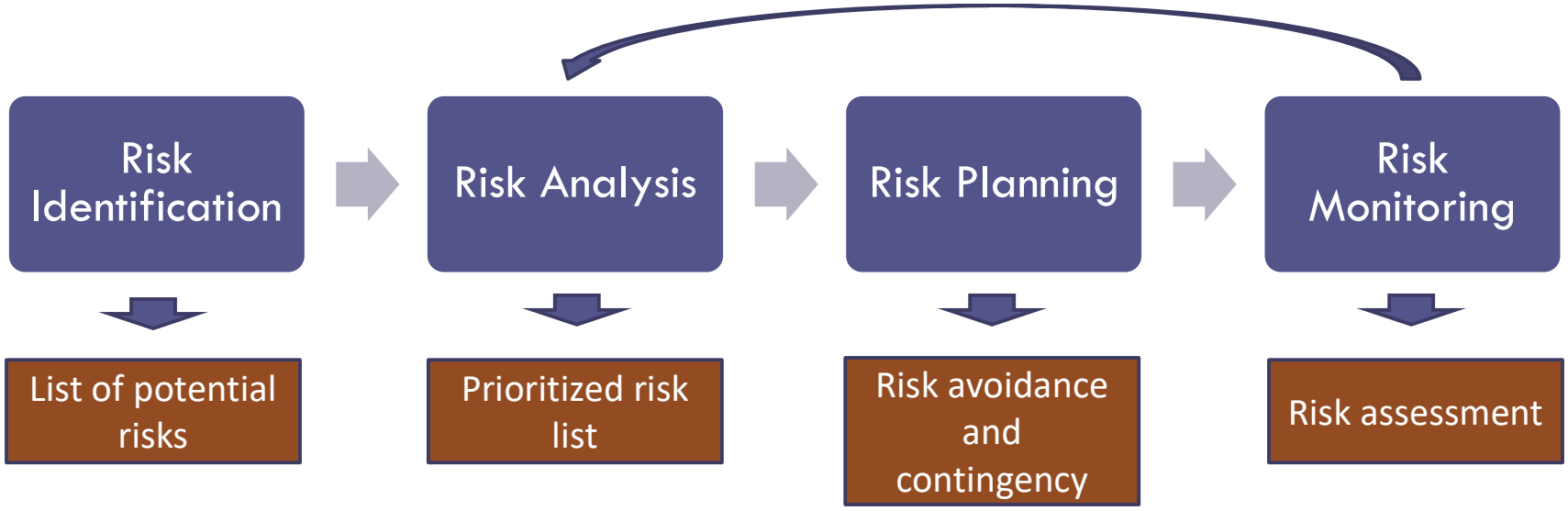


Risk Management

Risk management is one of the most important jobs for a project manager. Commonly, risks are classified by what they can affect:

- Project risks
- Product risks
- Business risks

Risk management includes four stages:



RAID Log

- The acronym RAID stands for *Risks*, *Assumptions*, *Issues* and *Dependencies*.
- RAID is a common Risk Management technique on large programs.

Categories	Description
Risks	Events that will have a negative impact on your project if they occur. Risk refers to the combined likelihood that an vent will occur and the impact on the project if it does occur. If the likelihood of the event happening and impact to the project are both high, you identify the event as a risk. The log includes descriptions of each risk, full analysis and a plan to manage them.
Assumptions	Any factors that you are assuming to be in place that will contribute to the successful result of your project. The log includes details of the assumption, the reason it is assumed, and the action needed to confirm whether the assumption is valid.
Issues	Something that is going wrong on your project and needs managing. Failure to manage issues may result in a poor delivery or even failure. The log includes descriptions of each issue, its impact, its seriousness and actions needed to contain and remove it.
Dependencies	Any event or work that is either dependent on the result of your project, or on which your project will depend. The log captures whom you are dependent on, what they should deliver and when. It may also include who is dependent on you.

Agile Methods – Key Points

- Agile methods are iterative development methods that focus on reducing process overheads and documentation and on incremental software delivery. They involve customer representatives directly in the development process.
- The decision on whether to use an agile or plan-drive approach to development should depend on the type of software being developed, the capabilities of the development team, and the culture of the organization.
- In practice, most projects include elements of plan-driven and agile processes.
- Agile development practices include user stories (as requirements), pair programming, re-factoring, continuous integration, and test-first development.
- A particular strength of extreme programming is the development of automated tests before a program feature is created. All tests must successfully execute when an increment is integrated into a system.
- The Scrum method is an agile method that provides a project management framework. It is centred round a set of sprints, which are fixed time periods when a system increment is developed.

Project Management – Key Points

- Good software project management is essential for software projects to be able to deliver on time and within budget.
- Risk management is an important part of a project manager's responsibilities. It involves identifying risks and making plans to avoid the risks' consequences.
- Plan-driven development is organized around a complete project plan that defines the project tasks , milestones and schedule , and allocates resources.
- A project milestone is an outcome of a critical task or a group of tasks. Milestones are used for tracking the project progress and management reporting.
- Agile planning involves the entire team. The plan is developed incrementally, and if deviations arise, it is adjusted to avoid delaying of a release.

Agile Methods – Exercises

- Explain why the rapid (agile) delivery and deployment of new systems is often more important to businesses than the detailed functionality of these systems.
- Name a few principles, underlying agile methods, that lead to accelerated development and deployment of software.
- Extreme programming captures software requirements as user stories, with each story written on a card. Discuss the advantages and disadvantages of this approach to requirements development.
- Explain why test-first development can help the programmer to develop a better understanding of the system requirements.
- Compare and contrast the Scrum and conventional plan-driven project management practices.
- Explain why it can be necessary to introduce some plan-driven documentation and management practices when scaling agile methods to large projects developed by distributed teams.

Project Management – Exercises

1. What are the two process areas, related to project management, defined in the CMMI?
2. Name a few areas of a project manager's responsibility.
3. Explain what a project plan should include.
4. Explain what milestones are and why it is important to include them in a project plan.
5. Explain why the process of project planning is iterative and why a plan must be continually reviewed during a software project.
6. Name the conventional stages of risk management.
7. Explain what the RAID log is.