



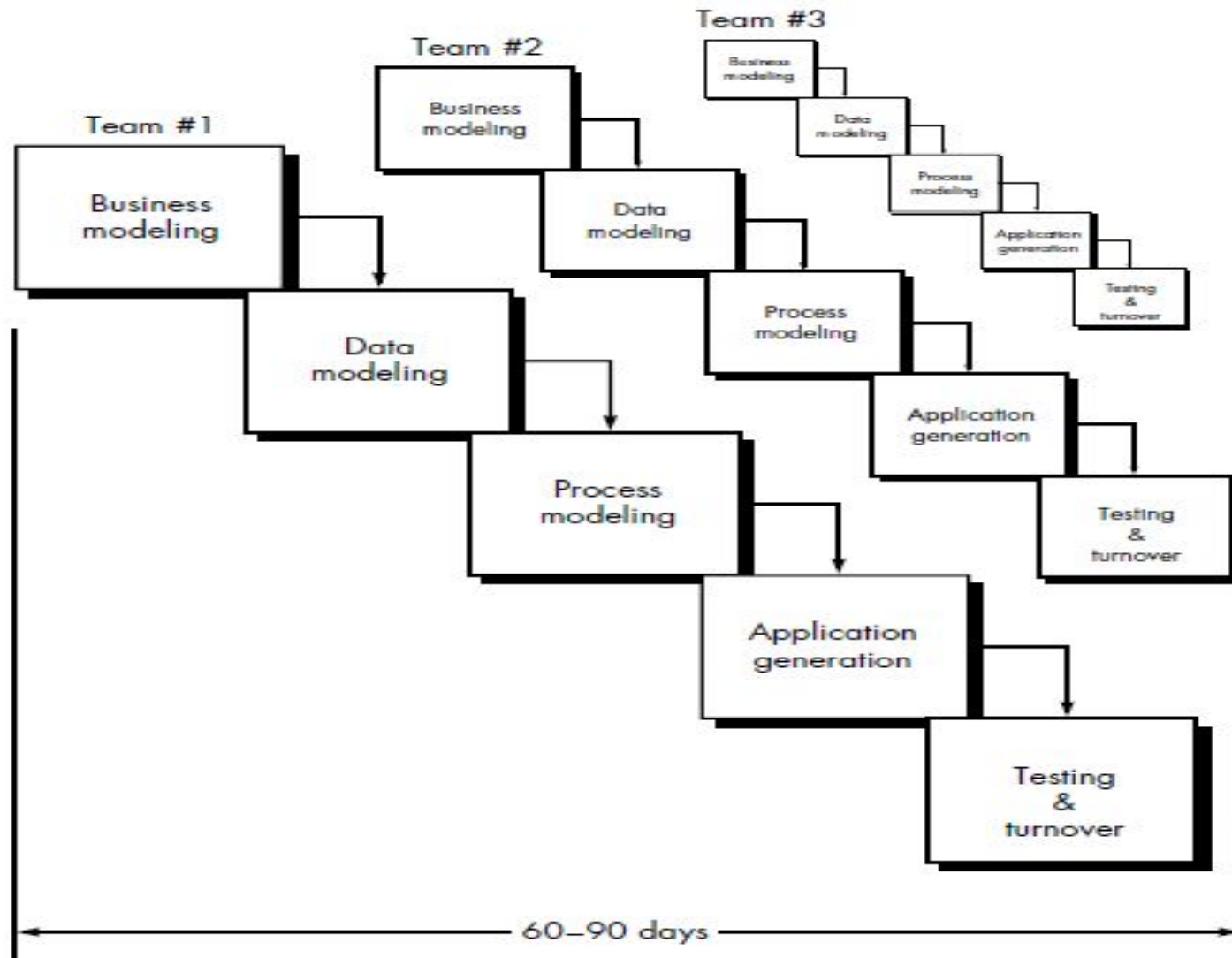
Agile

Momna Zaneb

Rapid Application Development (RAD)

- The RAD Model is an incremental software development process model that emphasizes an extremely short development cycle.
- If requirements are well understood and project scope is constrained, the RAD process enables a development team to create a “fully functional system” within very short time periods (e.g., 60 to 90 days).
- Used primarily for information systems applications.

Rapid Application Development (RAD) Model



Rapid Application Development (RAD)

- Business modeling: The information flow is identified between various business functions.
- Data modeling: Information gathered from business modeling is used to define data objects that are needed for the business.
- Process modeling: Data objects defined in data modeling are converted to achieve the business information flow to achieve some specific business objective. Description are identified and created for CRUD of data objects.
- Application generation: Automated tools are used to convert process models into code and the actual system.
- Testing and turnover: Test new components and all the interfaces.

Advantages of the RAD model:

- Reduced development time.
- Quick initial reviews occur
- Encourages customer feedback

Disadvantages of RAD model:

- Requires highly skilled developers/designers.
- High dependency on modeling skills.
- Inapplicable to cheaper projects as cost of modeling and automated code generation is very high.

When to use RAD model:

- RAD should be used when there is a need to create a system that can be modularized in 2-3 months of time.
- It should be used if there's high availability of designers for modeling and the budget is high enough to afford their cost along with the cost of automated code generating tools.
- RAD SDLC model should be chosen only if resources with high business knowledge are available and there is a need to produce the system in a short span of time (2-3 months).



Week 2

- Agile manifesto
- XP
- Scrum
- Kanban
- DevOps

Agile Methodology

Agile is an **incremental, iterative approach to project management and software development** that helps teams deliver value to their customers faster and with fewer headaches. Instead of betting everything on a "big bang" launch, an agile team delivers work in small, but consumable, increments.



Agile Manifesto

Individuals and interactions over processes and tools

Working software over comprehensive documentation

Customer collaboration over contract negotiation



Responding to change over following a plan



12 Principles of Agile



- Our highest priority is to satisfy the customer through early and continuous delivery of valuable software.
- Welcome changing requirements, even late in development. Agile processes harness change for the customer's competitive advantage.
- Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale.
- Business people and developers must work together daily throughout the project.

12 Principles of Agile

- Build projects around motivated individuals.
Give them the environment and support they need,
and trust them to get the job done.
- The most efficient and effective method of
conveying information to and within a development
team is face-to-face conversation.
- Working software is the primary measure of progress.
- Agile processes promote sustainable development.
The sponsors, developers, and users should be able
to maintain a constant pace indefinitely.

12 Principles of Agile

- Continuous attention to technical excellence and good design enhances agility.
- Simplicity--the art of maximizing the amount of work not done--is essential.
- The best architectures, requirements, and designs emerge from self-organizing teams.
- At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly.



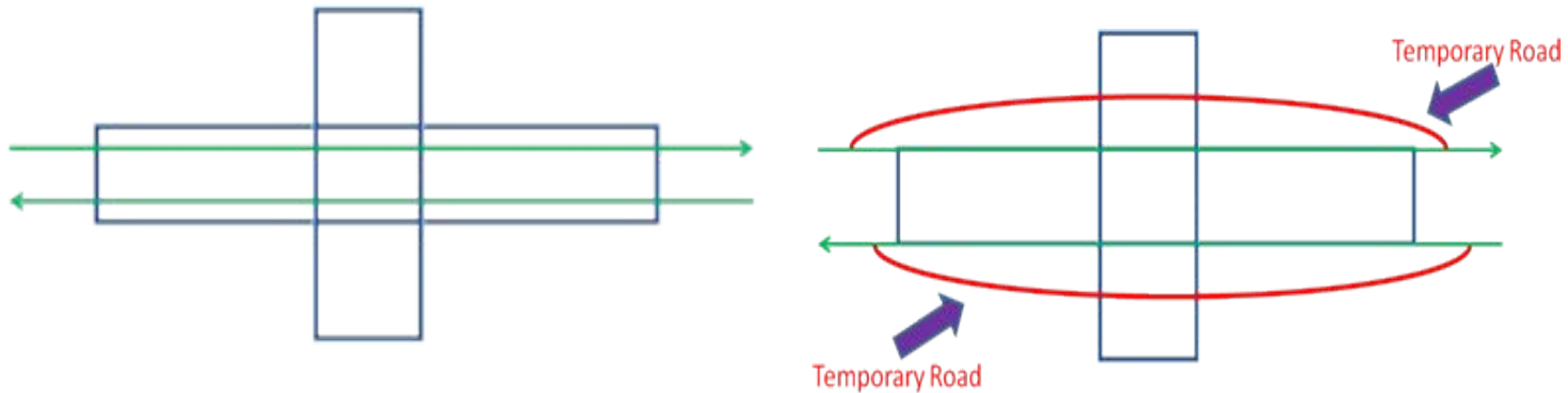
EXAMPLE 1

FLYOVER

CONSTRUCTION

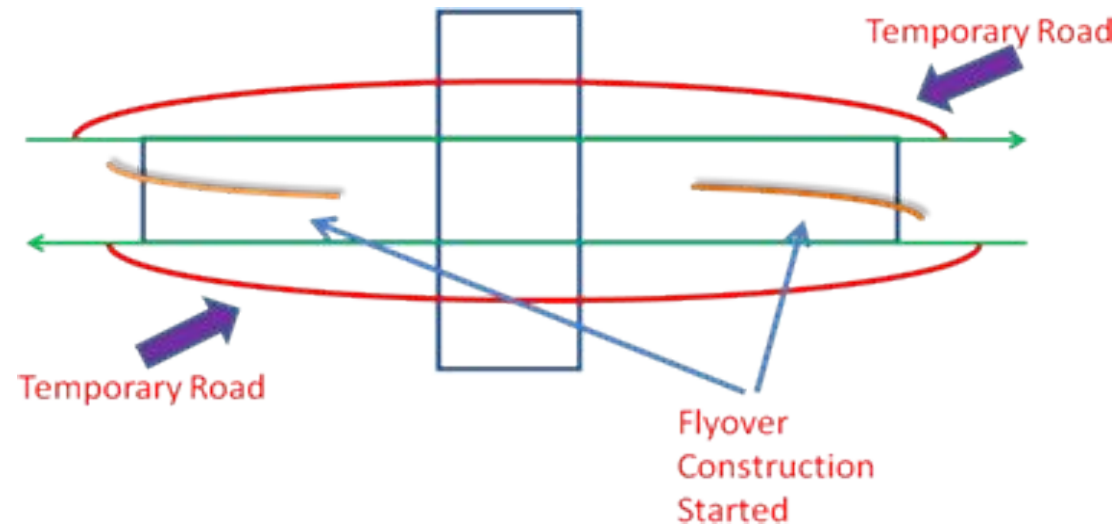
EXAMPLE- CONSTRUCTION OF FLYOVER TRADITIONAL APPROACH

- Step1: A temporary road is constructed next to the main crossroad.



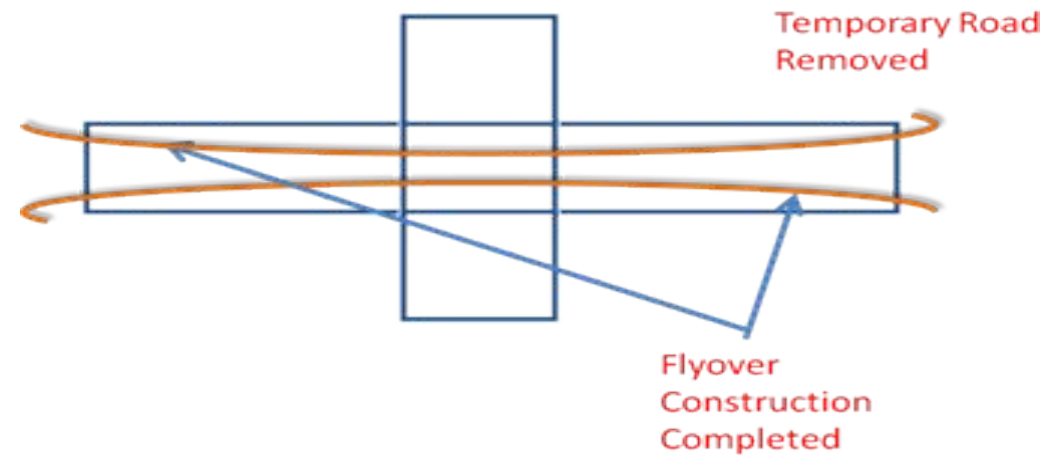
EXAMPLE- CONSTRUCTION OF FLYOVER TRADITIONAL APPROACH

- Step 2: Flyover construction work is started from both directions.



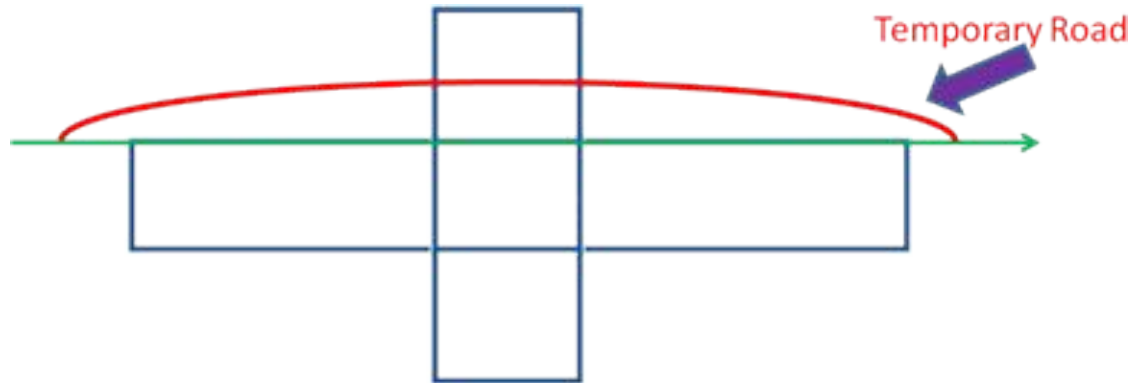
EXAMPLE- CONSTRUCTION OF FLYOVER TRADITIONAL APPROACH

- Step 3: After 18 months or so, the work is completed and the temporary road is removed.



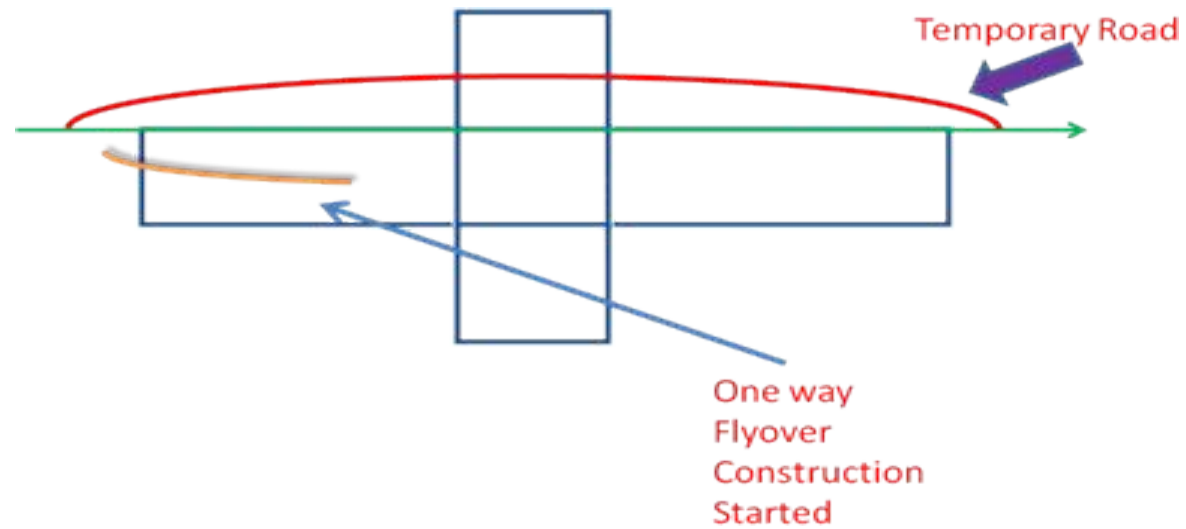
EXAMPLE- CONSTRUCTION OF FLYOVER Agile Approach

- Step 1: A one-way temporary road is constructed



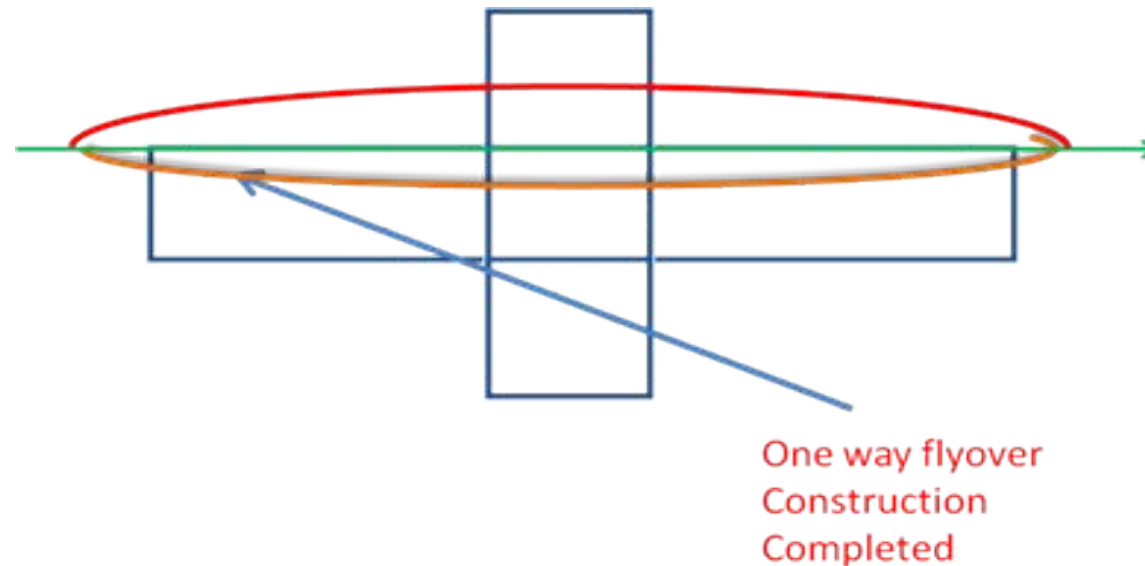
EXAMPLE- CONSTRUCTION OF FLYOVER Agile Approach

- Step 2: A one-way flyover construction is started.



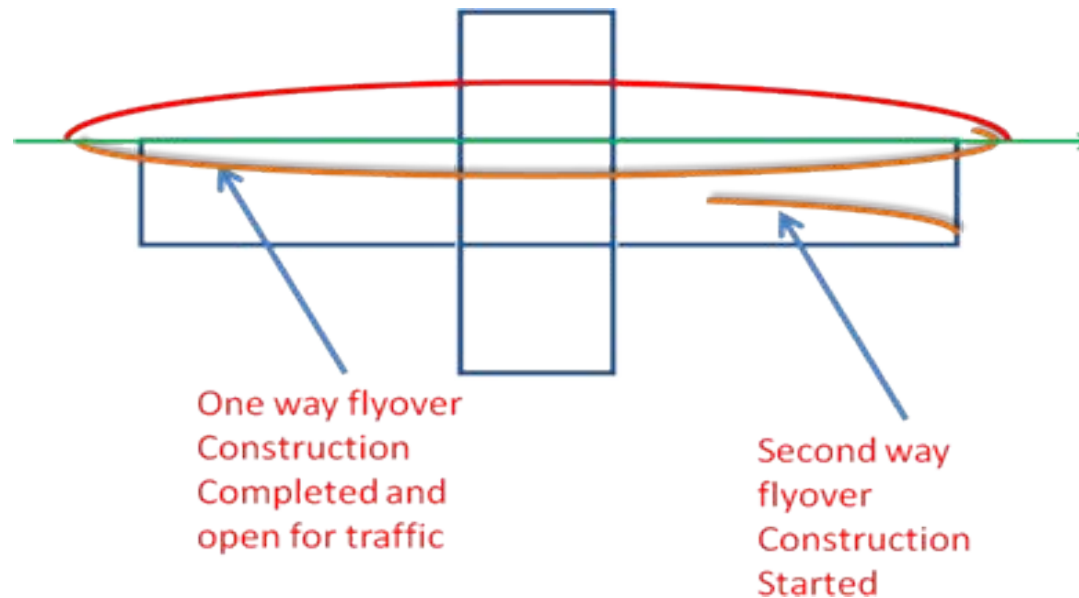
EXAMPLE- CONSTRUCTION OF FLYOVER Agile Approach

- Step 3: The one-way flyover construction is completed and opens for two-way traffic. The overall traffic is still slow, but much better than without any flyovers. Here the end customer (commuter) is using what we call a product of incremental delivery.



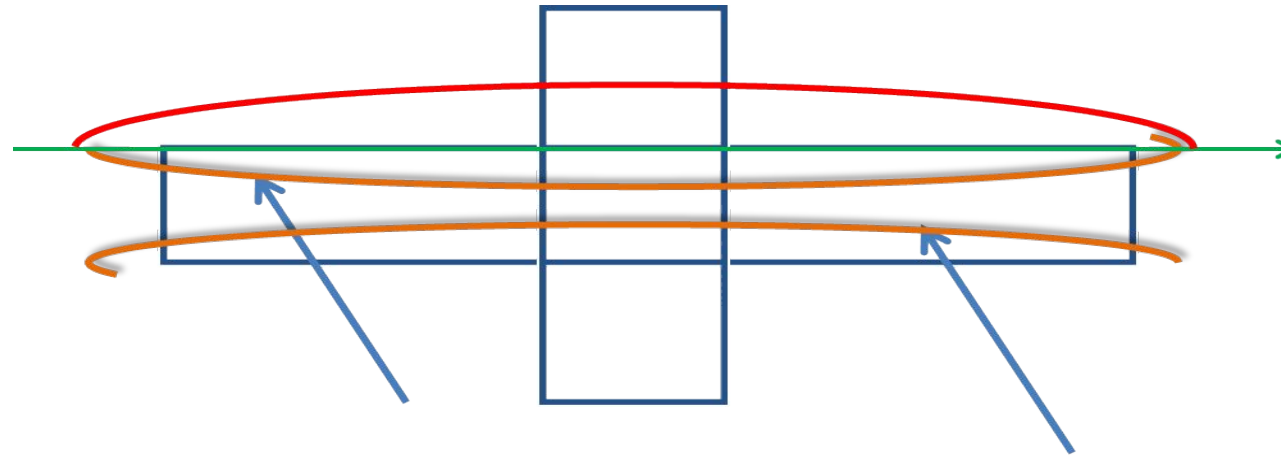
EXAMPLE- CONSTRUCTION OF FLYOVER Agile Approach

- Step 4: The same temporary road is used along with the original road.
- Step 5: The second-direction flyover construction starts.



EXAMPLE- CONSTRUCTION OF FLYOVER Agile Approach

- Step6: The second-direction flyover construction finishes.





Both way flyover construction is completed and open for traffic.

RESULTS

- This incremental delivery helped customers use the project (the flyover) in nine months instead of waiting twice that long (plus some inevitable delays).

Benefits:

- The completion of the one-way flyover was much faster (completed in approximately nine months).
- With a divider on the flyover, the same flyover was marked for two-way traffic, which managed to reduce the load on this crossroad.
- The traffic situation improved at the junction, though it's still slow at peak hours.
- With two-way traffic on the flyover, there was no need to construct another temporary road.
- The second-way flyover was constructed without any extreme pressure, as one flyover was already operational, which reduced the pressure.
- The end user got to use the flyover in half the time (9 months instead of 18)



EXAMPLE 2

SOFTWARE APPLICATION

EXAMPLE 2 SOFTWARE APPLICATION

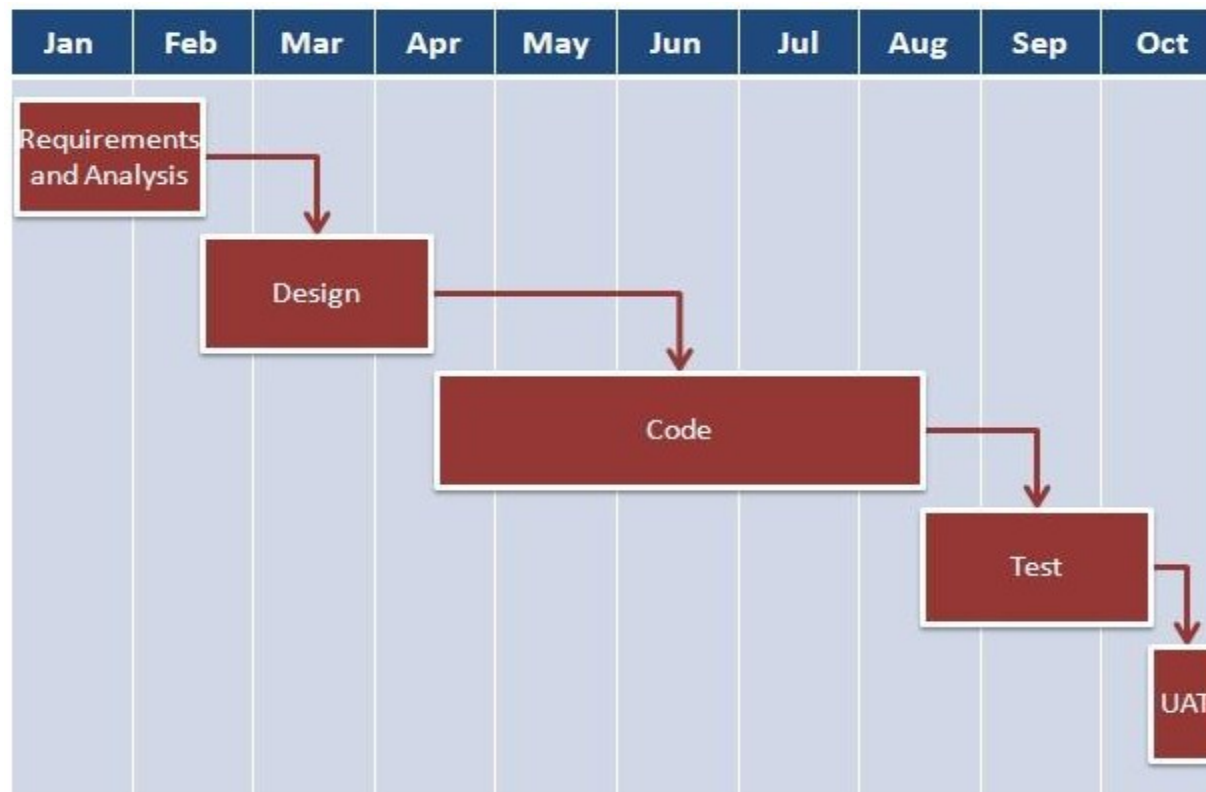
- Google is working on project to come up with a competing product for MS Word, that provides all the features provided by MS Word and any other features requested by the marketing team. The final product needs to be ready in 10 months of time.
- Let us see how this project is executed in traditional and Agile methodologies.

EXAMPLE 2 SOFTWARE APPLICATION

- In traditional Waterfall model:
 - At a high level, the project teams would spend 15% of their time on gathering requirements and analysis (1.5 months)
 - 20% of their time on design (2 months)
 - 40% on coding (4 months) and unit testing
 - 20% on System and Integration testing (2 months)
 - At the end of this cycle, the project may also have 2 weeks of User Acceptance testing by marketing teams.
 - In this approach, the customer does not get to see the end product until the end of the project, when it becomes too late to make significant changes.

EXAMPLE 2 SOFTWARE APPLICATION

- The image below shows how these activities align with the project schedule in traditional software development.

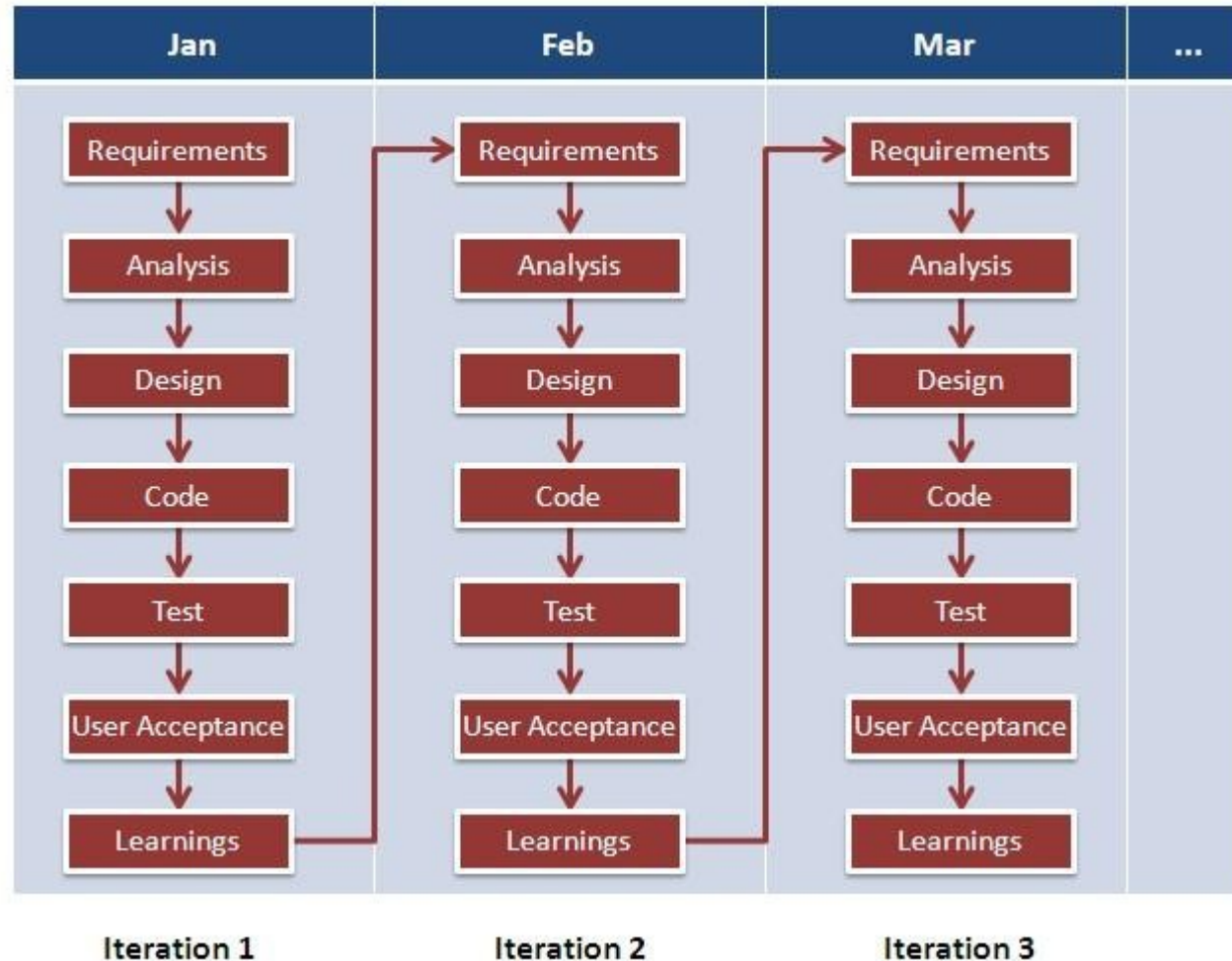


EXAMPLE 2 SOFTWARE APPLICATION

- In the **Agile methodology**, each project is broken up into several "iterations".
- All iterations should be of the same time duration (between 2 to 8 weeks).
- At the end of each iteration, a working product should be delivered.
- In simple terms, in the Agile approach the project will be broken up into 10 releases (assuming each iteration is set to last 4 weeks).
- Rather than spending 1.5 months on requirements gathering, in Agile software development, the team will decide the basic core features that are required in the product and decide which of these features can be developed in the first iteration.
- Any remaining features that cannot be delivered in the first iteration will be taken up in the next iteration or subsequent iterations, based on priority.
- At the end of the first iterations, the team will deliver a working software with the features that were finalized for that iteration.
- There will be 10 iterations and at the end of each iteration the customer is delivered a working software that is incrementally enhanced and updated with the features that were shortlisted for that iteration.

EXAMPLE 2 SOFTWARE APPLICATION

The image below shows how these activities align with the project schedule in Agile software development.



EXAMPLE 2 SOFTWARE APPLICATION

Iteration 1



Iteration 2

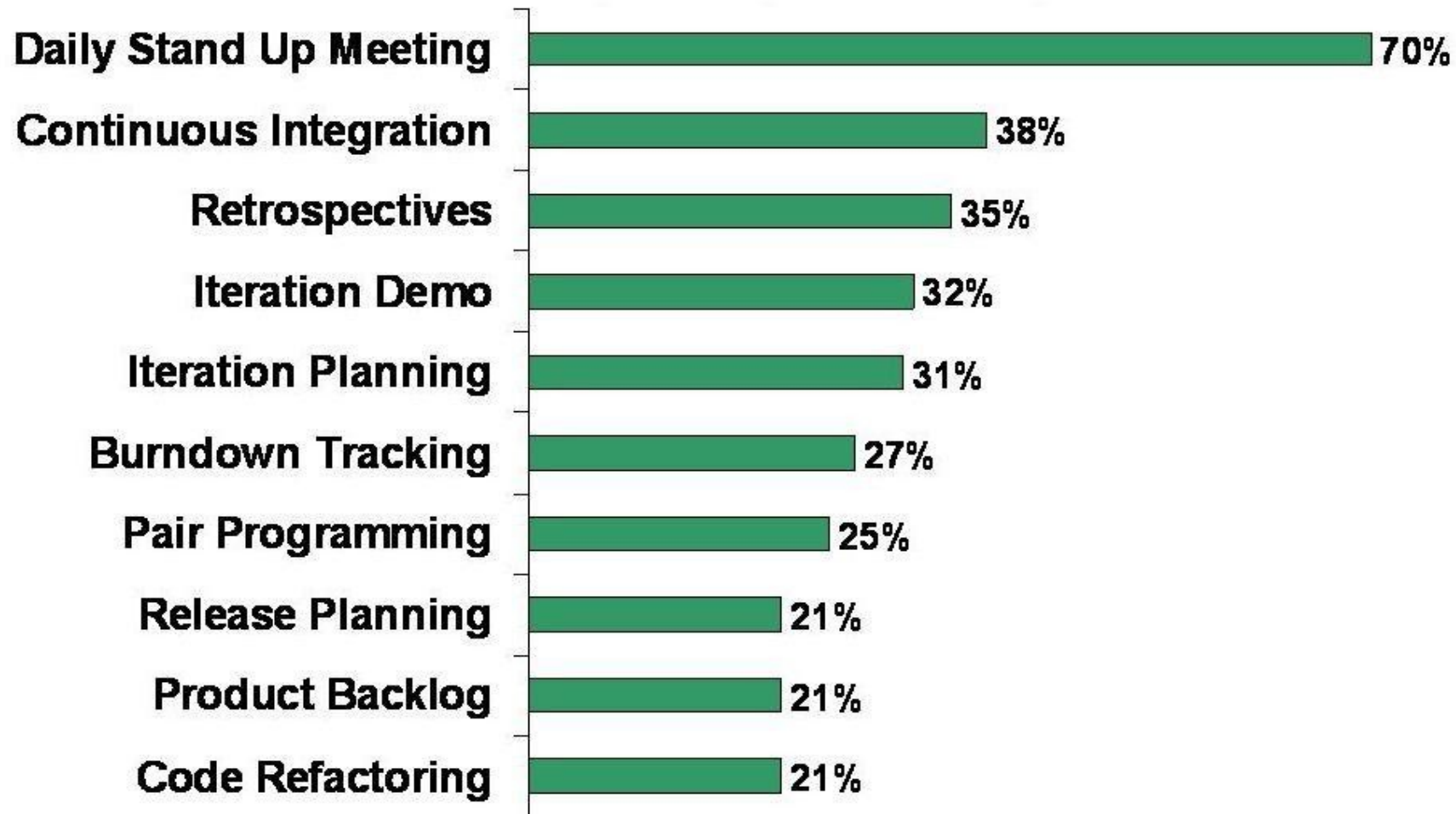


Iteration 3



TOP 10 AGILE PRACTICES

Agile Practices Easiest to Learn (Top 10 out of 30)



Source: Ambysoft Agile Practices 2009 Survey, <http://www.ambysoft.com/surveys/practices2009.html>
Copyright 2009 Ambysoft Inc.



XP

Extreme Programming (XP)

Based on set of five values that establish a foundation for all work performed as part of XP

- Communication
- Simplicity
- Feedback
- Courage
- Respect


In order to achieve effective **Communication** between software engineers and other stakeholders (e.g., to establish required features and functions for the software), XP emphasizes **close, yet informal (verbal)** collaboration **between customers and developers**, the establishment of effective metaphors for communicating important concepts, continuous feedback, and the avoidance of voluminous documentation as a communication medium.

To achieve **Simplicity**, XP restricts developers to **design only for immediate needs**, rather than consider future needs. The intent is to create a simple design that can be easily implemented in code). If the design must be improved, it can be refactored⁴ at a later time



Extreme Programming (XP)

Feedback is derived from **three sources: the implemented software** itself, **the customer**, and other **software team** members. By designing and implementing an effective **testing strategy** the software (via test results) provides the **agile team with feedback**. XP makes use of the **unit test as its primary testing tactic**. As each class is developed, the team develops a unit test to exercise each operation according to its specified functionality. As an increment is delivered to a customer, the user stories or use cases that are implemented by the increment are used as a basis for acceptance tests. The degree to which the software implements the output, function, and behavior of the use case is a form of feedback. Finally, as new requirements are derived as part of iterative planning, the **team provides the customer with rapid feedback regarding cost and schedule impact**






Extreme Programming (XP)

Strict adherence to certain XP practices demands **Courage**. A better word might be **discipline**. For example, there is often significant pressure to design for future requirements. Most software teams succumb, arguing that “designing for tomorrow” will save time and effort in the long run. An agile XP team must have the discipline (courage) to design for today, recognizing that future requirements may change dramatically, thereby demanding substantial rework of the design and implemented code.

By following each of these values, the agile team inculcates **Respect** among its members, between other stakeholders and team members, and indirectly, for the software itself. As they achieve successful delivery of software increments, the team develops growing respect for the XP process.



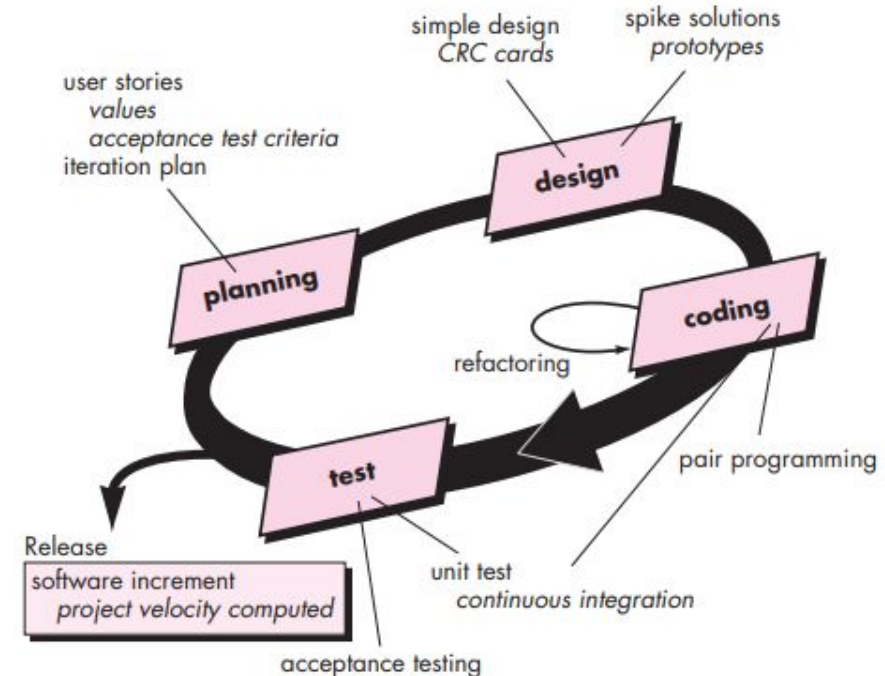
Extreme Programming (XP) Process

The XP Process Extreme Programming uses an object-oriented approach as its preferred development paradigm and encompasses a set of rules and practices that occur within the context of four framework activities:

- Planning
- Design
- Coding
- Testing

FIGURE 3.2

The Extreme Programming process



Extreme Programming (XP) Process

Planning. The planning activity (also called the **planning game**) begins with listening—a requirements gathering activity that enables the technical members of the XP team to understand the business context for the software and to get a broad feel for required output and major features and functionality.

Listening leads to the creation of a **set of “stories”** (also called **user stories**) that describe required output, features, and functionality for software to be built.

Each story (similar to use cases) is written by the customer and is placed on an index card.

The customer assigns a value (i.e., a priority) to the story based on the overall business value of the feature or function.

Members of the XP team then assess each story and assign a cost—measured in development weeks—to it.

If the story is estimated to require more than three development weeks, the customer is asked to split the story into smaller stories and the assignment of value and cost occurs again.

Extreme Programming (XP) Process

It is important to note that new stories can be written at any time. Customers and developers work together to decide how to group stories into the next release (the next software increment) to be developed by the XP team.

Once a basic commitment (agreement on stories to be included, delivery date, and other project matters) is made for a release, the XP team orders the stories that will be developed in one of three ways:

- (1) all stories will be implemented immediately (within a few weeks)
- (2) the stories with highest value will be moved up in the schedule and implemented first
- (3) the riskiest stories will be moved up in the schedule and implemented first




Extreme Programming (XP) Process

After the first project release (also called a software increment) has been delivered, the XP team computes **project velocity**.

Stated simply, **project velocity is the number of customer stories implemented during the first release.**

Project velocity can then be used to

- (1) Help estimate delivery dates and schedule for subsequent releases
 - (2) Determine whether an overcommitment has been made for all stories across the entire development project. If an overcommitment occurs, the content of releases is modified or end delivery dates are changed.
- 

Extreme Programming (XP) Process

Design

- Keep it simple
- The design provides implementation guidance for a story as it is written—nothing less, nothing more.
- The design of extra functionality (because the developer assumes it will be required later) is discouraged
- CRC (class-responsibility collaborator) cards identify and organize the object-oriented classes that are relevant to the current software increment. The CRC cards are the only design work product produced as part of the XP process.
- Spike solution for difficult design problem

Extreme Programming (XP) Process

Coding.

- After planning and design first develop unit test then code.
- Rapid feedback with instant unit testing
- Pair programming.
- Code Integration and Integration testing
- Continuous integration & Smoke Testing

Extreme Programming (XP) Process

Testing.

Wells [Wel99] states: “Fixing small problems every few hours takes less time than fixing huge problems just before the deadline.”

- **XP acceptance tests**, also called customer tests, are specified by the customer and focus on overall system features and functionality that are visible and reviewable by the customer. Acceptance tests are derived from user stories that have been implemented as part of a software release.



Scrum

Scrum

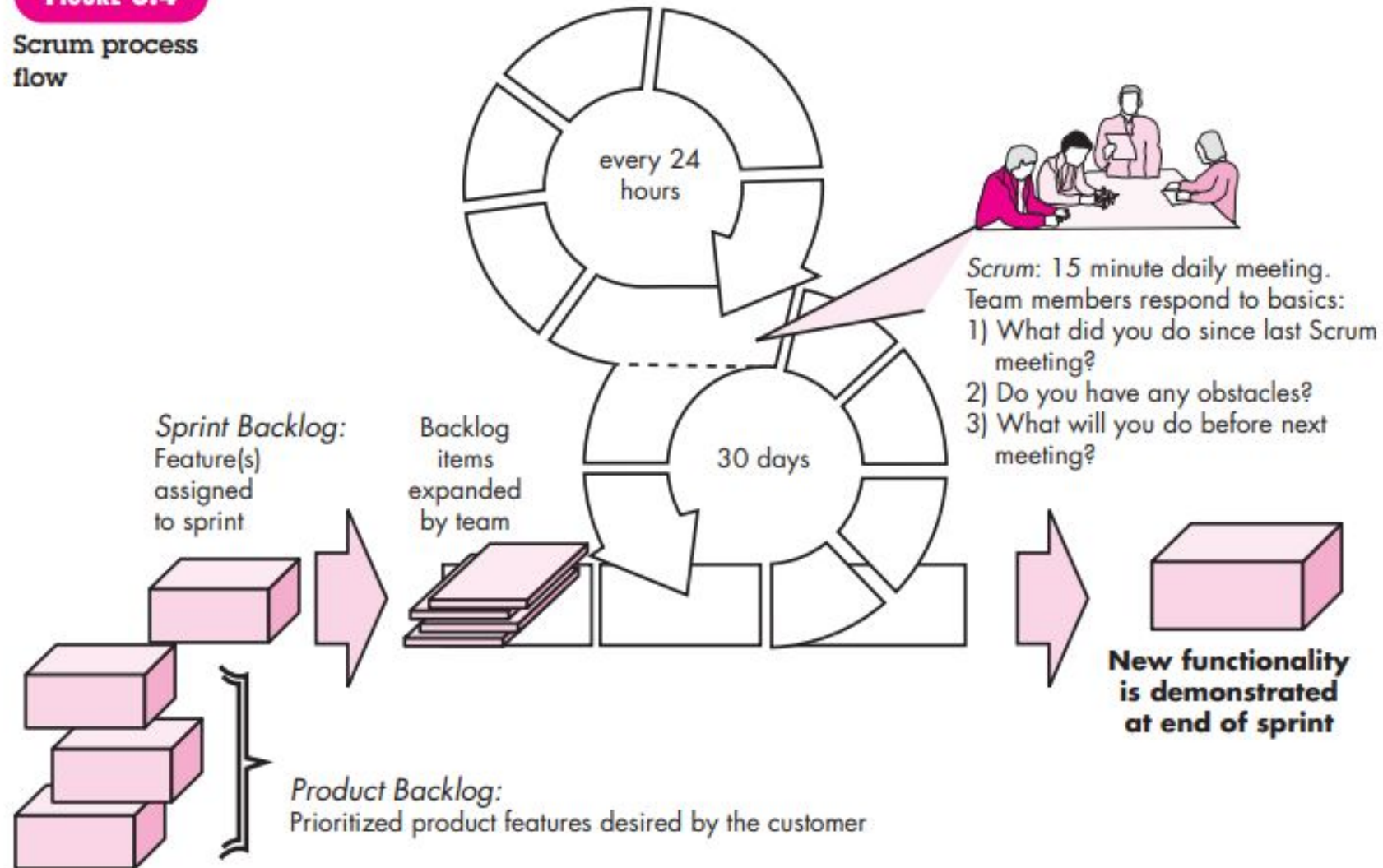
In a nutshell, Scrum requires a **Scrum Master**(aka team lead) to foster an environment where:

- 1.A **Product Owner** orders the work for a complex problem into a **Product Backlog**.
- 2.The Scrum Team turns a selection of the work into an Increment of value during a **Sprint**.
- 3.The Scrum Team and its stakeholders inspect the results and adjust for the next Sprint.
- 4.Repeat

Scrum Framework

FIGURE 3.4


Scrum process flow





Scrum-Common Terminologies

Product Backlog—a prioritized list of project requirements or features that provide business value for the customer. Items can be added to the backlog at any time (this is how changes are introduced). The product manager assesses the backlog and updates priorities as required.




Sprints - fixed length events of one month or less to create consistency. A new Sprint starts immediately after the conclusion of the previous Sprint. All the work necessary to achieve the Product Goal, including Sprint Planning, Daily Scrums, Sprint Review, and Sprint Retrospective, happen within Sprints.



Scrum-Common Terminologies

Product Backlog—a prioritized list of project requirements or features that provide business value for the customer. Items can be added to the backlog at any time (this is how changes are introduced). The product manager assesses the backlog and updates priorities as required.




Sprints - fixed length events of one month or less to create consistency. A new Sprint starts immediately after the conclusion of the previous Sprint. All the work necessary to achieve the Product Goal, including Sprint Planning, Daily Scrums, Sprint Review, and Sprint Retrospective, happen within Sprints.



Scrum-Common Terminologies

Scrum meetings — are short (typically 15 minutes) meetings held daily by the Scrum team. Three key questions are asked and answered by all team members:

- What did you do since the last team meeting?
 - What obstacles are you encountering?
 - What do you plan to accomplish by the next team meeting?
- 



Kanban

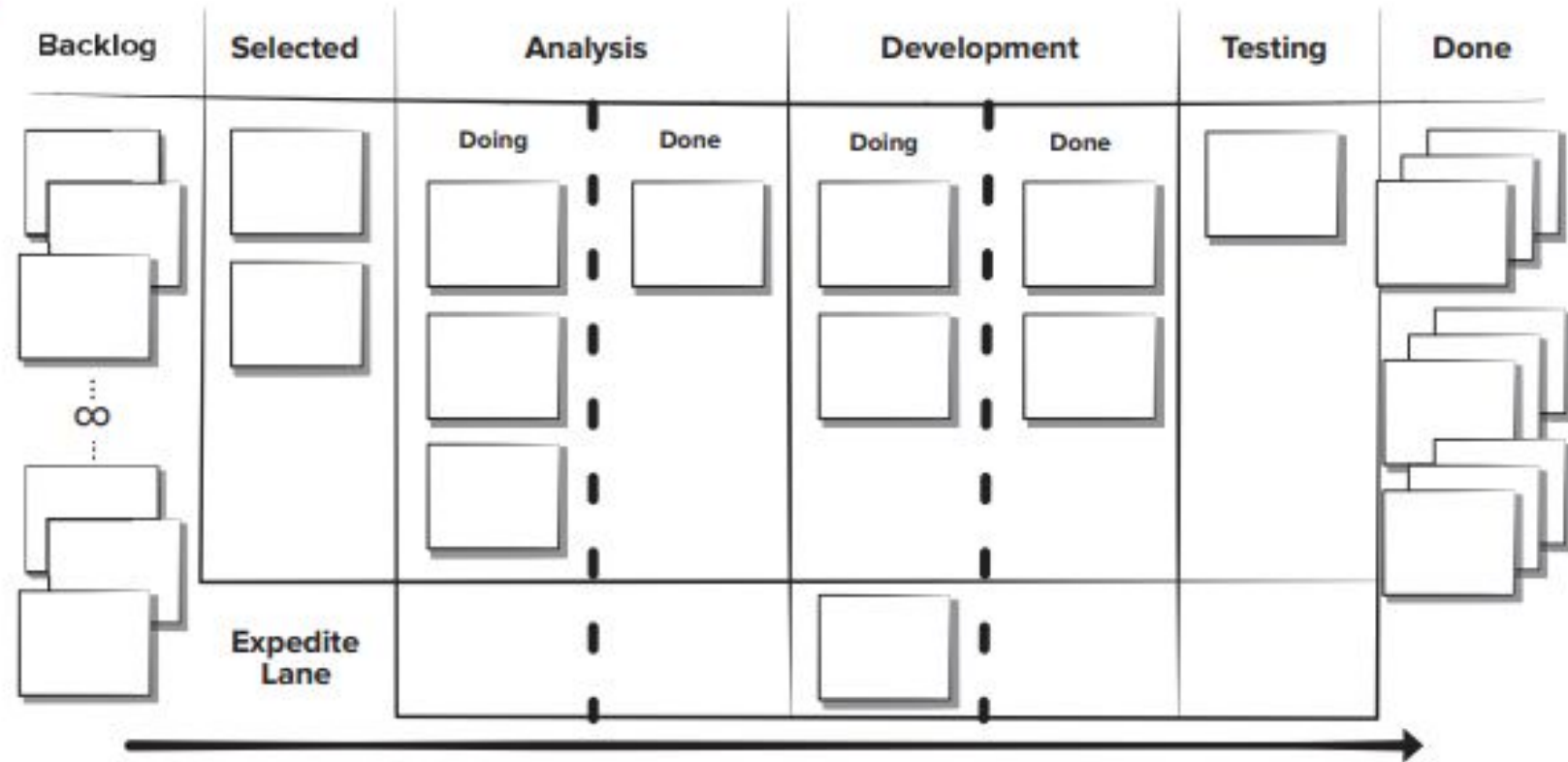
Kanban

- The Kanban designed to improve process/work flow
- Kanban focused on change management and service delivery
- Self organizing teams

Kanban Board

FIGURE 3.4

Kanban board



Kanban

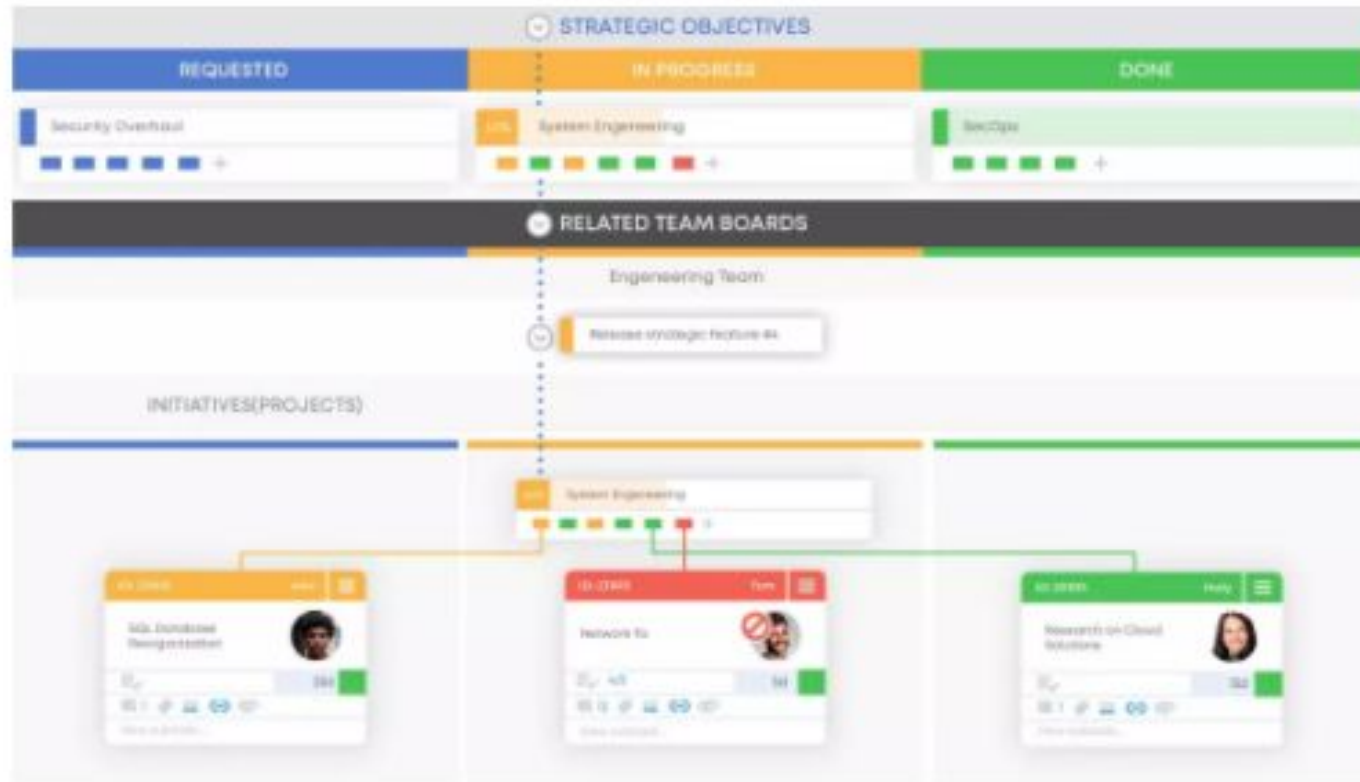
- ❑ The team meetings for Kanban are like those in the Scrum framework.
- ❑ **Visualize processes and workflow:** Visualize all work items and all project, program, and portfolio workflows through a set of inter-connected kanban boards.
- ❑ The basis of the daily Kanban standup meeting is a task called “walking the board.”
- ❑ Leadership of this meeting rotates daily.
- ❑ The team members identify any items missing from the board that are being worked on and add them to the board.
- ❑ **Use flow- and outcome-oriented metrics:** Use relevant metrics to manage workflow and outcomes such as system lead time, cycle time, blocked time, flow efficiency, etc.

Kanban Terminologies

- ❑ **Kanban board:** A Kanban board is one of the Kanban method's key components and is where you visualize all work items. It should be divided into a minimum of 3 columns – Requested, In Progress, Done, representing different process stages.
- ❑ **Kanban card:** Kanban cards represent the different work items moving through a Kanban board. They contain important details about the tasks such as description, deadline, size, assignees, etc.
- ❑ **Columns:** They split the Kanban board vertically, and each of them represents a different stage of the workflow. Each Kanban board has 3 default columns: Requested, In Progress, Done. Depending on the complexity of a work process, these three stages can be divided into many smaller sub-columns.
- ❑ **Swimlanes:** Horizontal lanes that split a Kanban board into sections. Teams use them to visually separate different work types on the same board and organize homogenous tasks together.

Kanban Terminologies

- ❑ **Cycle Time:** Cycle time begins at the moment when a new task enters the “in progress” stage of your workflow, and somebody is actually working on it.
- ❑ **Lead Time:** Lead time starts at the moment a new task is being requested (it doesn't matter if somebody is actually working on it) and ends with its final departure from the system.
- ❑ **Throughput:** The number of work items passing through (completed) a system or process over a certain period. The throughput is a key indicator showing how productive your team is over time.
- ❑ **Work in Progress (WIP):** This is the amount of work you are currently working on and it is not finished yet.
- ❑ **WIP limits:** Limiting work in progress means limiting the number of tasks your team can work on simultaneously to avoid overburdening and context switching.



Digital Kanban Board Example



Thank you